2018

State of DevOps Report

Presented by:

puppet + **splunk>**

Sponsored by:

aws   **cloudability**   **CYBERARK**

Cognizant   **diaxion**
IT Strategy & Optimisation

EFICODE ROOT
THE **DEVOPS** PLATFORM

Contents

Executive summary
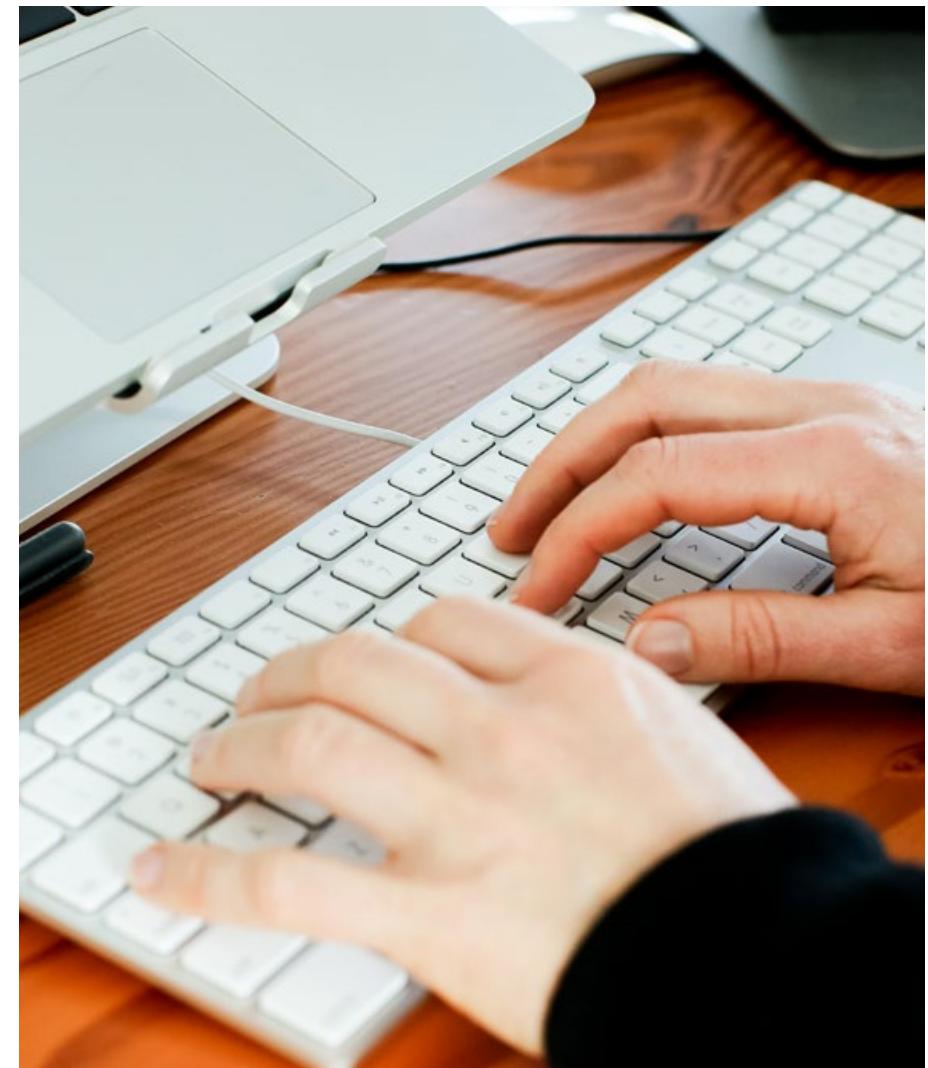
Over the past seven years, we've surveyed more than 30,000 technical professionals around the world to explore the relationships between IT performance, DevOps practices, culture, organizational performance and other elements that affect business outcomes. In the process, we've built the deepest and most widely referenced body of DevOps research available.

The 2018 State of DevOps Report breaks new ground in our understanding of the DevOps evolutionary journey. We have identified the five distinct stages of DevOps evolution, and the critical practices at each stage that help you achieve success and progress to the next phase of your journey.

In 2012, our research on IT performance broke new ground and paved the way for a fresh conversation between IT operations, development teams and the business. We were able to show that the traditional view of IT as a cost center is inaccurate: IT is a powerful driver of value in a world where speed, agility, security and stability are business imperatives.

Our subsequent reports, created in partnership with the team at DevOps Research and Assessment (DORA[1]) have shown how much progress the industry has made over the years, and how much work still lies ahead. The idea that you can increase throughput while simultaneously improving the resilience of the system — an important goal of many DevOps initiatives — is no longer new. Yet for all the words written about DevOps, no one has provided a pragmatic prescriptive approach to DevOps — until now.

The 2018 State of DevOps Report once again breaks new ground in our understanding of DevOps. This year we have quantified the DevOps journey, identifying stages of evolution and the prescriptive steps that will help you progress on this journey. Whether you manage systems, write code, manage teams or departments, the guidance our research provides will help you achieve success faster.

---

1   DORA was founded by Dr. Nicole Forsgren, Jez Humble and Gene Kim

We began with the hypothesis that every DevOps journey has distinct stages and that specific practices can accelerate successful DevOps adoption. Our second hypothesis was that the most successful DevOps journeys start as a ripple in the pond, then radiate out across the business. Individual teams see early success; that success spreads to multiple teams, then through a department, and finally out to multiple departments.

Why this report now? Because our industry needs it. While we've all made a lot of progress and many organizations have achieved early success, most still haven't been able to broadly replicate and scale that success. We've seen far too many teams whose DevOps journeys began eight or nine years ago, and who have experienced many starts and stops along the way. These teams tell us they feel they're still at the beginning of their journey, and wonder why they haven't made more progress.

It doesn't have to be this way. DevOps practices and tools are now at a more mature stage, and enough teams have shown DevOps success to prove it's not a fluke. There are in fact known stages of DevOps and specific practices that lead to success.

If you're just starting out, this report can help you achieve success faster. And if you're in the middle of your journey and feeling stuck, our findings can help you get back on track and scale your success.

Key findings

In a DevOps evolution, there are many paths to success, but even more that lead to failure.

Every organization is different and for most, the DevOps journey isn't linear. There are many starts and stops along the way, which can kill early momentum and lead to cynicism. Without a prescriptive path forward, it's not surprising that organizations are struggling to scale their DevOps success beyond isolated teams.

The question is, how do you foster DevOps so you can scale success across the business? If you're stuck, how do you get back on track and continue building momentum? DevOps is an ongoing evolution, and there is no final destination. But there are ways to achieve success faster. We've identified the five stages in a DevOps evolution and the key practices that will help you advance to the next stage in your journey.

Executives have a rosier view of their DevOps progress than the teams they manage.

For nearly every DevOps practice, C-suite respondents were more likely to report that these practices were in frequent use. Because the C-suite relies on upwards communication — often filtered and sanitized by the time it reaches them — executives don't see the bottlenecks and broken processes that are stalling progress. So they have an incomplete understanding of DevOps progress and impact.

For example, 64 percent of C-suite respondents believe security teams are involved in technology design and deployment versus 39 percent at the team level. The best way to get everyone on the same page is through the mutually reinforcing DevOps pillars of automation and measurement. Automated systems enable better reporting of metrics that can be shared across the business.

Start with the practices that are closest to production; then address processes that happen earlier in the software delivery cycle.

We are often asked "Where do we start?" We recommend starting where the pain is most acute and visible, which is typically application deployments — the boundary between Dev and Ops. Let's face it: You're not going to magically fix your organization's culture overnight. But you can start by improving collaboration (and results) across this one critical functional boundary.

Cross-team sharing is key to scaling DevOps success.

We discovered that the foundational practices — the practices with the most significant impact across the entire DevOps evolutionary journey — are dependent on sharing, one of the key pillars of DevOps. Organizations that have small pockets of DevOps success, yet never manage to spread that success further, are stalled and can't progress to higher levels of automation and self-service. So the business impact of their DevOps success may not be felt where it matters.

To ensure you can scale your early success, prioritize the building blocks that can be reused and consumed across teams, such as deployment patterns. Promoting reuse of successful patterns, enabling teams to contribute improvements to other teams' tooling, and sharing both successes and failures are all critical to expanding the other three pillars of DevOps: culture, automation and measurement.

Automating security policy configurations is mission-critical to reaching the highest levels of DevOps evolution.
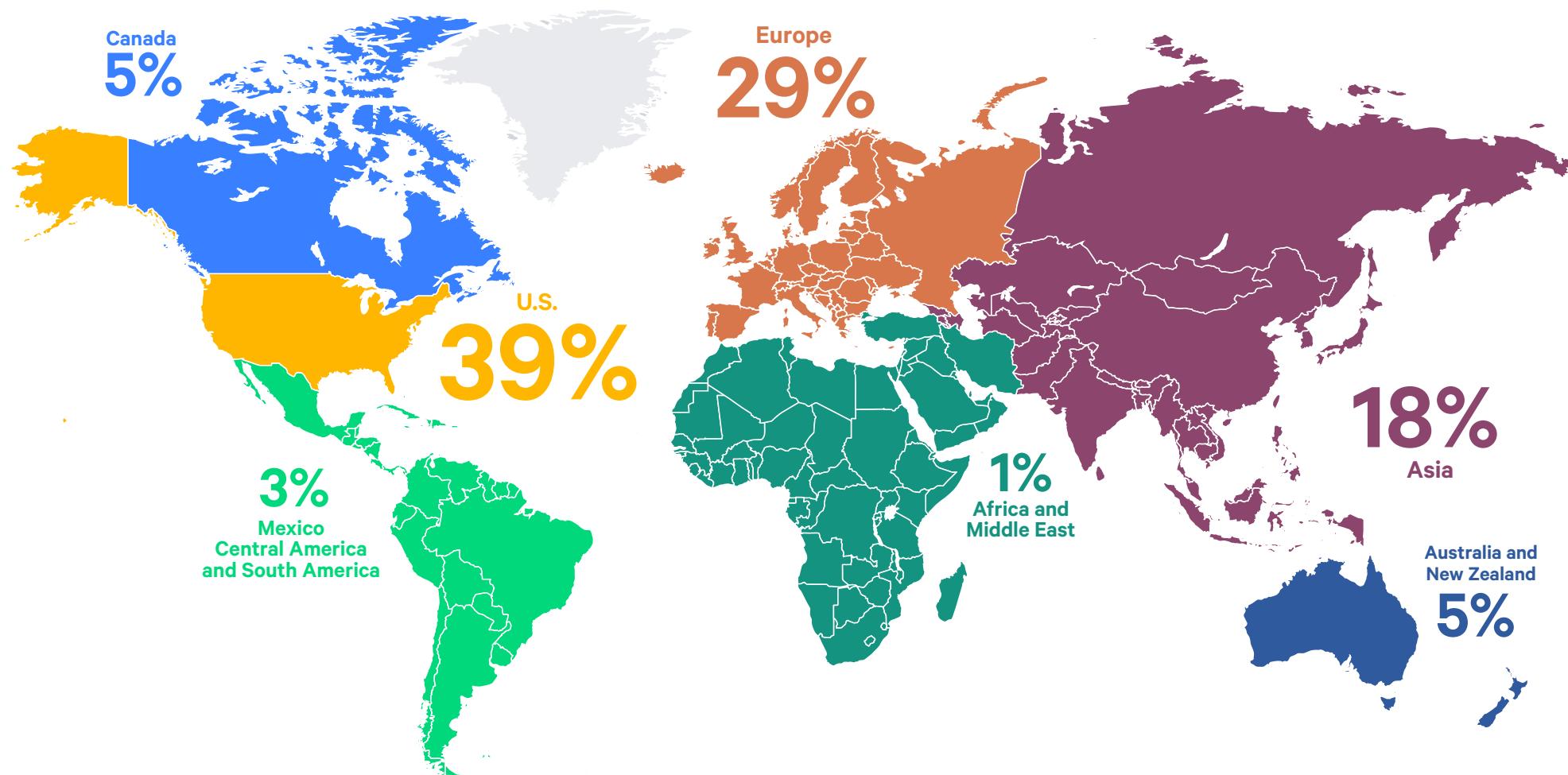
Highly-evolved organizations are 24 times more likely to always automate security policy configurations compared to the least evolved organizations. As organizations evolve, security policy becomes part of operations, not just an afterthought when an audit looms. This requires first breaking down boundaries between ops and security teams (which are further from production). As we see with all the fundamental practices of DevOps, this practice evolves from resolving immediate pain to a more strategic focus — in this case, from "keep the auditors off my back" to "keep the business and our customers' data secure." In other words, teams automate security policy configurations initially for their own benefit, and as their understanding evolves, the automation evolves to benefit the entire organization.

Who took the survey
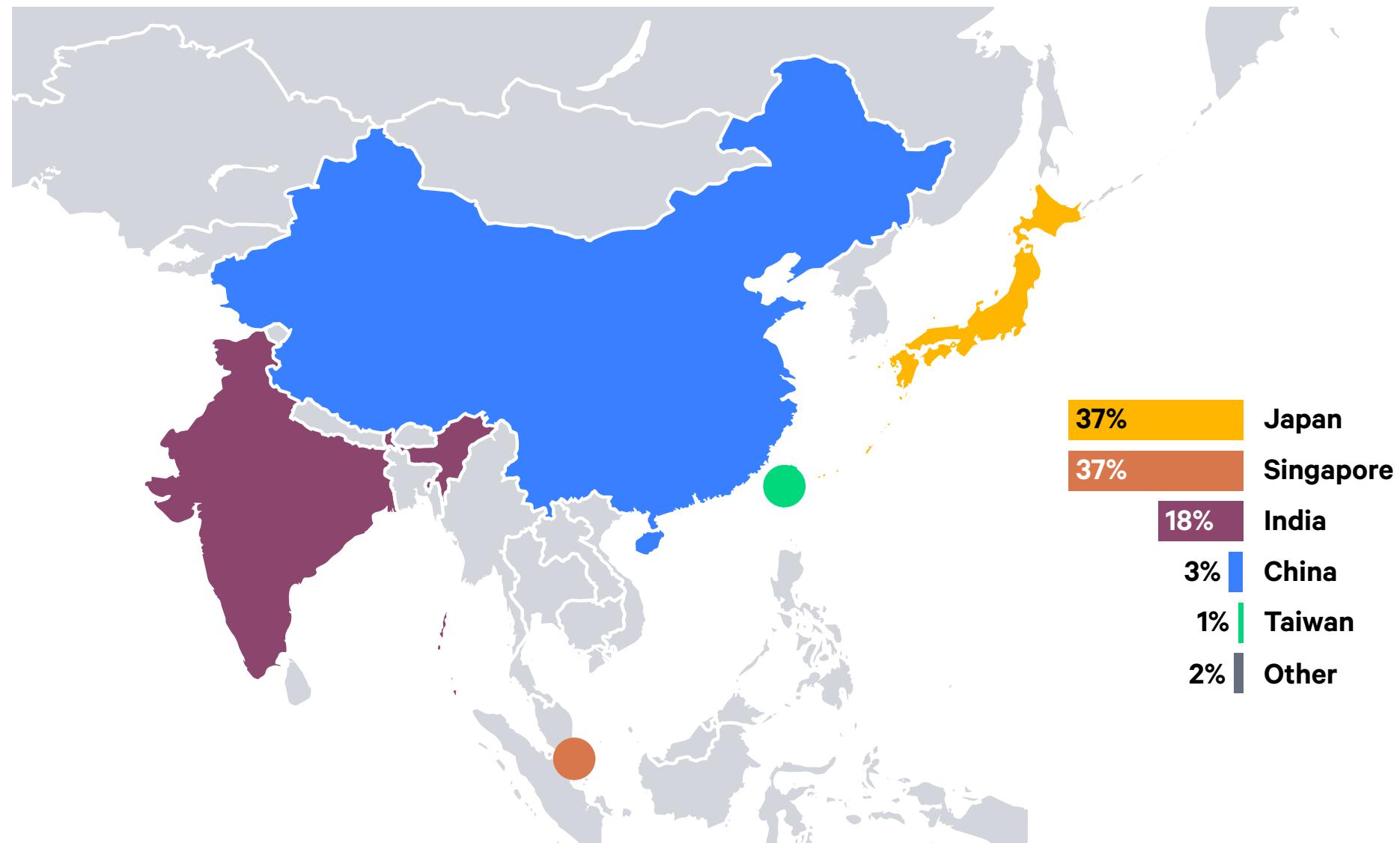
A key difference in this year's report is better global representation. Based on anecdotal evidence, we believe that different geographic regions demonstrate different levels of DevOps maturity, so we specifically targeted respondents outside of North America to ensure greater representation of organizations beyond the United States. This year, we offered the survey in four languages besides English: French, German, Japanese and Malay. These languages cover regions where we are seeing high interest in DevOps.

Responses by global region

Canada
5%

Europe
29%

U.S.
39%

18%
Asia

3%
Mexico
Central America
and South America

1%
Africa and
Middle East

Australia and
New Zealand
5%
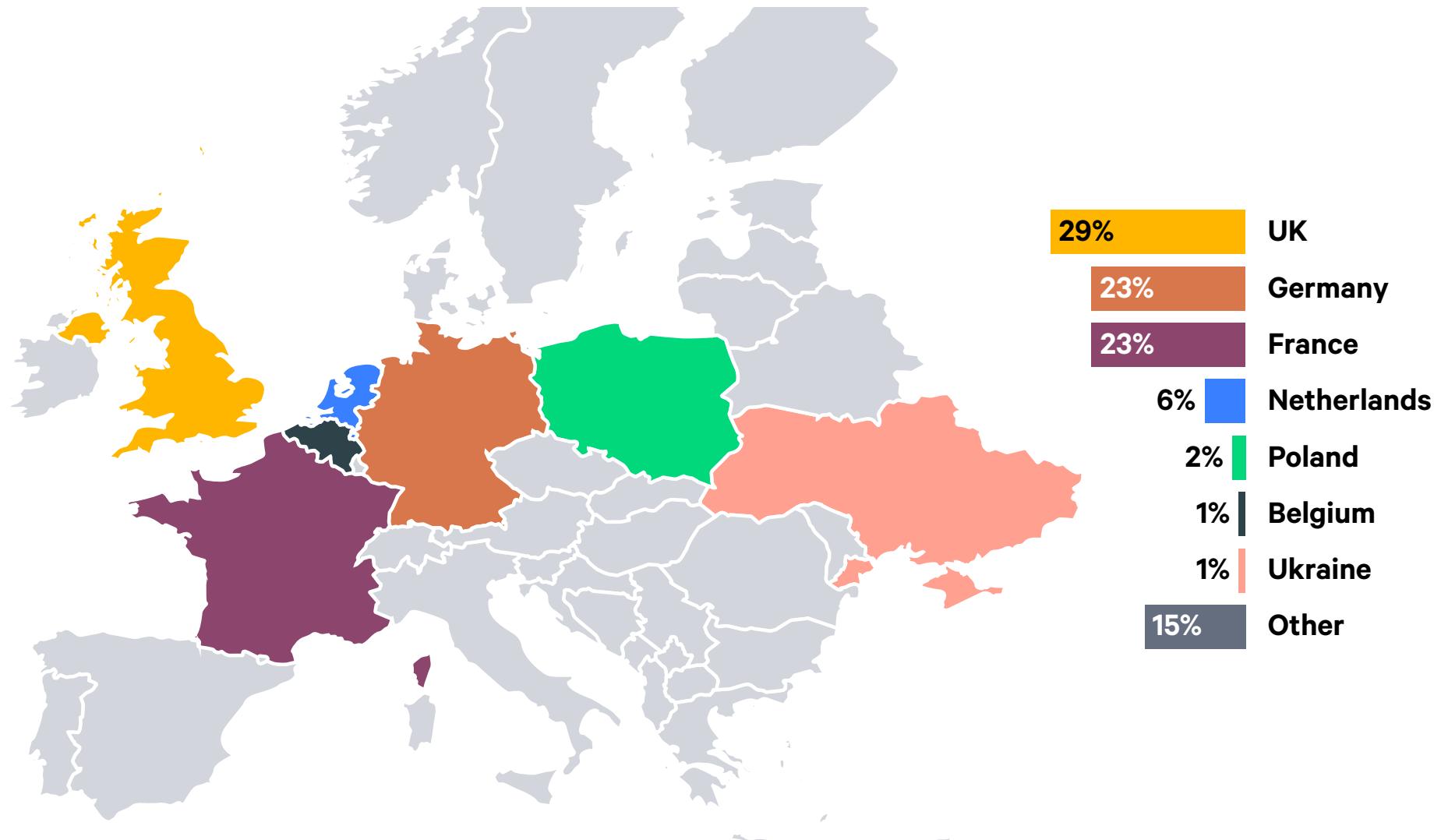
Asia by country

This year, 18 percent of survey respondents were from Asia.
To increase representation from Asia, we offered the survey in Japanese and Malay.

|	
37%	Japan
37%	Singapore
18%	India
3%	China
1%	Taiwan
2%	Other
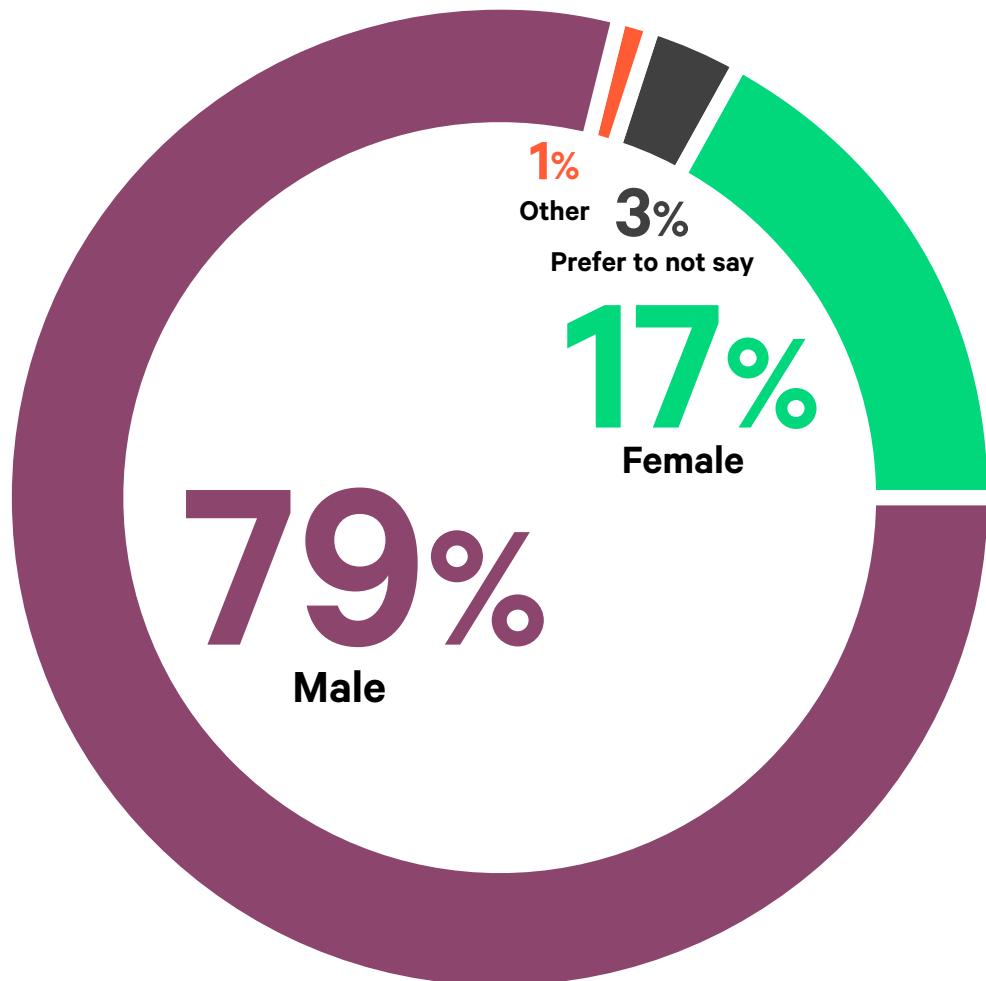
Europe by country

This year, 29 percent of survey respondents were from Europe.
To increase representation from Europe, we offered the survey in French and German.

29%	UK
23%	Germany
23%	France
6%	Netherlands
2%	Poland
1%	Belgium
1%	Ukraine
15%	Other

Gender identity
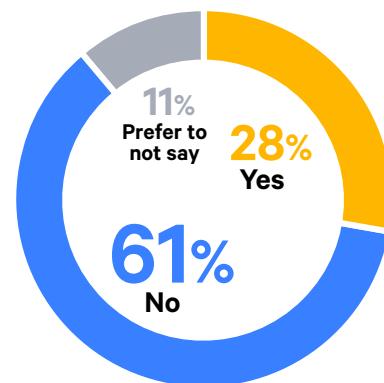
This year 17 percent of respondents were female, up from just 6 percent last year.

1%
Other

3%
Prefer to not say

17%
Female

79%
Male
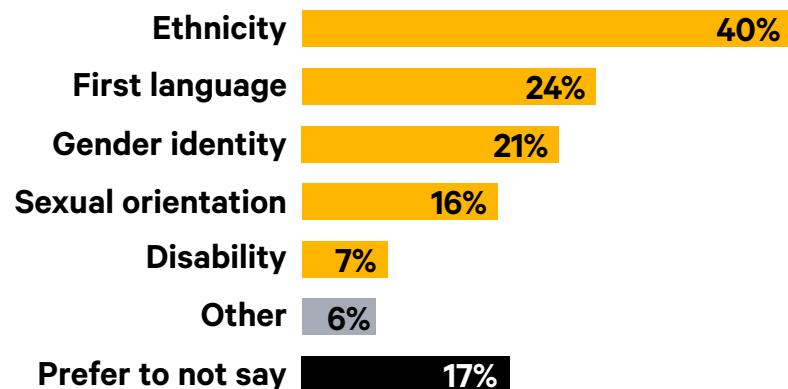
Minority status: visible/invisible

We asked respondents if they considered themselves a member of a visible or invisible minority. Twenty-eight percent responded "yes." Of those who responded yes, 40 percent identified as a member of an ethnic minority.
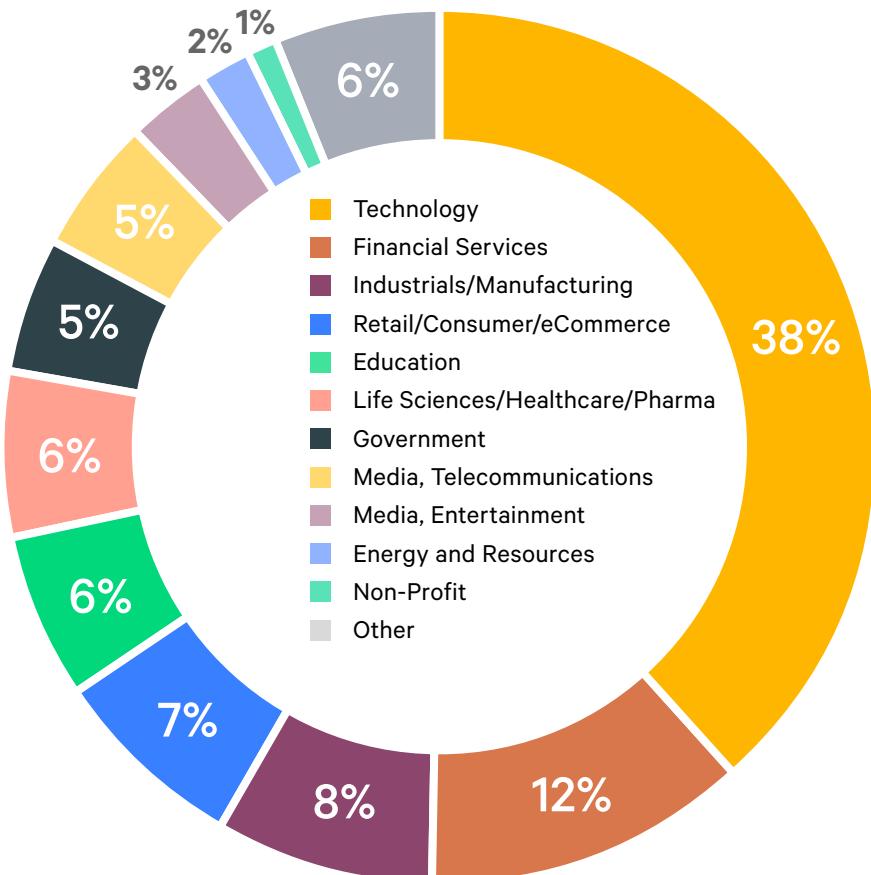
Do you identify as part of a visible or invisible minority in your organization?

11% Prefer to not say

28% Yes

61% No

Minority identity

Ethnicity	40%
First language	24%
Gender identity	21%
Sexual orientation	16%
Disability	7%
Other	6%
Prefer to not say	17%

Principal industry

- 38% Technology
- 12% Financial Services
- 8% Industrials/Manufacturing
- 7% Retail/Consumer/eCommerce
- 6% Education
- 6% Life Sciences/Healthcare/Pharma
- 5% Government
- 5% Media, Telecommunications
- 3% Media, Entertainment
- 2% Energy and Resources
- 1% Non-Profit
- 6% Other
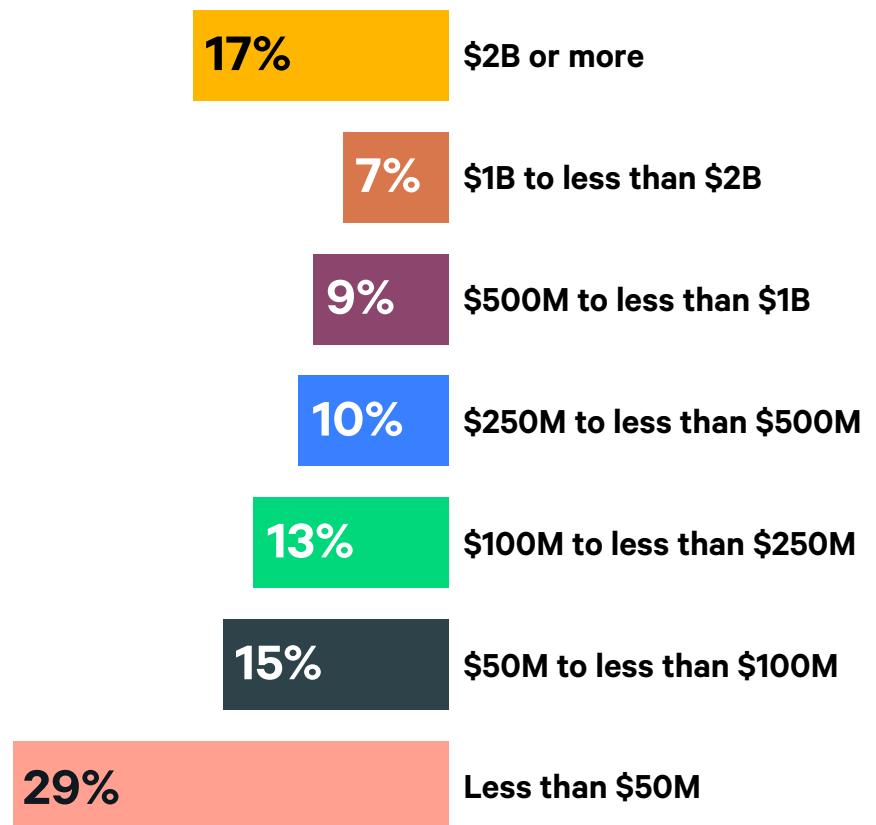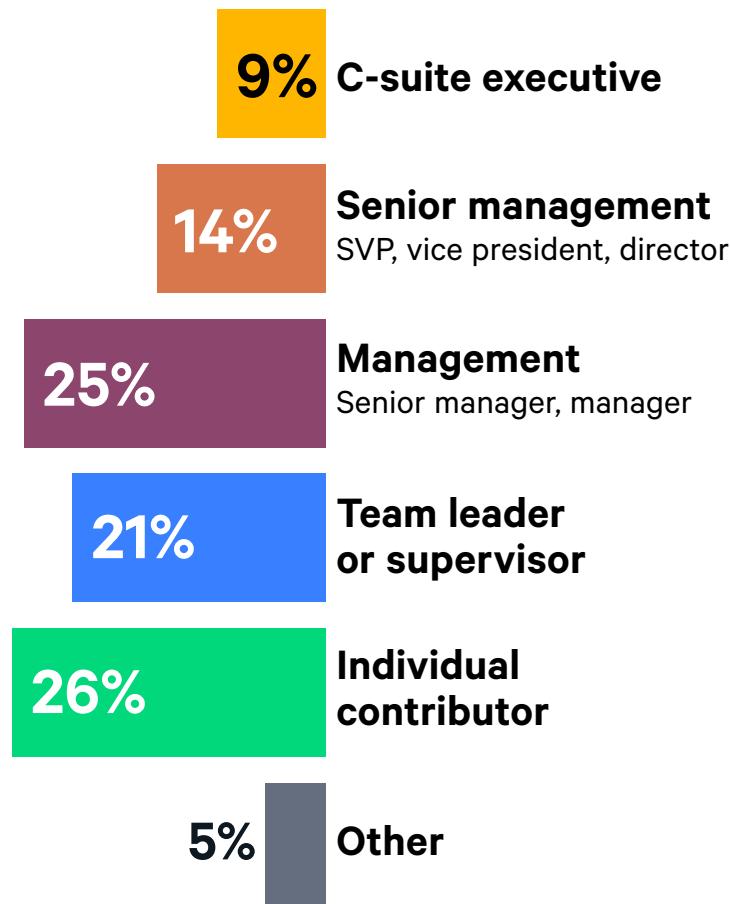
Organization annual revenue

- 17% $2B or more
- 7% $1B to less than $2B
- 9% $500M to less than $1B
- 10% $250M to less than $500M
- 13% $100M to less than $250M
- 15% $50M to less than $100M
- 29% Less than $50M
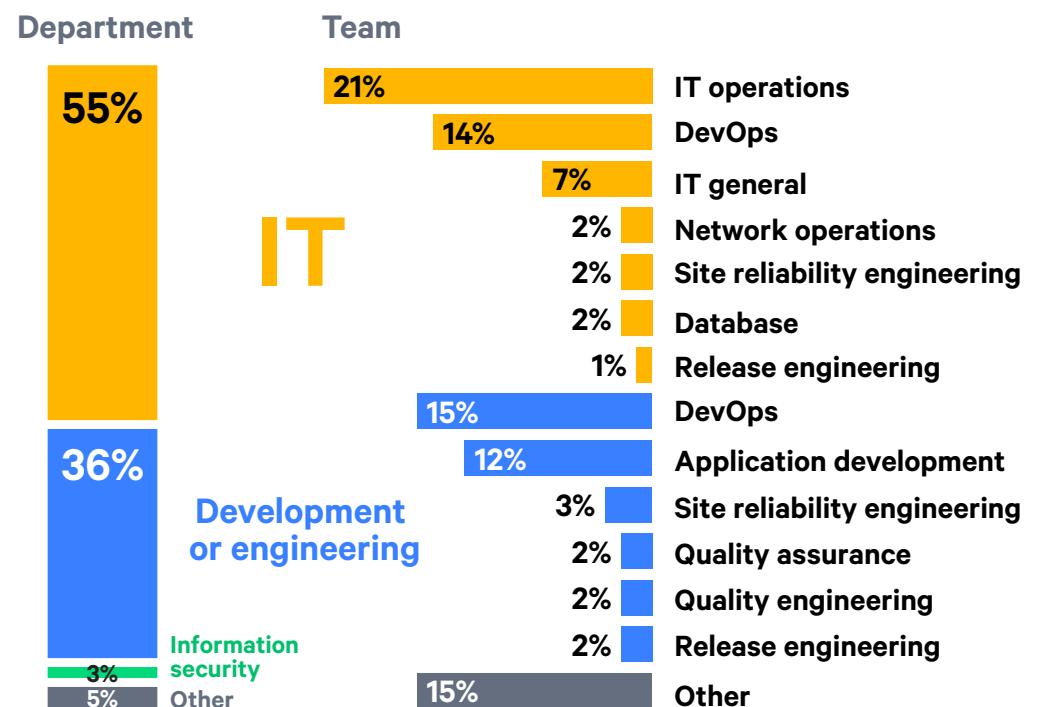
Role within organization

Nearly half (47 percent) of survey respondents are individual contributors or team leaders and 39 percent are in management. Nine percent of respondents said they are in the C-suite.

9% C-suite executive

14% Senior management
SVP, vice president, director

25% Management
Senior manager, manager

21% Team leader
or supervisor

26% Individual
contributor

5% Other

Department and team
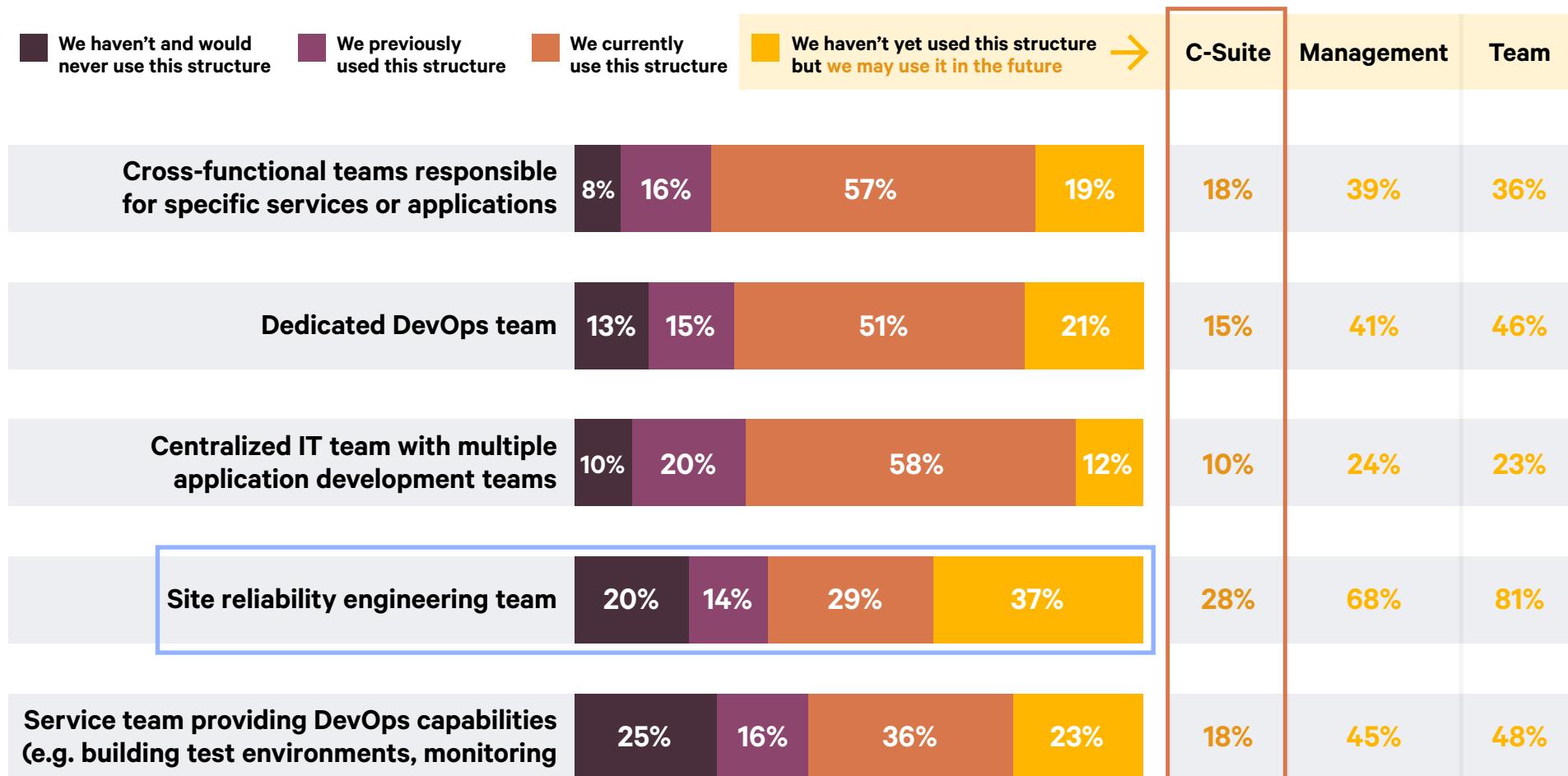
Over half (55 percent) of respondents reported working in an IT department, with 21 percent in an IT operations team and 14 percent in a DevOps team. Another 36 percent of respondents reported working in an engineering or development team, with 15 percent in a DevOps team and 12 percent in an application development team. We've seen a steady increase in survey responses from people on DevOps teams, from just 16 percent in 2014 to 29 percent this year. It's interesting to note that DevOps teams reside equally in both IT and engineering departments.

Department

Team

55% IT

21% IT operations
14% DevOps
7% IT general
2% Network operations
2% Site reliability engineering
2% Database
1% Release engineering

36% Development or engineering

15% DevOps
12% Application development
3% Site reliability engineering
2% Quality assurance
2% Quality engineering
2% Release engineering

3% Information security
5% Other

15% Other
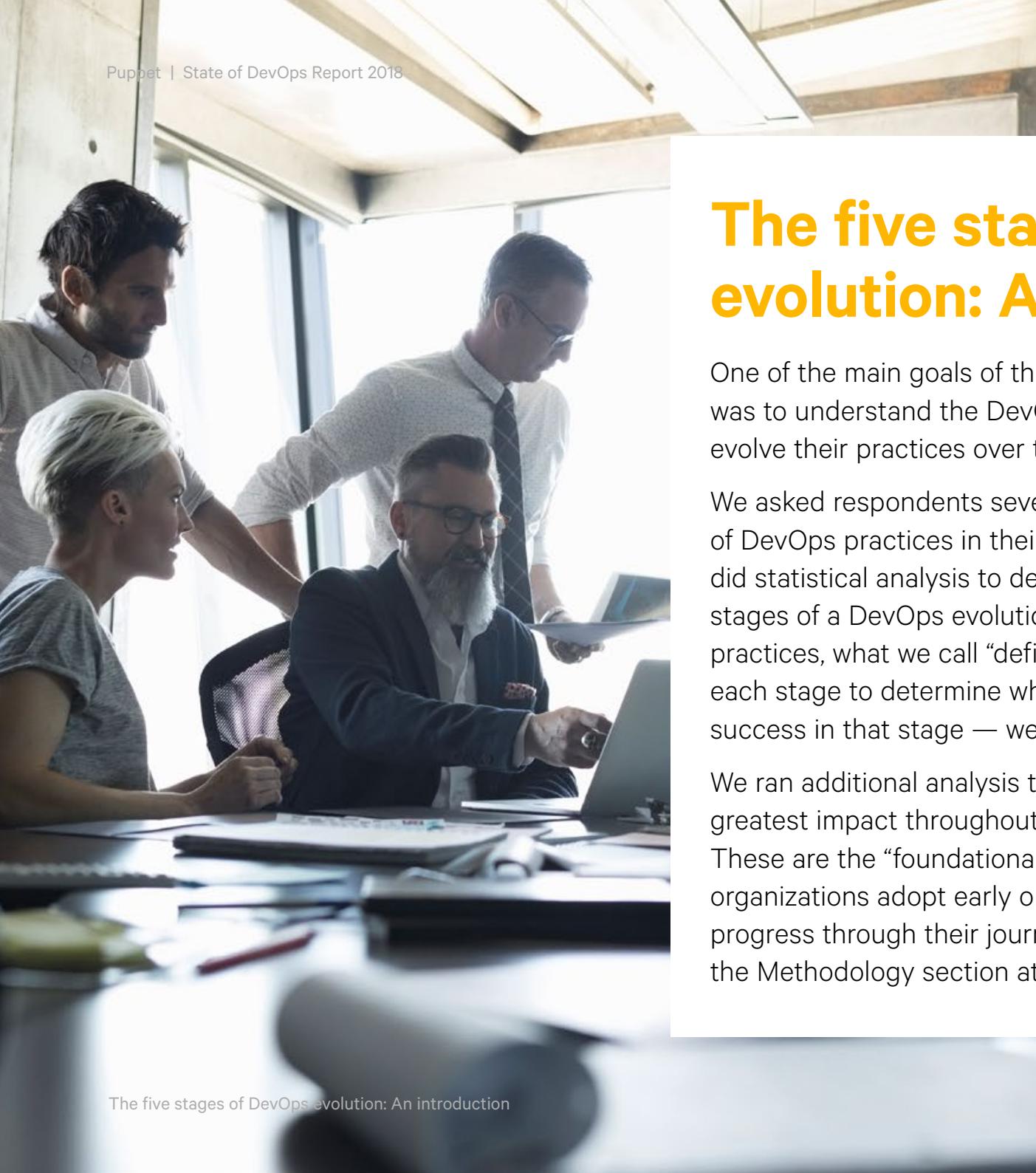
Organizational structures used in DevOps journey

We wanted to understand which organizational structures respondents were currently using or considering for future use. "Cross-functional teams for specific services or applications" and centralized IT teams are the two most widely used structures, followed closely by "dedicated DevOps team."

The lowest-reported organizational structure for current use was "site reliability engineering (Team)" but this structure had the highest percentage of responses for future use. Interestingly, the C-suite was far less likely than managers or team members to consider using any of the structures in the future.

| Structure	We haven't and would never use this structure	We previously used this structure	We currently use this structure	We haven't yet used this structure but we may use it in the future	C-Suite	Management	Team
Cross-functional teams responsible for specific services or applications	8%	16%	57%	19%	18%	39%	36%
Dedicated DevOps team	13%	15%	51%	21%	15%	41%	46%
Centralized IT team with multiple application development teams	10%	20%	58%	12%	10%	24%	23%
Site reliability engineering team	20%	14%	29%	37%	28%	68%	81%
Service team providing DevOps capabilities (e.g. building test environments, monitoring	25%	16%	36%	23%	18%	45%	48%

The five stages of DevOps evolution: An introduction

One of the main goals of this year's State of DevOps Report was to understand the DevOps journey and how organizations evolve their practices over time.

We asked respondents several questions about the frequency of DevOps practices in their organizations. Using this data, we did statistical analysis to determine the practices that define the stages of a DevOps evolution. Each stage is defined by two key practices, what we call "defining practices." We further analyzed each stage to determine which practices most contribute to success in that stage — we call them "contributors to success."

We ran additional analysis to see which practices have the greatest impact throughout the DevOps evolutionary journey. These are the "foundational practices" that highly evolved organizations adopt early on and continue to evolve as they progress through their journeys. For the full methodology, see the Methodology section at the end of this paper.

01 **Normalize the technology stack**

02 **Standardize and reduce variability**

03 **Expand DevOps practices**

04 **Automate infrastructure delivery**

05 **Provide self-service capabilities**

Stage 0: Build the foundation

When development and operations teams (and frequently other stakeholders, such as testing or security) are just starting to grasp the importance of collaboration and sharing, they rapidly implement technologies and processes to facilitate sharing of ideas, metrics, knowledge, processes, and technologies.

This early phase is not a "one and done" stage in a linear DevOps progression. The processes and approaches we identify as foundational are maintained and actually enhanced throughout an organization's DevOps evolution. So this foundational stage is critical to the DevOps evolution, and the health of a successful DevOps organization rests on the base that gets built during this initial stage.

Stage 1: Normalize the technology stack

At this stage, you may see the dev teams making a coordinated move to more agile development methods (e.g., an enterprise-wide Agile mandate), or a few teams organically adopting new methods for specific products or workflows.

Development teams at this stage have adopted version control, which is the first step on the path to continuous integration and continuous delivery. They're also beginning to normalize their tech stacks by eliminating redundant systems, perhaps refactoring applications to work on a smaller set of operating systems.

Stage 2: Standardize and reduce variability

This stage is where both dev and ops teams concentrate on reducing variance, continuing to standardize the tech stack by further reducing the number of operating systems to a single OS or OS family and building on a standard set of technologies: databases, key value stores, message queues, identity stores and more.

We typically see this consolidation happening independently within each team, without much cross-team collaboration. We also see this as an opportunity for early collaboration across teams. For example, if dev teams want to standardize on one database or identity store, they should consult with their ops colleagues, who likely have experience managing all of them and can share operational considerations.

This standardization phase reduces the overall complexity of the system, enabling teams to scale their expertise and apply consistent management and deployment patterns across multiple applications. The benefits are great: You can deploy new applications and services faster, and reduce errors that arise from inconsistency. Best of all, as the shared patterns evolve and improve, the quality of all services improve.

Stage 3: Expand DevOps practices

Now that the important foundational elements are in place, and the system is well understood, organizations can begin to address other pain points. Typically, deployments are a huge source of pain and garner a lot of attention from management when releases are delivered late, or a critical defect makes it to production — and customers notice.

Changes implemented in previous stages have caused application development teams' throughput to outpace the delivery team's ability to deploy. This discrepancy must be addressed quickly, or all the hard work at earlier stages will look like the effort made business outcomes worse, not better.

To resolve this issue, successful teams at this stage reuse deployment patterns for building applications and services, and infrastructure changes are tested before deploying to production. Both these practices provide predictability and reliability, building trust in the new methods and practices. With this new level of trust in the system, important cultural shifts can take place in the organization. For example, individual team members can gain the ability and organizational permission to do work without manual approval from outside the team, eliminating bureaucratic overhead and promoting more efficient workflows.

Stage 4: Automate infrastructure delivery

This stage in the DevOps journey is defined by the automation of systems configuration and provisioning, which many people consider to be a high-priority outcome of a DevOps initiative. Automating infrastructure delivery resolves the issue of developer throughput outpacing operations, and therefore the ability to deploy. Automated system configuration makes it possible for ops teams to deliver systems to developers and QA that match the eventual production environment — and deliver them faster.

Infrastructure automation certainly addresses a local pain point for IT operations teams, but it goes much further than that: It catalyzes the creation of self-service more broadly throughout the organization in subsequent stages. Self-service for multiple departments ultimately leads to greater efficiency and satisfaction throughout the organization.

Stage 5: Provide self-service capabilities

By the time an organization gets to Stage 5, you can see the cumulative effects of achieving high levels of automation and trust. At this stage, resources are available via self-service, and incident response is automated. IT teams don't automate just for the sake of automating; they do it to make the entire organization run with greater efficiency and precision. With the self-service capabilities developed in Stage 4, teams across the business can work at their own pace, freed from the bureaucratic overhead of manual approvals, handoffs, tickets and long wait times. As you can see, departments far beyond IT and development are now able to work more efficiently, benefiting the entire organization.

Foundational practices and the 5 stages of DevOps evolution

	Defining practices* and associated practices	Practices that contribute to success
Stage 0	• Monitoring and alerting are configurable by the team operating the service. • Deployment patterns for building applications or services are reused. • Testing patterns for building applications or services are reused. • Teams contribute improvements to tooling provided by other teams. • Configurations are managed by a configuration management tool.	
Stage 1	• **Application development teams use version control.** • **Teams deploy on a standard set of operating systems.**	• Build on a standard set of technology. • Put application configurations in version control. • Test infrastructure changes before deploying to production. • Source code is available to other teams.
Stage 2	• **Build on a standard set of technology.** • **Teams deploy on a single standard operating system.**	• Deployment patterns for building applications and services are reused. • Rearchitect applications based on business needs. • Put system configurations in version control.
Stage 3	• **Individuals can do work without manual approval from outside the team.** • **Deployment patterns for building applications and services are reused.** • Infrastructure changes are tested before deploying to production.	• Individuals can make changes without significant wait times. • Service changes can be made during business hours. • Post-incident reviews occur and results are shared. • Teams build on a standard set of technologies. • Teams use continuous integration. • Infrastructure teams use version control.
Stage 4	• **System configurations are automated.** • **Provisioning is automated.** • Application configurations are in version control. • Infrastructure teams use version control.	• Security policy configurations are automated. • Resources made available via self-service.
Stage 5	• **Incident responses are automated.** • **Resources available via self-service.** • Rearchitect applications based on business needs. • Security teams are involved in technology design and deployment.	• Security policy configurations are automated. • Application developers deploy testing environments on their own. • Success metrics for projects are visible. • Provisioning is automated.

* The practices that define each stage are highlighted in bold font.

CAMS and the DevOps evolutionary model

One of our driving hypotheses this year was that the vast majority of successful organization-wide DevOps initiatives are built on existing pockets of success within one or more teams, and that conversely, top-down initiatives without prior success at the team level tend to fail.

This hypothesis comes out of the direct experience of the authors of this report. We have all worked with multiple large organizations where we've observed a distinct pattern of success. 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.

This doesn't mean the path to wider DevOps adoption in the organization is always smooth and trouble-free, nor that scaling existing success automatically leads to the entire organization humming along smoothly on a path of continual improvement. DevOps practices are still relatively new compared to the age of most large enterprises, and one should expect it will take significant time, effort and discipline to create change in a large organization.

The importance of effective leadership in a DevOps transformation and the critical role that managers play is discussed in our 2015 State of DevOps Report.

The 2015 DevOps Survey and its resulting database are the exclusive property of Puppet, Inc. and DevOps Research and Assessment, LLC. All rights reserved. Authors: Dr. Nicole Forsgren, Jez Humble, Gene Kim, Alanna Brown, Nigel Kersten

The authors have all seen example after example of major top-down IT-related initiatives — often labeled "DevOps" — that have failed to deliver fundamental improvements such as the ability to deliver IT services faster, with higher levels of quality. Perhaps even more depressingly, it's quite common to hear of major differences in perception between the C-suite and folks on the ground when it comes to evaluating progress. Even when we allow for the pervasive skepticism among practitioners in our industry, and the tendency of executives to present an optimistic view both inside and outside their organizations, we suspect there's a serious disconnect here.

We wanted to measure the outcomes of IT-related initiatives instead of relying on anecdotes, especially in view of the widely differing perspectives that people relating these anecdotes come from. So we turned to the CAMS[2] model, a widely accepted framework for DevOps. Originally coined by Damon Edwards and John Willis, CAMS stands for culture, automation, measurement and sharing.

The elements of these acronyms are, of course, broad categories without formal boundaries. But the CAMS definition has proven to be a useful and workable model over many years, particularly as DevOps itself has evolved into new areas such as mainframes, network operations and security.

---

[2]   itrevolution.com/devops-culture-part-1

Change management as it is traditionally applied is outdated. We know, for example, that 70 percent of change programs fail to achieve their goals, largely due to employee resistance and lack of management support. We also know that when people are truly invested in change, it is 30 percent more likely to stick.

mckinsey.com/featured-insights/leadership/changing-change-management

We hypothesized that if our survey respondents' organizations had delivered concrete progress in these four areas, then they were further along in their DevOps journey. If an organization hadn't managed to achieve anything substantial, then it wasn't very far along.

We asked four questions for each of the CAMS pillars, ranging from adoption within a single team to expansion across multiple departments.

For culture, we asked:

Where would you say you are culture-wise on your DevOps journey so far?
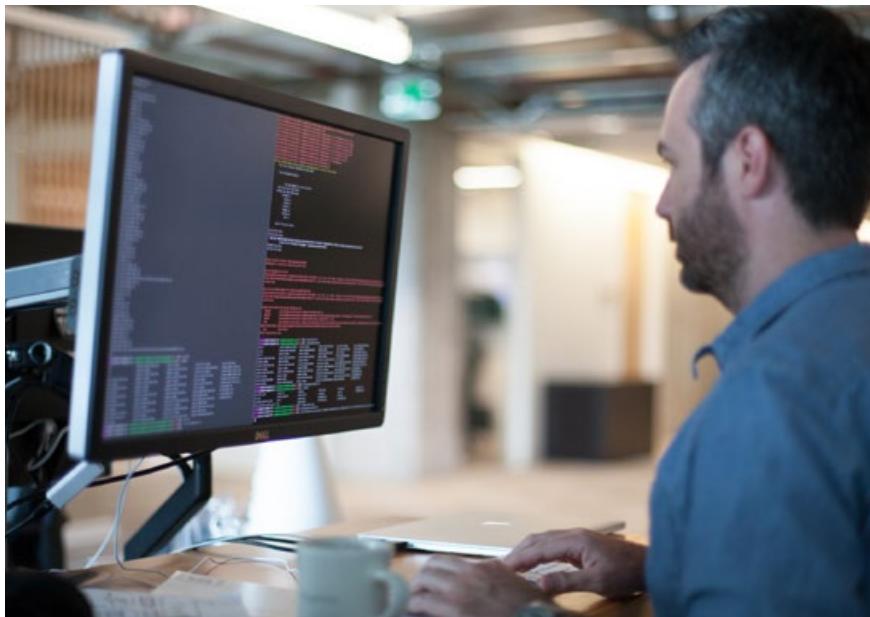
- We have a single team that has a strong DevOps culture.
- We have multiple teams within a department with a strong DevOps culture.
- We have a single department that has a strong DevOps culture.
- We have a strong DevOps culture across multiple departments.

For automation, we asked:

Where would you say you are automation-wise on your DevOps journey so far?

- Teams are automating services they control, for their own needs.
- Teams are automating services they control, for others' needs.
- Teams are collaborating to automate services for broad use.
- A few key services are available via self-service.
- Most services are available via self-service.

When it comes to measurement, we have found that expansion from teams to departments manifests as a shift from manually gathered IT system metrics to automated measurement of business objectives. The most sophisticated teams we've seen not only improved their IT processes and practices, but also managed to focus on delivering business value rather than just technology. These teams have applied their existing cultures of automation and measurement to business objectives.

For measurement, we asked:

Where would you say you are measurement-wise on your DevOps journey so far? Select all that apply.

- We manually measure key system metrics (e.g., computer performance, throughput, etc.).
- We automatically measure key system metrics.
- Business-level objective measurements are manually gathered using system level metrics.
- Business-level objective measurements are automatically available on demand.

For sharing, we asked:

Where would you say you are, sharing-wise, on your DevOps journey so far?

- Patterns and best practices are shared within teams.
- Patterns and best practices are shared across teams.
- Patterns and best practices are shared across the organization.
- Patterns and practices are shared outside the organization.

CAMS and the evolutionary scale
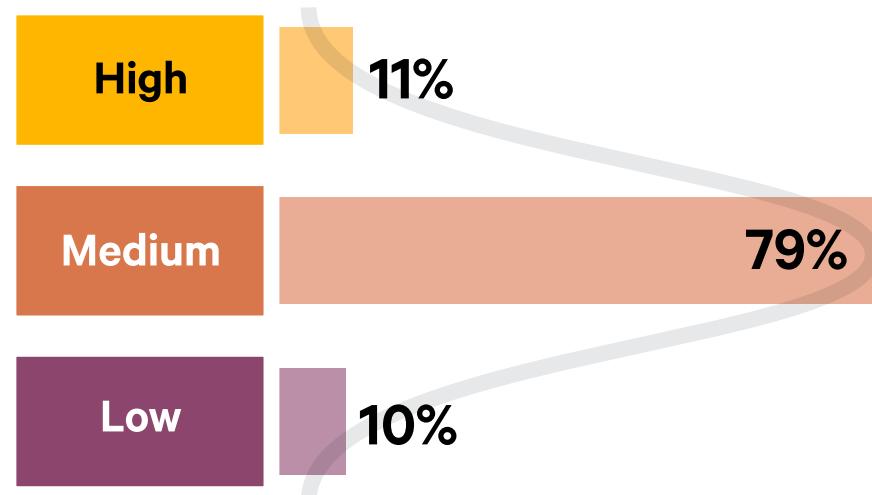
To understand an organization's progress for each of the CAMS pillars, we developed a model to measure each organization's position on an evolutionary scale. To create the evolutionary scale, we scored responses based on how frequently the respondent's organization was doing each practice (1 = Never, 2 = Rarely, 3 = Sometimes, 4 = Most of the time, 5 = Always). We then summed these scores to create a composite score. Based on this composite score, we then grouped those responses into three categories: Low, Medium and High. Organizations that are employing all the practices with a high frequency are highly evolved, or High. Those organizations employing practices with low frequency are Low, and those doing some practices sometimes are Medium. See the *Methodology* section for a detailed explanation.

Ninety percent of respondent organizations are at least Medium. Almost 11 percent are Low and just under 10 percent are High. This tells us that DevOps practices have become mainstream, and that it's much harder to make the leap from Medium to High than it is from Low to Medium. From this, we extrapolate that organizations can gain a serious competitive advantage if they concentrate on further evolving their DevOps practices.

We suspect that people can get to Medium status with less effort because the automation path is relatively well defined, but the jump to High requires implementation of DevOps culture and sharing, which are more difficult to grasp and instill.

How does an organization's evolutionary progress correspond to CAMS? We found that the highly evolved organizations (High) have expanded DevOps culture and practices across multiple teams and departments, and that the least evolved organizations (Low) have not.

Percentage of respondents by evolutionary scale

High   11%

Medium   79%

Low   10%

Culture

We see this pattern of expansion quite clearly for culture. The highly evolved organizations had the lowest number of responses to "We have a single team that has a strong DevOps culture" and the highest number of responses to "We have multiple teams within a department that have a strong DevOps culture."

Automation

The results around automation aren't as clear, but past experience has shown us that the path from a low degree of IT automation to a high degree isn't neat or linear. As you automate more and more services, expanding outwards from the core responsibilities of a single team, not only do you discover more services that could be automated, but you begin to deal with services that are increasingly difficult to automate. That's because services that span different functional areas of the business have not only more dependencies, but more complex ones, so automating them is correspondingly more complex — and more expensive.

Cultural progress by evolutionary scale
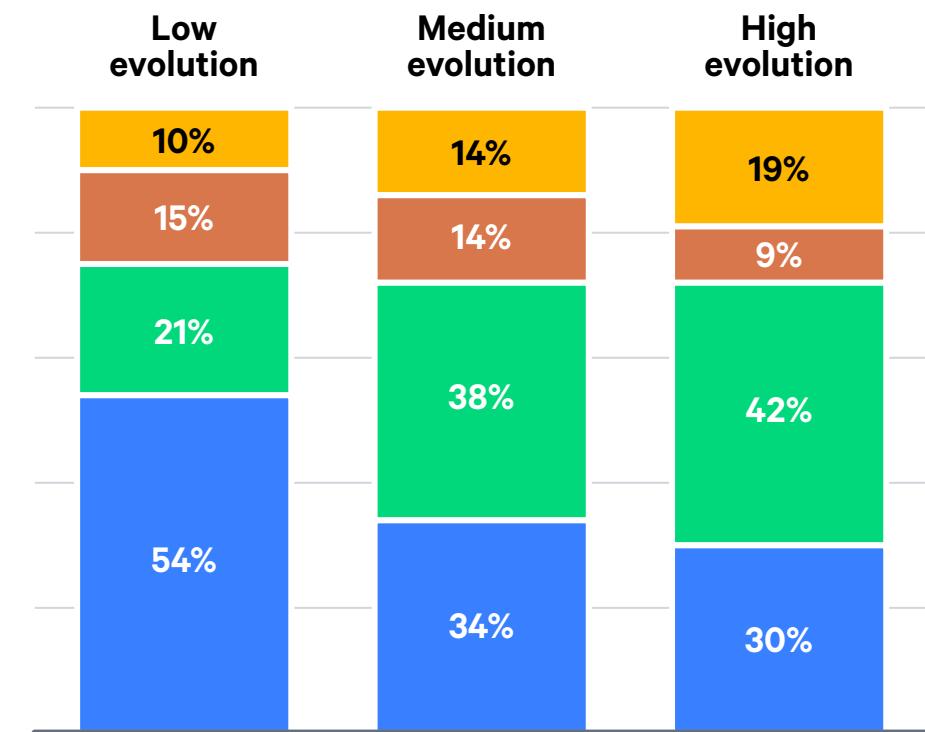
We have a strong DevOps culture …

- 🟧 … across multiple departments.
- 🟫 … across a single department.
- 🟩 … across multiple teams within a department.
- 🟦 … across a single team.

| Low evolution	Medium evolution	High evolution
10%	14%	19%
15%	14%	9%
21%	38%	42%
54%	34%	30%
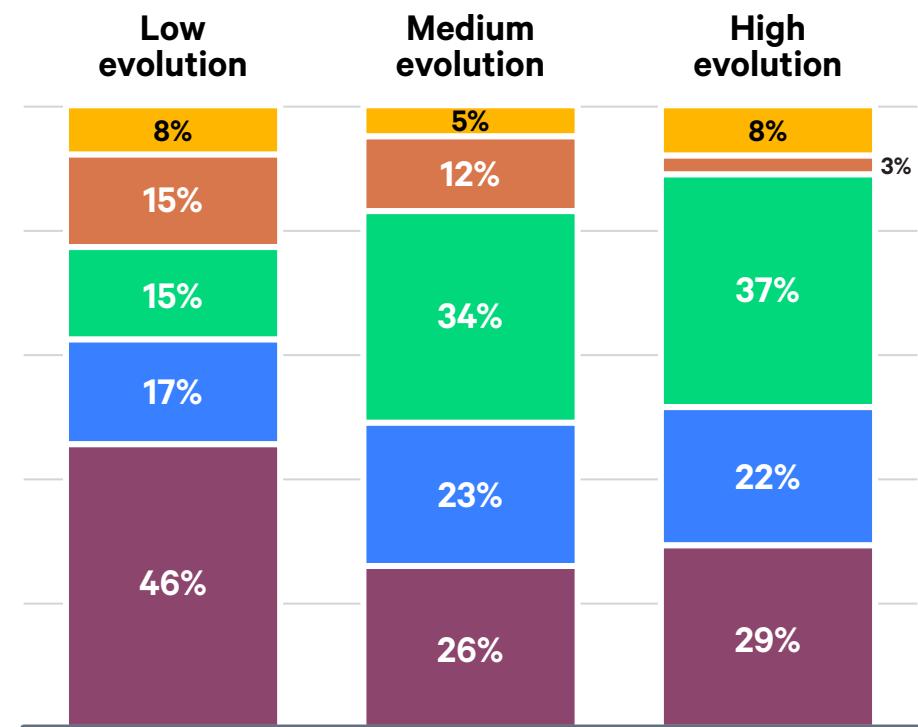
We do see a distinct improvement from Low organizations to Medium and High ones when it comes to teams collaborating to automate services that are broadly consumed. We also see a corresponding drop in the relative proportion of services that are automated for internal team consumption only.

Our hypothesis is that the minimal difference in degree of automation between the Medium and High cohorts reflects the fact that automation is arguably the easiest-to-implement pillar of the CAMS model. Automation is well understood by technical people, has a relatively predictable path, and can succeed if you give disciplined technical practitioners the time and bandwidth to automate, along with a mandate to do so.

DevOps, however, is not just about automation. The cultural changes that are required for DevOps success are significantly more difficult to implement and require broader organizational input and support. It's also harder to measure the outcomes in terms the business can understand. And where it's pretty easy to find leading examples for automation that you can emulate, it's harder to port one organization's cultural-evolution experience directly to another organization and get a successful outcome.

Automation progress by evolutionary scale

- 🟨 **Most services are available via self-service.**
- 🟧 **A few key services are available via self-service.**
- 🟩 **Teams collaborate to automate services for broad use.**
- 🟦 **Teams automate services they control, for others to use.**
- 🟪 **Teams automate services they control, for their own use.**

| Low evolution	Medium evolution	High evolution
8%	5%	8%
15%	12%	3%
15%	34%	37%
17%	23%	22%
46%	26%	29%
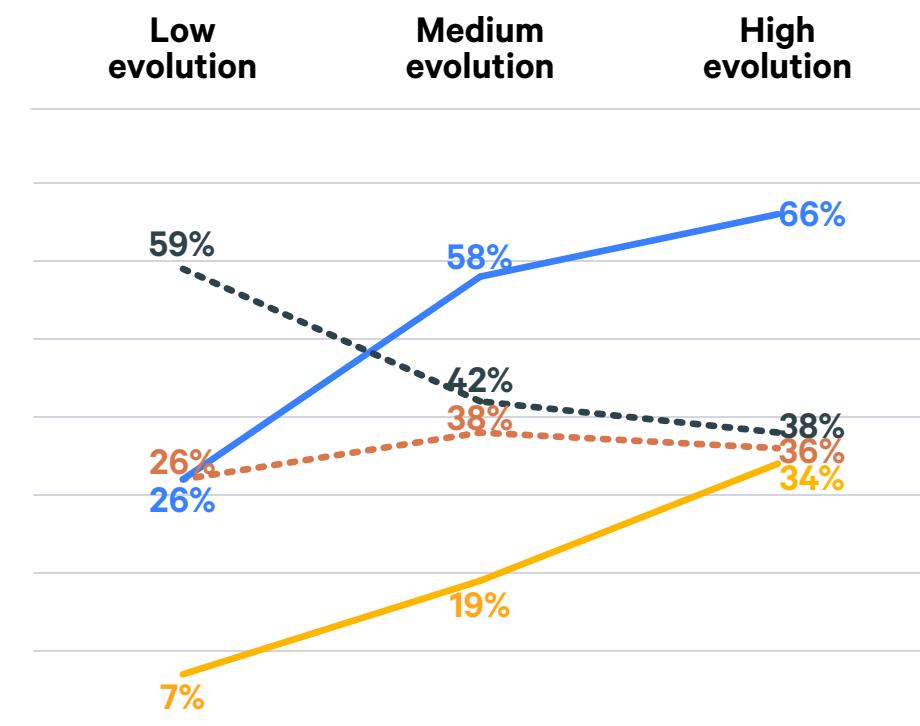
Measurement

Our hypothesis for measurement was that a more mature DevOps organization would be more likely to have automated a collection of key metrics, and that those metrics would primarily speak to business objectives. We found this to be true. Highly evolved organizations have significantly higher levels of automated business metrics that are available on demand, as well as the highest level of automated system measurement. The High organizations also have the lowest level of manually gathered system metrics.

All three groups — High, Medium and Low — had about the same proportion of manually gathered business metrics. We suspect this is due to the fact that there's typically a standard set of business metrics that people are used to collecting manually — whether they're doing DevOps or not — such as revenue, renewal rates, customer acquisition costs, overhead costs and variable cost percentages. These may have automated elements, but tying together data from disparate systems is often done manually. And automation can get difficult quickly when you have, for example, one system that starts each week on Monday, and a related system that starts the week on Sunday. It can also be difficult to decide when an order actually occurred in a company that spans time zones or has complex procurement processes.

Measurement by evolutionary scale

- - - - **We manually measure key system metrics.**

───── **We automatically measure key system metrics.**

- - - - **Business-level measurements are manually gathered using system level metrics.**

───── **Business-level measurements are automatically available on-demand.**

Low evolution	Medium evolution	High evolution

59%
58%
66%
42%
38%
38%
38%
36%
26%
26%
34%
19%
7%

Different roles, different perspectives on progress

As we looked at how different roles — C-suite, management, individual contributor — viewed their DevOps progress, we found that the C-suite had a much more optimistic outlook. For nearly all practices, the C-suite reported higher frequency of use, in some cases with very wide discrepancies. For example, 54 percent of the C-suite reported that security policy configurations are automated, compared to 38 percent at the team level. Fifty-seven percent of the C-suite respondents reported that incident responses were automated compared to 29 percent at the team level.

It's tempting to blame the C-suite for being out of touch, but it's important to keep in mind that upward communications are often filtered and sanitized, contributing to C-suite optimism.

The best countermeasures to this inaccurate communication are the mutually reinforcing pillars of automation and measurement. Automated systems enable better reporting of business metrics. Rather than relying on information that's filtered upwards to executives, you have an objective measurement system to share across the business, helping everyone get onto the same page.

Differences in perception of DevOps practices in use

|	C-Suite	Management	Team
Teams contribute improvements to tooling provided by other teams	64%	46%	35%
We balance lowering technical debt with new feature work	61%	44%	33%
Incident responses are automated	57%	38%	29%
Security teams are involved in technology design and deployment	64%	48%	39%
A cross-functional review is done before implementation of a project	58%	47%	36%
Experiences and lessons are shared externally (e.g., meetups / conferences, blog posts, etc.)	49%	38%	28%
Success metrics for projects are visible	58%	46%	38%
Rearchitect applications based on business needs (e.g., reduce operational costs, ease of deployment, etc.)	57%	46%	37%
Resources (e.g., accounts, infrastructure, etc.) made available via self-service	53%	42%	34%
Before starting a project, we establish concrete success criteria	61%	51%	43%
Service changes can be made during business hours	61%	46%	43%
Teams use continuous delivery	58%	47%	41%
We create learning opportunities across teams (e.g., training, internal DevOps workshops, etc.)	54%	48%	38%
Automate security policy configurations	54%	44%	38%
We have post-incident reviews and share results	64%	56%	48%

Sharing
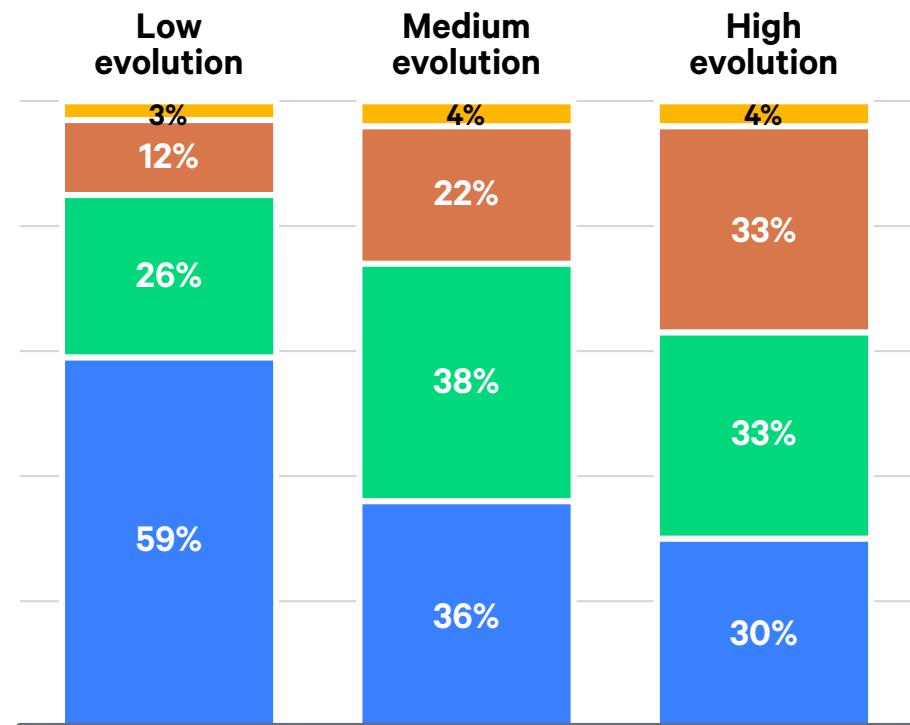
As DevOps evolution advances, we see sharing within the organization increase — an expected outcome. Highly evolved organizations move from sharing largely within individual teams to sharing widely across the entire organization. We have found that this sharing of best practices and patterns tends to go hand in hand with higher degrees of well-designed automation. It makes sense: As automation systems evolve and consist of increasingly high-level abstractions that are more integrated with the rest of your systems, it becomes progressively easier to share automation and deployment patterns. Deployment and consumption of these abstracted automation systems becomes simpler and faster.

Sharing outside the organization is negligible for all cohorts, which is a missed opportunity. It's still too hard for enthused teams in many enterprises to share their patterns and tooling with the rest of the world, with their organization's blessing. The barriers are seldom technical. For example, a team may have to jump through legal hoops to use open-source some software, or get approval from the company's public relations team to share a case study or speak at an event.

Sharing by evolutionary scale

Patterns and best practices are shared …

- 🟨 **… outside the organization.**
- 🟫 **… across the organization.**
- 🟩 **… across teams.**
- 🟦 **… among individuals within teams.**

	Low evolution	Medium evolution	High evolution
… outside the organization.	3%	4%	4%
… across the organization.	12%	22%	33%
… across teams.	26%	38%	33%
… among individuals within teams.	59%	36%	30%

Benchmarking and CAMS

CAMS is a useful model for benchmarking your organization's evolutionary progress.
Our data shows that highly evolved organizations:

- Have a DevOps culture that spans multiple departments.

- Have automated more services for broad use.

- Automate more measurement for business objectives.

- Share patterns and best practices broadly across their organizations.

In the next chapters, we'll dive deeper into each stage of the DevOps journey.

Every organization starts from its own unique place. It has legacy technologies, established ways of doing things, its own specific business mission and its own particular culture. So there is no single path to a DevOps transformation; instead, there are many possible evolutionary paths.

Our research revealed five stages of DevOps evolution and also a set of foundational practices that are critical throughout a DevOps evolutionary journey. These practices evolve as organizations evolve and are all dependent on sharing.

Stage 0: Build the foundation

Analysis of the data from the 2018 State of DevOps survey revealed the foundational practices that successful teams employ. These practices correlate so strongly with DevOps success, we've determined they are essential at *every* stage of DevOps development. In other words, the practices that must be adopted at any given stage in order to progress to the next stage remain important even for those organizations that have evolved the furthest on their DevOps journey, and that have already showed the most success.

Each foundational practice can be described in a sentence:

- Monitoring and alerting are configurable by the team operating the service.
- Deployment patterns for building applications or services are reused.
- Testing patterns for building applications or services are reused.
- Teams contribute improvements to tooling provided by other teams.
- Configurations are managed by a configuration management tool.

When we examined each of these practices more closely, we found that highly evolved organizations (see **The evolutionary scale**) were much more likely to **always** be using these practices throughout the evolutionary journey than the less-evolved organizations. What we take from our findings is that the foundational practices listed above are integral to DevOps, and critical for DevOps success.

The foundational practices and CAMS

The importance of the foundational elements of DevOps shouldn't surprise anyone who takes more than a passing interest in DevOps. Other well-regarded constructs are built around these same foundations. The CAMS model, one of the earliest descriptions of DevOps, encompasses these foundational known-good patterns, from the importance of measurement and sharing to the need for automation. Other tropes and methodologies common in the DevOps discourse, concepts such as shifting left, empowered teams, test-driven development and more also reinforce these foundational patterns and practices.

It's all about sharing

On studying the foundational practices revealed by our research, we realized that they are all dependent on sharing, and that they all promote sharing.

- **Monitoring and alerting are configurable by the team operating the service.** Monitoring and alerting are key to sharing information about how systems and applications are running, and getting everyone to a common understanding that is vital for making improvements, whether within a single team and function or across multiple teams and functions.

- **Deployment patterns and testing patterns for building applications or services are reused.** Sharing successful patterns across different applications or services often means sharing across different teams, establishing agreed-upon ways of working that provide a foundation for further improvements.

- **Teams contribute improvements to tooling provided by other teams.** This form of sharing promotes more discussion between teams around priorities and plans for further improvements in tooling, process and measurement.

- **Configurations are managed by a configuration management tool.** A configuration management tool enables development, security and other teams outside Ops to contribute changes to system and application configurations. This makes operability and security a shared responsibility across the business.

Our discovery that all the fundamental practices enable or rely on sharing tells us that the key to scaling DevOps success is adoption of practices that promote sharing.

It makes sense: When people see something that's going well, they want to replicate that success, and of course people want to share their successes. Let's say your team has successfully deployed an application 10 times, and let's also say this type of deployment has normally given your team (and others) a lot of trouble. Chances are, someone will notice and want to know how you're doing it. That's how DevOps practices begin to expand across multiple teams.

The 5 foundational practices, one by one

This section describes each foundational practice in some detail, and how the practice contributes to the evolution of DevOps.

Monitoring and alerting are configurable by the team operating the service

Core to the DevOps movement is the two-sided coin of empowerment and accountability, which Amazon CTO Werner Vogels summarized in his famous statement: "You build it, you run it."[3] So our research looked at how many teams that run applications and services in production — whether comprised of devs, operators, software release engineers or others — are able to define their own monitoring and alerting criteria.

Empowered teams that run applications and services in production can define what a good service is; how to determine whether it's operating properly; and how they'll find out when it's not. This empowered monitoring approach can take many forms. For example:

- "Drop a monitoring config in a location and we'll pick it up."
- "Log into this web interface to configure your monitoring."
- "Add some monitorable outputs to your infrastructure code."
- "Here's an API for you to configure monitoring as code."

We found that once organizations start to see traction with DevOps, 47 percent of the highly evolved (High) cohort are able to define their own monitoring and alerting criteria for apps and services in production, compared to just 2 percent of the least-evolved (Low) cohort. The High cohort is *24 times* more likely to have adopted this practice! Conversely, the Low cohort was 23 times more likely to never use this practice.

24x

more likely to make monitoring and alerting configurable by teams

---

3 Gray, J., Vogels, W., A Conversation with Werner Vogels, ACM Queue, queue.acm.org/detail.cfm?id=1142065. June 2006, retrieved Aug 2018.

Frequency by evolutionary scale: configurable monitoring and alerting

In our survey, we asked, "How frequently were these practices used after you started to see some traction with DevOps?" Below is the breakdown of answers for the practice, "Monitoring and alerting are configurable by the team operating the service."

	Low	Medium	High
Always	2%	17%	47%
Most of the time	8%	37%	47%
Sometimes	27%	32%	5%
Rarely	38%	11%	<1%
Never	23%	3%	-

Empowering teams to define, manage, and share their own measurement and alerting supports multiple elements of a DevOps transformation, including:

- Sharing metrics as a way to promote continuous improvement
- Creating and promoting a culture of continuous learning
- Cross-team collaboration and empowered teams
- Development of systems thinking in individuals and teams

These factors are core to a strong DevOps culture, as we discussed earlier, so it's not surprising that the highly evolved organizations we surveyed adopt this practice early.

Measurement is a core piece of DevOps

Empowered monitoring isn't for just Ops or for some newly created DevOps team: It's for all teams that work with technology. The embrace of empowered monitoring for all teams underlines one of the most important points of DevOps: that you don't need to create a new team with new superpowers, but instead should empower all existing teams so they can work together in new ways.

Self-service monitoring and alerting can just as easily and usefully be adopted by:

- developers running their own code.
- a team of developers working with their operations counterparts to deliver operable applications.
- a DevOps team working as a single cohesive group to define their own monitoring practice.
- a complex team of teams where ops specialists monitor applications delivered by devs as part of a broader system.

Regardless of the specific circumstances, self-service monitoring and alerting is a countermeasure to the long-standing anti-pattern of dev and ops working in silos. Simply opening access to these key metrics enables a sharing culture, populates feedback loops, enables continuous feedback, and promotes a culture of continuous learning across teams.

Instilling ownership and accountability by empowering service delivery teams to collect, share, and customize monitoring data is a fundamental part of the cultural change of DevOps, and enables even more fundamental shifts further along in the process.

Reuse deployment patterns for building applications or services

By the time our survey respondents had gained some traction with their DevOps initiatives, 46 percent of highly evolved organizations reported always reusing deployment patterns for building applications or services, versus 2 percent of the least-evolved organizations. So the highly-evolved teams are *23 times* more likely to always employ this practice.

Frequency by evolutionary scale: reuse of deployment patterns

We asked, "How frequently were these practices used after you started to see some traction with DevOps?" Here is the breakdown of responses for the practice, "We reuse deployment patterns for building applications or services."

	Low	Medium	High
Always	2%	14%	46%
Most of the time	7%	44%	47%
Sometimes	34%	33%	6%
Rarely	40%	8%	1%
Never	18%	1%	-

We asked about reuse of of deployment patterns because of the special nature of application deployment in most, if not all, organizations. Residing at the boundary between development and production, application deployment is where Dev and Ops most often meet — and most painfully collide. So improving application deployment is right at the core of DevOps, as it mediates the "wall of confusion"[4] at the intersection of Dev and Ops.

The use of repeatable patterns — whether created in-house or adopted from an external source — does more than alleviate the immediate pain and confusion of deployment.
It also makes it possible to share and spread the knowledge of how to deploy more widely in the organization, enabling more teams and individuals to work together on what needs to be a core competency for any business.

highly evolved orgs are

23x more likely to reuse deployment patterns

4   dev2ops.org/2010/02/what-is-devops

Reuse testing patterns for building applications or services

Just like the ability to share and reuse deployment patterns, organizations that are making progress in their DevOps evolution use repeatable testing patterns.

As organizations are starting to achieve traction with DevOps, 44 percent of highly evolved organizations reported that they always use repeatable testing patterns compared to fewer than 1 percent of the least-evolved organizations. That makes highly evolved organizations *44 times* more likely to use repeatable testing patterns.

highly evolved orgs are

44x

more likely to reuse testing patterns

Frequency by evolutionary scale: reuse of testing patterns

We asked, "How frequently were these practices used after you started to see some traction with DevOps?" Here's the breakdown of answers for "We reuse testing patterns for building applications or services."

	Low	Medium	High
Always	<1%	10%	44%
Most of the time	6%	38%	48%
Sometimes	32%	39%	7%
Rarely	40%	11%	<1%
Never	21%	2%	-

Automated testing and reuse of testing patterns can be one of the harder challenges to solve depending on your organization's structure and complexity. Though we do see this practice adopted by highly evolved organizations in the early stages of a DevOps evolution, it may not be the first thing you tackle. Here are some considerations as you prioritize this practice:

- If quality teams are quite disconnected from dev and ops teams, you may want to wait to integrate them into DevOps initiatives later on. Focus on establishing good testing patterns within your own team first. For ops teams, that could mean having a process for testing infrastructure changes before deploying to production. For dev teams, that could mean implementing test-driven development (TDD) or other methodology as part of your agile workflow.

- Activities closest to production, such as provisioning, monitoring, alerting, etc., are often higher priority for teams because that's where more issues become visible. Solve your deployment pains first to gain back time you can then use for improving your testing practices.

- Testing patterns may be less reusable than deployment patterns because testing deals with the specifics of an application or service, and also covers many different processes — smoke tests, unit tests, functional tests, compliance tests, complexity tests — in both static/white box or dynamic/black box environments.

- Testing in production is harder and often more complex than testing in pre-production, as its goals are different from those of pre-production test. Quality teams test in pre-production for compliance, stability, security, customer satisfaction and other core goals. With continuous delivery, teams can experiment in production to test new ideas (e.g., via blue/green or canary releases). This is valuable, but it's different yet again from testing in production, which focuses on quality, functionality, resilience, stability, and more.

We conclude that adoption of reusable test processes is a fundamental practice, but tends to get pushed out later in the evolutionary journey, after deployment patterns are well established. If you have to prioritize, we recommend waiting to tackle this one and focusing on the other practices first. However, once you do adopt this practice, it's important to ensure that testing patterns are shareable. For example, you can encode reusable tests into automated testing tools, and share access to those tools along with the resulting reports or dashboards among all stakeholders.

Teams contribute improvements to tooling provided by other teams

The ability to contribute improvements to tooling provided by other teams stands out as a key foundational capability.

As organizations expanded their DevOps practices, 44 percent of the highly evolved cohort reported that teams could always contribute improvements to other teams' tooling compared to fewer than 1 percent of the least-evolved cohort. In other words, the highly evolved cohort is 44 times more likely to employ this practice.

highly evolved orgs are

44x

more likely to contribute to other teams' tooling

Frequency by evolutionary scale: Contributing to other teams' tooling

We asked, "How frequently were the following used while you were expanding DevOps practices?" Here's the breakdown of answers for "Teams contribute improvements to tooling provided by other teams."

	Low	Medium	High
Always	<1%	11%	44%
Most of the time	4%	31%	45%
Sometimes	26%	40%	11%
Rarely	46%	15%	-
Never	23%	3%	<1%

Improvements to tooling are typically manual and ad hoc, and siloed within a single team until some change drives the need to open up to other teams. This change might be division-wide culture or organizational changes; new cross-boundary data sources such as semantic logging; or new collaboration across teams at functional boundaries such as provisioning or release automation.

Because the practice of cross-team contributions to tooling is dependent on teams putting their own houses in order first, we believe adoption of this practice can be left to a later stage with equal chances of success.

Configurations are managed by a configuration management tool

The practice of managing configurations with a configuration management tool rapidly takes root once organizations start to see traction with their DevOps evolution. Fifty-three percent of the highly evolved cohort reported employing configuration management always, compared to 2 percent of the least-evolved cohort. The highly evolved cohort is almost *27 times* more likely to *always* use a configuration management tool.

**Frequency by evolutionary scale:
Use of configuration managment tools**

We asked, "How frequently were these practices used after you started to see some traction with DevOps?" Here's a breakdown of answers to "Configurations are managed by a configuration management tool."

	Low	Medium	High
Always	2%	19%	53%
Most of the time	11%	37%	40%
Sometimes	28%	31%	8%
Rarely	36%	10%	-
Never	24%	3%	-

For long-time DevOps devotees, it is not surprising to see this practice emerge as a baseline for success. Automated configuration management was a prime mover of DevOps for many years, especially in the earliest days of thinking about infrastructure as code, and the DevOps movement largely coalesced around the earliest innovators in automated configuration management.

As DevOps evolves and expands, and developers liberated by automated provisioning move increasingly toward continuous delivery, Ops is under even greater pressure to maintain uptime, performance and availability in production. Auditability concerns also emerge as an organization's processes mature. So automated configuration and provisioning for production become just as important as provisioning for development and test. Achieving repeatability via configuration management assures stable, reliable and auditable production environments, and also enables later-stage capabilities — for example, self-service that emerge as new goals for the DevOps initiative.

27x

highly evolved orgs are 27x more likely to use configuration management tools

Implementing the foundational practices in your organization

With so much evidence that the five foundational practices lead to success, we know our readers will want guidance on how to implement them. Fortunately, our research does provide some indication of which practices to start with.

Three practices were always in active use by the majority of highly evolved organizations by the time they were seeing traction in their DevOps initiatives:

- Reusing deployment patterns.
- Using a configuration management tool.
- Allowing a team to configure monitoring and alerting for the service it operates.

While these three practices are foundational capabilities, and form a baseline for progression to higher levels of DevOps evolution, the order in which they are adopted is not important. Do start here, but don't worry about which comes first. It's likely you'll recognize one particular practice as something your own organization needs to prioritize.

The remaining two foundational practices are ones we recommend prioritizing after the other three practices are well established:

- Reuse testing patterns for building applications or services.
- Empower teams to contribute improvements to other teams' tooling.

Stage 1: Normalize the technology stack

Most organizations using any amount of technology are dealing with a lot of complexity, slowing down their efforts to advance the business. So it's not surprising that the earliest efforts in a DevOps transformation (or any kind of business transformation) would center around reducing complexity.

The two practices that define Stage 1 work to reduce complexity:

- Application development teams use version control.
- Teams deploy on a standard set of operating systems.

Why DevOps evolution starts with simplification

Starting a DevOps evolution by reducing complexity may surprise people who think of automation as the first step in DevOps, especially since automation is a core pillar of the movement. In fact, the automation practices typically associated with DevOps don't show up significantly until Stage 4. That's because a lot of preparation has to take place before automation can be properly designed and implemented.

Anecdotally speaking, we have seen organizations start with Stage 4 automation, without having been through normalization, standardization and expansion (Stages 1-3). These organizations do not achieve success, and we believe it's because they lack a foundation of collaboration and sharing across team boundaries. That sharing is critical to defining the problems an organization faces and coming up with solutions that work for all teams.

Our research shows that DevOps evolution begins long before Stage 4, so skipping the early stages means missing out on the learning that takes place during these periods. The early stages are also when teams establishing and succeeding at DevOps practices earn the trust of the business, which can mean more resources and permission to progress faster.

Application development teams use version control

The use of version control by application development teams represents a fundamental shift in how teams produce software. It's the first step to implementing continuous integration on the path to continuous delivery, which is frequently the goal of a DevOps initiative.
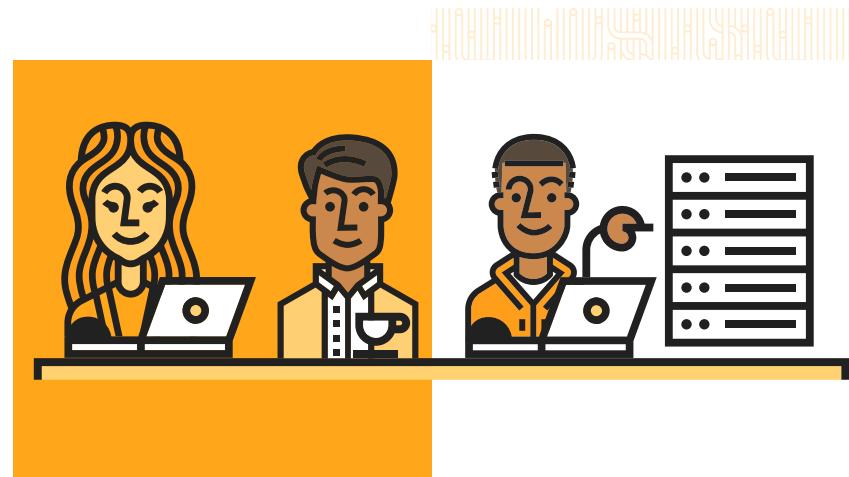
When app teams are using version control, they're usually producing deployable code much more frequently than before. So the pressure on ops teams to deploy quickly while keeping systems secure and stable increases. This is one of the main drivers for DevOps, and it's often how people know they need to move forward in their DevOps journey.

> In previous State of DevOps Reports, we've found that the use of version control for all production artifacts was highly correlated with key IT performance metrics: deployment frequency, lead time for changes, and mean time to recover. In last year's study, our analysis showed that the use of continuous delivery practices — deployment automation, continuous integration and testing, and version control for all production artifacts — predicted lower levels of deployment pain, higher IT performance, and lower change failure rates.

Teams deploy on a standard set of operating systems

In the early stages of a DevOps evolution, we see a concerted effort to normalize the stack and get rid of outliers or snowflakes that need to be maintained, tested and managed as one-offs. The more variance you have, the more complex, difficult and time-consuming it becomes to manage your IT.

So it's not surprising that the second defining practice for Stage 1 is deploying on a standard set of operating systems. In large enterprises, it's not uncommon to have multiple applications all running on multiple OSes. For example, one application may run on Windows 2012, another on Windows 2012 R2, and yet another on Windows 2016. Eliminating even one of those variables significantly reduces complexity, plus it becomes much easier to build a shared pool of knowledge around a common tech stack.

Primary contributors to success in Stage 1

We found that the following practices had the most significant impact for success in Stage 1.

- Build on a standard set of technology.
- Put application configurations in version control.
- Test infrastructure changes before deploying to production.
- Make source code available to other teams.

Build on a standard set of technology

Building on a standard set of technology is a contributor to success in Stage 1, it's a defining practice for Stage 2, and it shows up again in Stage 3 as a contributor to success. The prevalence of this practice in the early stages of evolution tells us that this is an important ongoing effort, and that the practice itself is constantly evolving. Starting with specific technologies within a single team's sphere of influence, standardization then spreads to technologies that require buy-in from multiple teams.

From a business perspective, there are many benefits to standardization: reduced licensing costs (it's cheaper to buy licenses in bulk); ability to hire for a specific skill set; and shared knowledge across teams, which ultimately leads to greater agility and faster delivery of higher quality software.

While standardizing the tech stack provides clear business benefits, rigidly adhering to standards can put a damper on learning and innovation. The key is to regularly revisit standards and build in exceptions for innovation and experimentation.

We recommend standardizing with an eye to what is optimal for all applications, not just a few applications. Use proven technologies and reliable processes for what goes into production, and provide clear processes and guidelines for adding any new technology to enable product incubations, research and experimentation.

Put application configurations in version control

Putting your application configurations in version control is part of the process of normalizing the environment. To keep things simple, many teams initially combine their application configuration data with their app configuration code. At some point in the lifecycle, this becomes difficult to deploy and maintain, so app configuration data gets pulled out into configuration files.

As development gets more distributed, teams need version control, and put both application code and app configuration files into their VC (version control) system. Eventually someone realizes that configuration data and sensitive information need to be managed more rigorously.

It's standard practice to separate your data from your code because some configuration data varies per deployment environment (dev, test, stage, production) while application code remains the same. It's also how you ensure that your sensitive information is safeguarded from exposure.

Separating data from code is low-hanging fruit, and makes sense in these early stages. It also builds the foundation for automated deployment. With app configurations in version control, you can track who makes what changes, and roll back changes as needed. If you're just starting out, there are a host of key value store tools available that solve this problem — for example, etcd, ZooKeeper, and Consul. If you're deploying to immutable infrastructure, you're forced to solve these problems up front.

Test infrastructure changes before deploying to production

Also contributing to Stage 1 success is testing infrastructure changes before deploying to production. This becomes a critical practice at Stage 3. Teams at Stage 1 are normalizing testing procedures for infrastructure changes, but it's unlikely that these are fully automated procedures going through an established pipeline. That normally happens in later stages.

Testing infrastructure changes in Stage 1 does a few important things. It builds trust in the system so teams can gain autonomy to work without manual approvals, and also provides the foundation for creating reusable deployment patterns, which you can't do unless you have a standard way of testing changes.

Make source code available to other teams

Another practice with significant impact on Stage 1 is making source code available to other teams. Opening up version control is an early driver because it encourages collaboration via contributions from other teams. It's likely that there are pockets of success around the organization, and reusing what those teams have already built accelerates development of capabilities, enabling success to scale.

Stage 2: Standardize and reduce variability

In Stage 1, we see organizations normalizing their technologies and processes. By the time they reach Stage 2, organizations have already begun the process of standardizing on a set of technologies; separated application configurations from data and placed them in version control; and adopted a consistent process for infrastructure testing and a pattern of sharing source code.

In Stage 2, organizations are working to further standardize and reduce variability, a theme that is prevalent at every stage in the DevOps evolution. Every organization has variance, which can stem from a number of different causes, including:

- Adoption of new technologies to replace many functions of older technologies; yet the older technologies never actually get removed.
- Homegrown products that don't follow any common industry standards and lack common interfaces.
- A proliferation of tools that overlap and haven't been rationalized.
- Mergers and acquisitions.

At Stage 2, reuse of technologies and patterns becomes important. This drives Dev and Ops to collaborate and make architectural decisions that affect the deployability and testability of applications. Because of driving toward these commons standards, teams start to invent ways to increase velocity in standards adoption, and further reduce variance. This drives innovation at the team level to optimize processes and workflows around the blessed technology stacks.

Teams collaborating together are likely to see success, particularly through their primary interface points for applications and processes — for example, Ops providing good compute, storage and network deployments, identity management and more. Service delivery will improve as collaboration improves at this stage, reflecting **Conway's Law.**

A primary anti-pattern to watch for at this stage is each team normalizing on its own standards. This will lead to a greater degree of global variance, and is exactly the wrong direction.

The defining practices for Stage 2 are:

- Build on a standard set of technology.
- Deploy on a single standard operating system.

One of the barriers to adopting DevOps in the enterprise is the sheer complexity of the organization. As enterprises grow over time, they inevitably add new applications and services, adopt new technology stacks, and still have to deal with legacy applications and systems. Technical debt piles up.

Because of increased complexity, fragile systems, and variable processes, teams end up spending most of their time reacting to problems rather than driving innovations. The answer in this stage isn't to adopt a new tech stack and re-architect everything. This isn't the time to add a new database. Instead you need to standardize on proven technologies, optimizing for the 80 percent cases and your global use cases. This can be done only in collaboration with other teams.

The main benefit in this stage is reducing variables and therefore complexity, buying time for further investments in collaboration, automation, sharing, and metrics in subsequent stages.

Applying the scientific method to reducing variables in software

The number of variables in any process or system is directly proportional to its complexity. With fewer variables in play, it is easier to execute a process. And with fewer variables, you can also isolate them, modify them and measure the impact of each change. Next you reduce the variables to optimize flow. Then you make changes in those variables to further optimize output.

Build on a standard set of technology

Let's start from the perspective of a delivery team, whether that's a dev team responsible for delivering its own code, a team that includes developers and operations people, or a team that includes operators and software release engineers. The team needs to deploy services. Is each service built on the same architecture? Do they all use the same message queue? Do they all use the same components and patterns? When the answer to any of these questions is "No," complexity enters the system and the maintenance burden increases.

Once a delivery team has standardized its patterns and components, the team no longer has to continually re-learn how different technologies operate, scale, fail, recover, and upgrade. The time recovered can be used to increase velocity or to develop things that truly differentiate the application or service — both of which can help provide a competitive advantage for the entire organization.

Some teams do this without much thought. Others, particularly those who inherit code from all over an organization, have to take a methodical approach towards eliminating variables and achieving standardization. Start by choosing foundational elements to normalize on — for example, you could select a single relational database management system and a single key value store.

If you're starting with several combinations, elimination of even one helps cut time spent on maintenance, not only for the delivery team but also for other service-providing teams.

You can also reduce variables by normalizing your testing workflows, build, and shipping patterns. The primary objective here is that improvements and optimizations to the build/test process apply to more than a single application, because several apps use common components.

When building new applications or services, it's important to look at the tools you have. Rather than use a new database, could you reuse what you have already? Keep in mind that there are always costs for retiring technology, but it's usually worth it, as you recapture those costs in savings over the long term.

The necessity of normalization and standards doesn't mean teams should not innovate. Ideally, teams driving better understanding of their problem domain are innovating, and with technology where warranted. There should be a lower barrier to trying something, but the barrier should rise significantly when it comes to introducing a new piece of technology into a production lifecycle.

**The key benefits of standardizing a team's patterns and technologies are:**
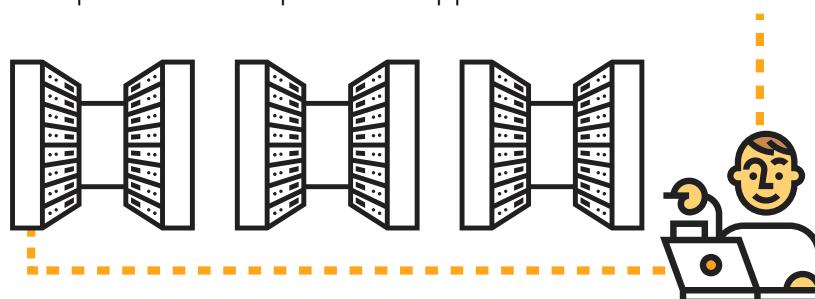
- Faster delivery velocity.
- More flexibility for development staff to work on different applications, services or components.
- Reduced surface area for security vulnerabilities.
- Fewer moving parts to maintain, upgrade and learn.

Deploy on a single standard operating system

If each software delivery team has normalized its own stack, that's a local optimization for this team — but what about the operations team? If Operations supports several delivery teams, they may have several normalized stacks to support. This is where we see shifts into more global optimizations, often led by operations. Allowing operations to select standards for operating system and versions, monitoring interfaces and deployment systems can eliminate several debates for software delivery teams, giving them more time to focus on their core mission.

Organizations can move faster when a single operating system, or a small set of operating systems, is the standard. You save time on patching, tuning, upgrading and troubleshooting when there's just one OS or at least a very small number in use.

Operating system standardization seems straightforward at first, but once you dig in, it isn't. If you support software applications that have long life cycles, for example, you may have something that works only on an operating system that was launched more than half a decade ago. There may be incompatibilities between specific patches and a particular application.

A common path forward is to first eliminate any stray operating systems in your fleet. If you have five OSes, reduce to two. Next, normalize your compute resources that are running the same operating system. Make sure they have the same sets of patches, the same update level, the same BIOS/firmware (if applicable), and so on. If you don't do this, all those variables will make it much more complex to troubleshoot and perform maintenance. Once you do normalize, you'll have more time and attention available for further improvements, and the right base to build them on.

Our advice for standardizing your operating systems:

- Even if you can't get down to a single OS, remember less is more.
- On a single operating system, work on reducing variability as much as possible.

Beyond operating system standardization is the rest of the technology stack. The owners and choosers of the technologies in play here can vary. Standardizing across many teams on technology choices like database systems, message queues, logging aggregation utilities, monitoring/metrics instrumentation and collection, and key value stores allows for any lessons learned in supporting and maintaining those tools to be reapplied to other applications and teams.

Use what you have

For a great perspective on using what you have (rather than immediately branching into new things just because you can), read the write-up of **Dan McKinley**'s talk at **boringtechnology.club**. Dan provides a real-world look at tool choice from his time at Etsy, Stripe and Mailchimp.

Standardizing in these spaces is a challenge. Often a development team will gravitate towards new technology, or something that specializes in a problem they have, without taking into account the burden on operations and the cost of caring for yet another piece of technology. For example, if you need to store json blobs, a tool like MongoDB might immediately come to mind. However, if you already have PostgreSQL as a backend data store, you could put json blobs in there, and so avoid learning about MongoDB, scaling it, understanding its failure patterns, backup requirements, monitoring hooks and more. And Ops will not be burdened with yet another system to maintain.

Contributors to success in Stage 2

There are three practices that have a significant impact on Stage 2 success:

- Deployment patterns for building applications and services are reused.
- Rearchitect applications based on business needs.
- Put system configurations in version control.

Reuse deployment patterns for building applications or services

While this practice contributes to Stage 2 success, it is also a defining practice for Stage 3. See the chapter on *Stage 3* for a detailed analysis of this practice.

Stage 2 and Stage 3 don't necessarily have to be performed linearly; however, our data shows that organizations need to complete both stages before moving on to Stage 4 in nearly all cases.

Rearchitect applications based on business needs

Teams at Stage 2 have made the deliberate choice to build on a standard technology stack, so now they have to make changes to existing applications. This is an opportunity to rearchitect not only for new technology requirements, but also to check in with the business and make any necessary adjustments there. For example, there may be new auditing guidelines that require more logging than before.

For some application delivery teams, rearchitecting could mean replacing a home-grown message queue with a well-known open source or commercial component. For others it could mean updating architecture to fit with a deployment model where app configurations have been separated from application code.

The primary goal of architecture changes is to support standardization and align with its goals — greater velocity and easier maintainability. The sharing of common components and interfaces has an additional benefit: Staff may have more time to work on applications, instead of spending so much time on the mechanics of delivery.

Put system configurations in version control

As we've seen in past State of Devops Reports, the use of version control predicts IT performance: The teams that use it have higher IT performance than those that don't. In Stage 2 of their DevOps evolution, teams are ensuring that system-level configurations are in version control.

Storing system configurations in version control is a vast improvement over scripts living on people's workstations, providing a number of advantages:

- You can see changes over time, see how they evolved, and know who made them.
- Anyone with access to the version control system can audit changes.
- You get automatic backups of key configuration files.

Keeping system configurations in version control is also one of the first steps to adopting software development practices for infrastructure. This in turn is key to automated infrastructure delivery, and a building block toward infrastructure as code.

During Stage 2, organizations storing configurations in version control are deploying them via scripts, manually, or an automation framework, depending on how far they have come along their automation path.

Stage 3: Expand DevOps practices

Stages 1 and 2 reduce the overall complexity of the tech stack so teams can achieve more repeatable outcomes with limited variance. Stage 3 is about expansion of DevOps practices to the wider group of teams in IT and service delivery.

In Stage 3, DevOps practices spread beyond the Dev and Ops teams, where they first take root. As collaboration increases and the organization focuses on improvements around service management, deployment, reducing wait times and minimizing approvals, these efforts touch areas beyond the technology departments. Sharing improved tools, applications and services — as well as knowledge — with other functional areas of the business now becomes key to expanding on prior DevOps success, and scaling DevOps across the organization.

Our research shows that Stage 3 is where DevOps initiatives morph from small pockets of success in a few teams to a wave that spreads across and eventually transforms an entire organization.

We've observed in our findings that Stage 2 (reducing variation in the tech stack) and Stage 3 can take place in order, in reverse order, or at the same time. But both need to happen before progressing to Stage 4 (automating infrastructure delivery). We think it makes more sense to focus on reducing variability in the earlier stages so that there are fewer one-offs to manage, saving your team time and distraction. But if that's not possible because some of those things are outside your control, then work first on the things you can control. What's most important is that the IT service management team and any other teams relying on services work together during this stage.

The defining practices at Stage 3 are:

- Individuals can do work without manual approval from outside the team.
- Deployment patterns for building apps and services are reused.

> Stage 2 and Stage 3 don't necessarily have to be performed linearly; however, our data shows that organizations need to complete both stages before moving on to Stage 4 in nearly all cases.

Individuals can do work without manual approval outside the team

In past State of DevOps reports, we've found that having an external change approval board had a negligible impact on stability, but a detrimental effect on agility. Despite this evidence, we see all too often that the authority to make decisions is removed from the people who have the relevant information and are doing the actual work.

Empowering teams and individuals certainly supports the spirit of a DevOps evolution, in addition to getting work done more quickly. When someone can get work done with minimal handoffs, approvals and wait time, they're happier and more productive. So Stage 3 is where bureaucracy should shrink, and processes should be redefined and updated to reflect the mutual trust being earned via DevOps investments.

The data indicate that organizations in Stage 3 allow individuals to work with relatively few approvals required from outside the team. In some organizations, simple changes require review and approval from a change advisory board, which includes a mandatory waiting period. Successful organizations are reducing this bureaucratic red tape by partnering with IT service management (ITSM) teams to revisit processes, and building trust to speed up approvals — or ideally, to eliminate them.

An example: An operations person could work with ITSM teams and get approval to make certain types of standard and safe changes. This probably means establishing a track record of incident-free changes, creating a standard method for deploying the standard changes, and documenting it. This is a small, simple step that can help to build trust between operations and the ITSM teams.

Another option is to give the operations team the power to approve specific types of changes depending on the severity. Perhaps a team lead or supervisor approves each change, rather than routing changes to an ITSM team, another layer of management, or multiple people. This approach can certainly save time while helping to build trust in the operations team's ability to keep systems safe, efficient and aligned with business goals.

> **Sometimes individuals can do lots of work without approval from outside their management chain simply because nobody else knows about it and they can sneak the work in. While that accomplishes the latter of the correlated data point, "Individuals can do work without manual approval outside of the team," it isn't the best path forward.**

The primary driver of bureaucratic process is normally communication and broadcasting of potential impacts and issues. If those exigencies are kept in mind while shortening and simplifying the change control process, people who want to improve technical functionality can begin reversing their view of change control as an obstacle to get around.

Deployment patterns for building applications and services are reused

Universally, organizations in Stage 2 reuse deployment patterns. Deployment patterns can be quite simple, or as sophisticated as using specialized tooling that integrates with ticketing and monitoring systems — and everything in between.

Reuse of deployment patterns at this stage can mean simply that you have two software projects and that you deploy both of them the same way, whether to dev, test, staging or production. Someone who deploys App A should be able to deploy App B without lots of documentation and hand-holding.

Some organizations begin by standardizing on entry points for deployment — for example, to deploy any application, you type `./deploy <environment>`. The rest of the deployment process may vary from one application to another, but at least you have the same invocation to launch any deployment. That's a good start.

The next step is to use the same tools for your deployments. For example, all deployments may run through continuous integration (CI), with the CI system performing a set of jobs that result in a full deployment after passing. Your team may use a specialized tool designed for an application suite, such as enterprise resource planning (ERP) tools.

You also need to consider the order and style of your deployments. Perhaps database migrations always happen first. Systems may get flushed from load balancers before the deployment or as part of it. Your organization might do blue/green deployments or require an outage to deploy.

When running several types of applications and systems, you may see some deployment patterns emerge as universal for all your applications or systems, and others that apply to certain families of applications — for example, n-tier web apps or cloud-native services. Other patterns may be specific to an application.

Some organizations are strict about separation of duties, so a team that deploys an application cannot be the team that wrote and developed the application. This is an even stronger case for unified deployment process flow, tools and patterns. Failures can be investigated and managed uniformly across different services, so the teams responsible for deployment are less likely to have to go back to service authors when a deployment fails.

In organizations where deployment patterns are truly mastered, multiple applications use the same pipelines and jobs for deployment; only the application name and possibly a few other parameters are fed to the job as configuration. With deployments standardized and reused to this degree, any optimization to the deployment job or pipeline is immediately consumed by all applications, so the benefits multiply quickly.

Even if your organization has many cross-functional self-sufficient teams, there is still a lot of value to reusing deployment patterns. When each team invents its own deployment patterns, that limits agility, and the team doesn't have time to spend on truly differentiating work. This also makes it harder for developers and infrastructure engineers to move between teams, which further limits agility (and, by the way, makes it harder for your people to grow and develop at your organization, threatening retention). It's possible that with cross-functional teams, deployment patterns will be limited — for example, to entry points — and quickly move into application- or service-specific details.
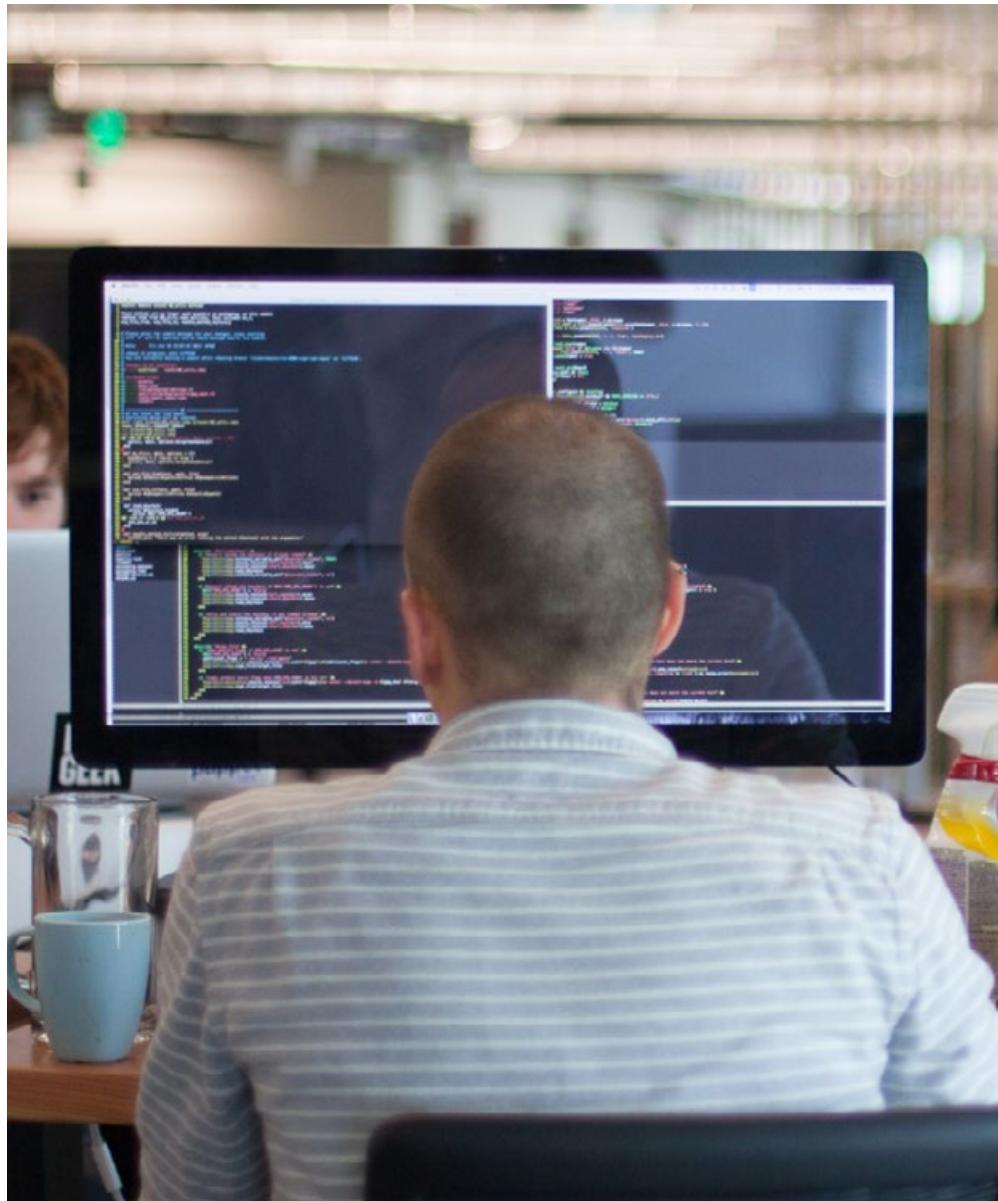
Infrastructure changes are
tested before deploying to production

A practice that's associated with Stage 3 is testing infrastructure changes before deploying to production. For success in expanding your DevOps initiative, you need to demonstrate the ability to make predictable and reliable changes.

Many people think infrastructure testing should be fully automated, relying on continuous integration and an infrastructure-as-code approach. While automation is more reliable and generally faster, keep in mind that it's the *validating* that matters, and that you can test infrastructure changes manually.

Why are we pointing this out? Because infrastructure changes can vary widely, and while some lend themselves to automation with a reasonable amount of effort, other changes are just too infrequent or expensive to validate in an automated fashion. So don't get too locked into the method — just make sure that you validate infrastructure changes prior to a production deployment.

For example, when replacing core network switches in a data center, the engineers should be sure they understand the new switch, have tested its capabilities, have a deployment plan, and know they must validate functionality. Most of this can be done in a lab or development environment so most scenarios are accounted for before production. Any change in production should model the same paths taken in the lab environment.

Contributors to success in Stage 3

Six practices have a significant impact on Stage 3:

- Individuals can make changes without significant wait times.
- Service changes can be made during business hours.
- Post-incident reviews occur and results are shared.
- Teams build on a standard set of technologies.
- Teams use continuous integration.
- Infrastructure teams use version control.

We'll discuss each of these contributing practices next.

Individuals can make changes without significant wait times

At this stage of the DevOps evolution, our research shows that organizations need to work on reducing wait times for approvals. These make it harder to be agile, and go against the DevOps principle of empowering people and teams.

But wait periods aren't put in place just to be awkward, though it can sure feel like that's the case. It's helpful to look at the reasons for each wait and ask what would have to change in order to eliminate it. For example, an approval requirement could predate an improved process. So if a team could show 10 successful sequential deployments — with no incidents — it's possible that this type of deployment could now be done without any waiting period at all.

Another angle is to look at what work can be done during business hours versus the work that requires an outage window, off-hours maintenance, or mandatory wait times, and to work on reducing the number of operations that must be performed during off hours. Again, it's a matter of looking at what would have to happen to eliminate that requirement.

Working to accomplish changes and deployments without wait times isn't about getting carte blanche permission to bypass organizational process. It's about revisiting why process exists so you can simplify it, normalize it and optimize it. When processes are simpler and consistent, they're also easier to automate, which comes in handy as organizations progress toward self-service.

Service changes can be made during business hours

As your team continues to expand DevOps practices — and succeed — the constraints imposed by red tape and bureaucratic process should be reducing. Once wait times have been analyzed for reduction, you can move onto performing service maintenance tasks and changes during business hours.

Some organizations do maintenance *only* during business hours, making use of canary deployments, blue/green deployments or active/passive sides of an application. These architecture and deployment patterns optimize for rolling change through the system often, and allow for a relatively easy backout plan if a change goes awry.

Getting to the point where you can make changes during business hours takes some preparation. First, you need to define what business hours means for your organization. If you're always on (like a web service), customers both internal and external may expect your service to be available all the time. Second, you need to demonstrate success in making changes reliably so the business partners and stakeholders of your service trust your abilities. You need to be believed when you say changes will have no impact on performance or customers.

Learn about post-incident reviews from Jason Hand.

Post-incident reviews occur and results are shared

Post-incident reviews are a blameless look back at what happened during an incident, how it happened, and what improvements could be made to shorten the duration of the incident, improve the understanding of the systems behind the incident, and prevent it from happening again. Post-incident reviews come out of both the sharing pillar of CAMS and DevOps principles. They are designed to replace traditional approaches to after-action reviews such as root cause analysis, and drive toward understanding and continuous improvement rather than looking for a single cause for any incident.

A distinguishing feature of post-incident reviews is the inclusion (where appropriate) of people who can provide a business perspective on the incident. Participants can include business analysts, management — even customers or consumers of the application or service you're reviewing — in addition to IT teams, delivery teams, operations teams, and ITSM teams. This cross-functional collaboration helps to foster trust, and builds the sense of shared responsibility for success and improvements.

Improvements from a well-run post-incident review can include revisiting and simplifying processes; updating communication patterns; and working from a position of empathy with other stakeholders of the application or service.

Once a post-incident review is done, share the results. People who were not directly involved may be able to learn something. They may spot a flaw in an adjacent process, or simply be curious what happened when they couldn't reach your site for hours. Some organizations share results with their customers publicly, while others make them available to internal customers and stakeholders. The more you share, the more collaboration and trust you'll foster.

Teams build on a standard set of technologies

Building on standardized technology contributes to success in Stage 1, is a defining practice of Stage 2, and shows up again as a contributor in Stage 3. The prevalence of this practice in all these stages tells us that standardizing on technologies is an ongoing effort, not a single moment in time.

As mentioned in *Stage 0: Build the foundation*, the kind of tooling improvements teams make evolve over time. Normally this starts with separate teams making improvements for their own purposes, so these efforts are siloed, ad hoc and often manual. At some point, changes to the organization or the technology drive a need to collaborate with other teams, often at functional boundaries such as provisioning or release automation. That's when the real cross-collaboration on tooling improvements begins.

Teams use continuous integration

In years past, we've seen using CI as a leading indicator of whether or not a team will be high performing. CI is a must-do in the DevOps space, right after version control becomes ubiquitous.

CI systems and pipeline flow can vary immensely based on the types of software in play, job construction, tooling, and who consumes the workflow. The important things to optimize for are feedback cycle time and correctness. When feedback cycle times are short, more iterations can occur, and so quality improves.

Correctness also matters, so CI systems require maintenance, adjustments and improvement over time. For example, if you add a new operating system or browser to your support matrix, all relevant jobs should be able to pick it up. Or it may make sense to run only fast tests during working hours, and wait to run slower tests at night or during a weekly window when feedback cycle time is not as critical.

Infrastructure teams use version control

The use of version control by infrastructure teams has a significant impact on Stage 3 of DevOps evolution, and is also an associated practice for Stage 4. See *Stage 4: Automate infrastructure delivery* for a detailed analysis.

Stage 4: Automate infrastructure delivery

Stage 4 is where infrastructure teams take center stage. The defining practices at this stage are all about automating infrastructure delivery — what many think of as the *beginning* of a DevOps initiative.

These infrastructure automation practices appear later in the evolutionary journey than we might have expected because they are enabled by things that characterize earlier stages: normalization, reduction of variables, and expansion of the DevOps evolution beyond tech teams into the business. Success in establishing these factors in earlier stages makes it much easier to achieve success in Stage 4.

Of course, this isn't to say that infrastructure automation isn't happening in prior stages — it is, in a limited way. As we discuss in the chapter *Stage 0: Build the foundation,* the practice of managing infrastructure configurations with a configuration management tool rapidly takes root early on, when operations teams are standardizing to solve for their own needs.

The key difference in Stage 4 is that the objective driving infrastructure automation at this stage is to provide greater agility to the entire business, not just for a single team.

While this stage is gratifying — it feels like you're really doing the DevOps now — it's important to recognize that the previous stages make it possible to get to infrastructure automation. We've seen organizations try to jump immediately to this stage without going through the prior stages, and the result is frustration: It takes these organizations longer than expected to make any real progress with infrastructure automation.

By establishing clear standards and cooperation across multiple teams in earlier stages — and by achieving visible successes that build trust in automation — infrastructure teams earn the organization's blessing to develop automation that can make a clear difference to the business.

Automation for infrastructure evolves in Stage 4. It often begins with teams automating for their own needs, and then begins to align with the business. This is also the stage where infrastructure automation develops to provide uniform capabilities and services for technology delivery. The goal is to provide more reliable services and capabilities through a formal automation pipeline and workflow that couple with the services and applications built on that infrastructure.

Infrastructure teams at this stage of the DevOps journey begin to adopt agile development practices such as use of version control for both system configuration and application configurations, and adopt tooling used by application development teams. Teams at this stage also automate security policy configurations within their sphere of influence.

The nice thing about the early work that gets done in Stage 4 is that it is largely contained within the team itself, meaning handoffs and coordination can often be kept to a minimum. This allows the infrastructure team greater freedom to prioritize their work and timelines.

The defining practices for Stage 4 are:

- System configurations are automated.
- Provisioning is automated.

Associated practices are:

- Application configurations are in version control.
- Infrastructure teams use version control.

Automate system configurations

Automating system configurations and keeping them in version control is one of the first things people think of when you mention the automation pillar of CAMS. You need control over your infrastructure layer in order to achieve agility with the applications and services running on top of it. Once you can repeatably deal with account creation/removal, load balancer configuration changes, security patches and monitoring policy updates, you're no longer being held back by infrastructure that lags behind changing business and application demands.

Configurations for systems are normally built or rendered from a source of truth (version control) using an automation framework that's either off-the-shelf or internally created. Some teams have a goal of automating all change, giving them completely repeatable, rebuildable systems. Other ops teams choose to automate the most common tasks – where the return on investment is easy for other teams and management to see — leaving the complicated or infrequent changes to be dealt with in a more ad-hoc manner.

While many teams look to automation to speed up changes, that's just one benefit of infrastructure automation. There are others:

- **Overall speed.** Automated tasks should be faster than manually completed tasks.
- **Consistency.** Automated tasks follow a set process and thus produce predictable results.
- **Documented behavior.** Tasks now have a defined way they are supposed to work, so are easier to troubleshoot.
- **Portability.** With the right automation framework, teams can use content written by others to improve velocity and maintenance of their automation library.

By starting with higher ROI items, you'll effectively start paying for the investment in automation right away. This time can then be spent automating more things, simplifying processes, or improving other services built on the infrastructure.

> When you begin automating infrastructure, automate items you run into with the highest frequency across the widest swath of infrastructure components. This will have a big impact, free up your own time in meaningful ways, and buy you time to work on more complex automations. For example, automating logging or monitoring configuration for all systems will free up time while providing consistency.

Automated provisioning
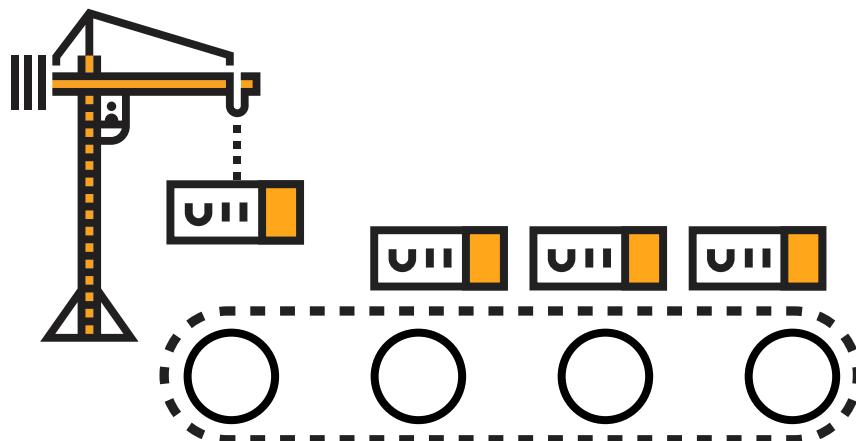
Automated provisioning is another defining practice at Stage 4. When combined with automated system configurations, you get the basis of a self-service infrastructure (covered in *Stage 5*). This significantly changes the role of operations from order-takers to owners and operators of a service-providing organization. Instead of treating each service request as a one-off, operations teams develop and offer a menu of standardized services aligned with business objectives.

Provisioning can be the automatic creation of a resource of nearly any type. Most often, teams use the word when they're talking about OS instances, network connectivity, storage, and accounts. However, some teams take automated provisioning much further, with hooks into pager systems, DNS, CDN, load balancers, databases and more.

As with system configurations, it's best to begin with the most frequently requested item; gain some wins, consistency and time savings; and then move onto the next most frequent request. As with most steps in the DevOps evolution, you want to choose tasks that will win the confidence — even gratitude — of others both inside and outside your team.

At this point, provisioning could be done with a framework, or even a set of shared scripts and utilities stored in version control. The key is that the team can level up and perform the provisioning in an automated way. A key acceptance criteria for good automated provisioning is that the customer can't tell who was assigned the provisioning tasks (implying that no special adjustments were made by individual people on the infrastructure team).

We did not ask survey respondents about storing application configurations in service discovery and key value stores such as etcd, Consul, and ZooKeeper. All of these are great for real-time look-up and provide system-wide health checks, discovery and routing. Some teams have built versioning workflows around these tools, while others work with them as a live system.

Application configurations are in version control

By now your application source code is in version control. Great. What about the configuration for your application? Many applications, whether off-the-shelf or developed in-house, are built using some patterns from a **12-factor application**. Even if an application doesn't follow all 12, configurations are normally externalized, not hard-coded into the application.

The settings that make your application deployment operate in your environment are critical. Gone are the days when an administrator would log directly into a system and then hand-edit a property file. Application configurations should be versioned, auditable, contain history, and ideally, the reasons why they've been changed.

Separating your app configurations from source code allows for the same source code and artifact to be deployed and validated across multiple environments, with the only change being in configuration.

This separation becomes paramount when you want to provision individual development environments or move towards self-service. It's not efficient to recompile an application for each and every user, or hardcode their parameters into it. Thus, separating code from configuration data allows for more rapid deployments, updates, and validation.

Infrastructure teams use version control

One of the important shifts that happens in Stage 4 is infrastructure teams adopting good development practices, such as the use of version control. As previously noted, the use of version control for all production artifacts is highly correlated with IT performance. It's the first step to continuous delivery of your infrastructure code. Use of version control makes it easy to recreate environments for testing and troubleshooting, boosting throughput for both Dev and Ops. It also reduces the time to recover if an error is identified in production. You can quickly either redeploy the last good state, or fix the problem and roll forward, all with history and auditing capabilities.

Contributors to success in Stage 4

Two key practices have a significant impact on Stage 4:

- Automate security policy configurations.
- Make resources available via self-service.

We'll discuss each of these below.

Automate security policy configuration

Beyond having security policies, organizations often have external constraints that require demonstration of and adherence to security policies via measured controls. Those external forces can be an audit committee, Sarbanes-Oxley, Payment Card Industry Standards (PCI), NIST, General Data Protection Guidelines (GDPR) and myriad other regulatory standards.

At this point in the DevOps evolution, most security policy automation occurs at the team level, with the primary objective being to ensure that interaction with auditors is kept to a minimum. It's done this way for practical reasons, as the team level is where handoffs are minimal and ROI is immediately realized. As the organization improves and evolves, it will start to solve the security policy problem more broadly.

The best way to adhere to security policy is to know whether you're compliant, and fix systems when you're non-compliant. Organizations evolving on their DevOps journey do just that.

There's an evolutionary cycle for automating security policy, and it often starts with a single team member automating some policy by writing a scanner for the policy. Then she might write a corrector or enforcing script. From there, the script might generate a report that can be archived or shared among security teams or auditing staff.

As security policy automation gets a bit more mature, the use of configuration management systems emerges. Configuration management enables policy to be enforced upon system convergence, and reports to be handled in a standard way. Placing security policy into infrastructure configuration management acts as a normalizing function for the team — meaning any team member can update or improve enforcement of policy via the codebase.

From an application development team's perspective, the infrastructure it relies on must be compliant with security policy. A configuration management tool will consistently enforce the correct policies underneath the applications. This doesn't mean application delivery teams have nothing to do, though. Some teams may run static analysis on code via their continuous integration pipelines. Some teams will also use external tools to scan their applications for vulnerabilities such as the **OWASP top 10** via external tools.

Resources are made available via self-service

This practice is discussed in the next chapter *Stage 5: Provide self-service capabilities*.

Stage 5: Provide self-service capabilities

To move to Stage 5, an organization must have multiple departments committed to providing IT capabilities as a service to the business, rather than treating IT as a cost center that executes work orders. These departments include development, operations, security, ITSM and other functional areas.

In this last stage of the DevOps evolution, we see benefits to the organization multiply enormously as successful collaboration across functional boundaries accelerates. These gains are seen in several distinct areas:

- Application architecture moves beyond standardizing on technologies and begins to evolve towards working with and supporting cloud migration, container adoption, and proliferating microservices.

- Security policy automation moves from servicing the needs of a team to becoming the baseline for how security and compliance are measured throughout a department, or even the entire organization. Additionally, automated provisioning advances to provisioning of whole environments for developers, testers and other technical staff.

Once you start succeeding across multiple functional boundaries, the pillars of DevOps — Culture, Automation, Measurement, and Sharing — become more pervasive across the organization.

The two defining practices for Stage 5 are:

- Incident responses are automated.
- Resources are available via self-service.

The two associated practices in this stage are:

- Rearchitect applications based on business needs.
- Security teams are involved in technology design and deployment.

Incident responses are automated
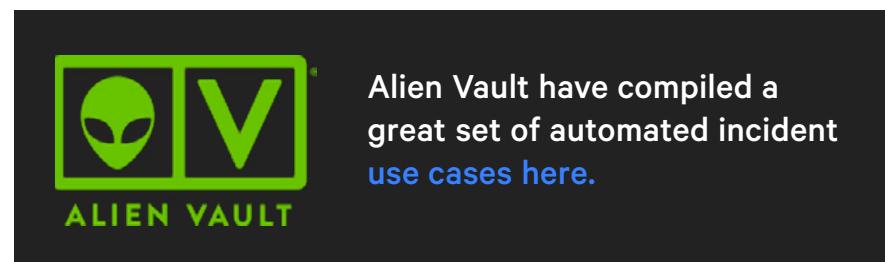
Manually responding to critical incidents is expensive in multiple ways. It's expensive in terms of engineering attention and focus, and it's expensive in terms of the time it takes to detect, identify and remediate the incident. Particularly when dealing with intrusions or malware, a response can be very expensive indeed if it doesn't completely remediate the issue. Missing just one infected machine or command-and-control server address is all it takes to render the entire response useless.

All of this means there's a huge amount of value to be gained by automating incident response. Automating eliminates unnecessary distractions, improves time to remediation by reducing handoffs, and ensures that your remediation processes are consistently applied.

Fully automating your incident response system is a daunting task, but you don't need to automate for every type of incident. Instead, think about your automation as being there to augment human judgement. Focus on the processes and systems that let you identify issues, as well as those you deploy when responding. Make it simple for your operators to get to whatever data they need to form a judgement, and once they've done so, automate response processes — things like adding a malicious IP to all your firewalls across your infrastructure; collating data for later forensics; or completely isolating an infected machine.

For those of you in smaller environments, it may not be immediately obvious why automated incident responses don't come into play until Stage 5. It's because of significant organizational and process barriers in enterprises that often stop incident-response people from achieving complete remediation. Lack of access to metrics and logs; having to file tickets to get others to validate firewall rule changes; the need for signoffs from service owners; the inability to push final changes through to production — all of these barriers to fast feedback and action cycles must be removed in order to add automation to your incident responses. Many enterprises haven't reached this point.

Some fundamental technical capabilities need to be in place to deliver automated incident responses, but it's equally important to have a collaborative relationship with your security team. Bring the security team into the development lifecycle early, and start small by collaboratively automating response to one specific type of incident that crosses functional boundaries.

Alien Vault have compiled a great set of automated incident use cases here.

## Resources available via self-service
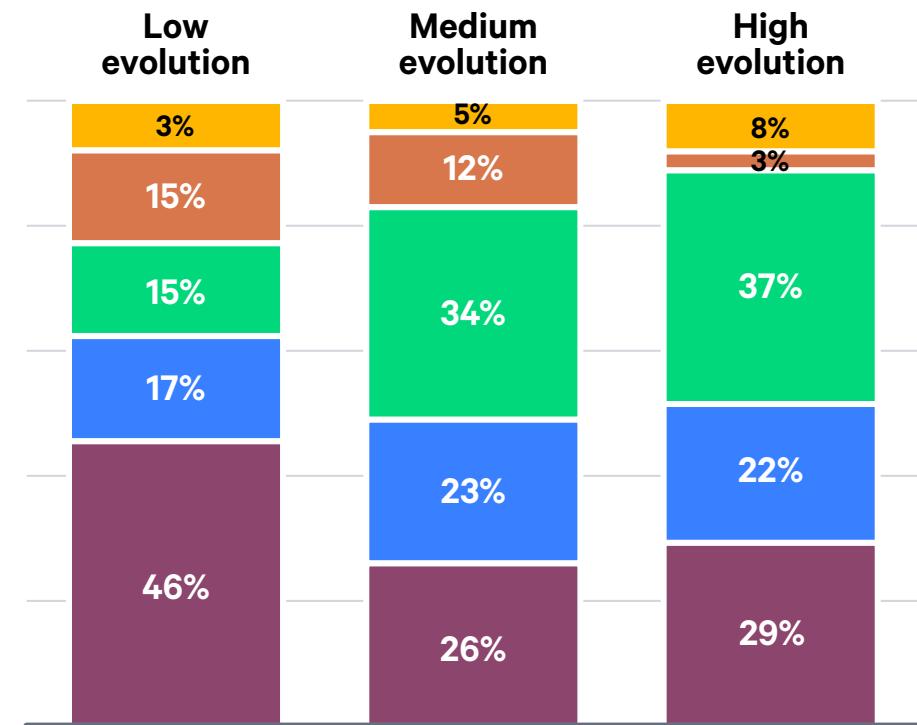
Great self-service systems are an incredible enabler across a business. The more you empower people to work at their own speed without having to wait — for tickets to be approved, license keys to be obtained, network settings to be updated or required configurations to be applied — the less frustration individuals will experience, the easier it will be to standardize configurations, and the more predictable work will be. Having to switch focus while you wait for someone else kills progress, not to mention enthusiasm for the task at hand.

It's important to note that you can and should be working towards self-service well before this stage, and it's absolutely possible to deliver real value incrementally before you reach the point of a comprehensive self-service catalog. Teams should build self-service systems for themselves and then their adjacent teams, next expanding outwards through the organization. This is exactly what the data shows successful teams do.

Comparing our Low and High evolutionary cohorts, we see this exact shift from a high proportion of self-service systems for internal team usage towards multiple teams collaborating to deliver systems that will be broadly consumed across the organization.

Automation progress by evolutionary scale

- 🟧 Most services are available via self-service.
- 🟫 A few key services are available via self-service.
- 🟩 Teams are collaborating to automate services for broad use.
- 🟦 Teams are automating services they control, for others' needs.
- 🟪 Teams are automating services they control, for their own need.

Low evolution	Medium evolution	High evolution
3%	5%	8%
15%	12%	3%
15%	34%	37%
17%	23%	22%
46%	26%	29%

Rearchitect applications based on business needs

The rearchitecting of applications in response to the business shows up in Stage 2 as a contributor to success, but our hypothesis is that there's a difference between what this means in the two different stages. We believe that in Stage 2, the activity is primarily technology stack standardization — for example, consolidating down to one or two database systems or middleware systems. We suspect that in Stage 5, rearchitecting applications for the business means more fundamental surgery is performed: adopting the 12-factor app methodology, moving to microservices, adopting containers or replacing components with cloud services.

Security teams are involved in technology design and deployment

As we discussed in *Stage 0: Build the foundation*, shifting left is about bringing more teams into the development and delivery process — for example, quality, security, database, audit and networking. Most teams begin the leftward shift by addressing deployment pain, which is the functional boundary between Dev and Ops. This represents one step to the left, whereas involving other teams such as quality and security means shifting the focus several steps further to the left, well before deployment. So it makes sense that getting security teams involved happens later in the DevOps evolution, after more acute problems have been addressed.

In our 2016 State of DevOps report[5], we found that high performers spend 50 percent less time remediating security issues than low performers. This is because they build security into the software delivery cycle as opposed to retrofitting security at the end.
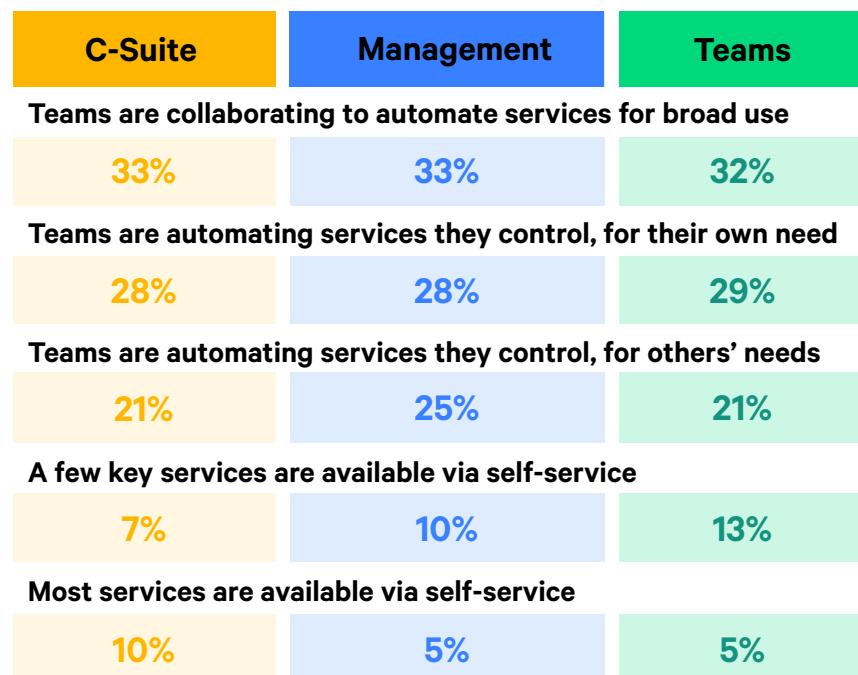
At Stage 5, security teams should get involved early in the software development process by:

- Conducting a security review for all major features while ensuring that the security review process does not slow down development.

- Integrating information security into the daily work of the entire software delivery lifecycle. This includes providing input during the design of the application, attending software demos and providing feedback during these demos.

- Testing security requirements as a part of the automated testing process.

- Creating pre-approved, easy-to-consume libraries, packages, toolchains and processes for developers and IT operations to use in their work.

---

5 The 2016 DevOps Survey is the property of Puppet, Inc. and DevOps Research and Assessment, LLC. All rights reserved. Authors: Dr Nicole Forsgren, Jez Humble, Gene Kim, Alanna Brown, Nigel Kersten

Perceptions of automation within the organization

Interestingly, there's generally very little difference in perception between the C-suite, managers and teams when it comes to *automated* self-service systems. This consistency can be explained by the fact that by their nature, there is wide awareness and impact of such systems, particularly those that are broadly consumed. The degree to which most services are available by self-service does show a difference in perception, one that is consistent with our findings around unrealistic optimism among C-suite executives.

C-Suite	Management	Teams
Teams are collaborating to automate services for broad use		
33%	**33%**	**32%**
Teams are automating services they control, for their own need		
28%	**28%**	**29%**
Teams are automating services they control, for others' needs		
21%	**25%**	**21%**
A few key services are available via self-service		
7%	**10%**	**13%**
Most services are available via self-service		
10%	**5%**	**5%**

Contributors to success in Stage 5

For any item you're looking to deliver via self-service, map out your existing manual process along with all required approval workflows and look for optimizations. It's common for organizations to institute manual approval steps in response to incidents and then rarely revisit them as environments evolve and new areas of automation become possible.

A common anti-pattern we've seen is for organizations to make significant investments in their self-service platform, yet use it to deliver only uncustomized payloads. You may be improving cycle times for your users by making it trivial to install software, but if you don't put the work in to customize catalog items *for your business*, you won't see the truly significant gains that are possible.

A common blocker for organizations is to focus on self-service platforms that are easy for a human to drive, but difficult to be consumed by an automated pipeline. Whether teams are building something themselves or deploying an off-the-shelf self-service catalog, it's critical that the platform can truly operate as an underlying substrata for other solutions such as CI/CD pipelines.

The value your operations and security teams should provide is their opinionated expertise, and the more you can bake this expertise into deployed software, the better your results will be. This doesn't mean that you ignore the needs of the users who consume the software: you need to understand what they're trying to achieve, have empathy for the environment they work in, and balance that with operational and security requirements. In many ways, this requires a shift towards a product-manager mindset.

This is why it's important for teams to start with self-service for their own use. Their own problems are the problems they understand best, so there's naturally more empathy for the user. Serving the customer you know best provides a great opportunity to start learning how to build self-service.

It becomes progressively more difficult to understand the user's needs as the user gets further and further away, organizationally and functionally, from the team building the service. While empathy and a product-manager mindset helps bridge these gaps, ultimately you're going to need to build these services in collaboration with the teams who are delivering adjacent functionality, as well as the teams who will consume your service.

App developers deploy testing environments on their own

When application development teams can deploy testing environments on demand, they are more productive (because they don't have to wait for a new environment, go back and forth via tickets, etc.), and application delivery is faster. Ops teams also benefit from providing this self-service capability, because they can then spend more time optimizing the system instead of provisioning systems. Once developers can deploy testing environments, it becomes much easier to enable automated deployments.

Automate security policy configurations

This practice proves to be significant in the later evolutionary stages of DevOps (see *Stage 4: Automate infrastructure delivery*). Why? Because security policy configurations are one of the harder things to automate.

Despite the difficulties, it's well worth automating security policy configuration. It's far cheaper to prevent and mitigate security issues in the application design than it is to react to them in production. **Automated incident response** (as covered above) is just one aspect of this more fundamental collaboration across functional boundaries.

That's why there's a "shift-left" movement in security right now. Security considerations are shifting from being primarily operational concerns in production to being incorporated in application design and build. It's a counter-movement to the traditional way of building software, where entirely separate teams focus on different parts of the build cycle. In this scenario, it's all too easy for each team to ignore or forget the concerns of other teams — at least until an incident occurs once the application is in production. So, just as bringing development and operations teams together enables operational considerations to be part of application design, the same is true for security teams and their areas of responsibility — they need to be included early in the software design cycle.

Success metrics for projects are visible

Stage 5 is the first stage where we see a significant number of respondents saying their organization makes success metrics visible. This finding stands in sharp contrast to the common advice that teams should implement dashboards early in their DevOps process.

However, it's not surprising. You need automated mechanisms in place to share metrics broadly, and that degree of automation is normally achieved in Stage 4, after a lot of groundwork has been done in the prior stages.

Once success metrics are clearly defined and visible to everyone, you'll find it's far easier to get agreement on what needs to be addressed next for the health of the business.

Conclusion

Every year, the State of DevOps Report teaches us something new. This year, our data has showed us that while there are many individual paths through a DevOps transformation, there are ways to achieve and scale success faster. Organizations have a choice: they can choose to be systematic about how they evolve, or they can take a more scattershot approach. Of course, it's possible that even an ad-hoc approach could work, but what we see among organizations that have reached the highest levels of DevOps evolution is that they didn't get there by accident.

We're thrilled to be able to provide something so concrete and useful to teams that are working hard to improve how they work and their responsiveness to the business.

We hope this report has given you some good ideas, and perhaps helped you realize you're moving forward on your DevOps journey better than you thought.

No matter what your response, we'd love to hear from you. Tell us about your challenges and triumphs, and ask us any questions you may have.
You can reach us at: **devopssurvey@puppet.com.**

Methodology

Full Methodology can be viewed **here**

The five stages of DevOps evolution

One of the primary goals of this research was to understand the adoption patterns of DevOps practices as organizations evolve. Our hypothesis was that there are distinct stages in a DevOps evolution and specific practices that enable organizations to scale success beyond isolated teams. We tested a large group of DevOps practices. Five factors representing five stages of DevOps evolution success were created using factor analysis. The two most distinctive factors (those that had high statistical significance and the least amount of overlap across other stages) were chosen to represent each stage.

In all, ten defining practices were determined based on factor analysis and these were used to create five positions along the DevOps evolutionary journey. These practices were then rank-ordered based on the percentage of organizations that consistently adhere to them within a stage. The higher the percentage of organizations that frequently adhere to a practice within a stage, the earlier it is in the evolutionary journey. The lower the percentage of organizations that frequently adhere to a defining practice within a stage, the more advanced that practice is on the evolutionary journey. At each stage, the remaining attributes tested were regressed against the defining practices for a particular stage. The result is the creation of a set of key contributors to success at each stage of the evolutionary journey.

The evolutionary scale

In order to determine each organization's position within the DevOps journey, we determined whether or not an organization consistently adhered to the defining success practices at each stage of evolution. We then summed all the defining practices completed across all stages, and produced a score that represented a given organization's position on the evolutionary scale. Organizations were next placed in one of three groups — Low, Medium or High — based on the number of defining practices they consistently perform.

Organizations that consistently adhere to all defining practices are considered highly evolved (High), while those that consistently adhere to only a few are positioned at early stages of the evolutionary journey (Low). If an organization performed the key practices in Stage 1 frequently, but did not frequently perform any of the key practices in Stages 2 through 5, we deemed that organization was not highly evolved. If an organization frequently performed all key practices in Stage 1, as well as all key practices in Stages 2 through 5, we deemed that organization was highly evolved.

Target population and sampling method

Our target population for this survey consists of practitioners and leaders working in, or closely with, IT, and especially those familiar with DevOps. Even though we don't have a master list of these people, we were able to describe their characteristics. However, we don't know exactly where they are, how to find them, or how many of them exist, so we used two methods to obtain respondents:

- **Snowball sampling.** This means we promoted the survey via email lists, online promotions and social media, and also asked people to share the survey with their networks, growing the sample like a snowball. This sample is therefore likely limited to organizations and teams that are familiar with DevOps, and as such, may be doing some of it. We also extended our survey globally to the Asian Pacific region, Europe, the Middle East and Africa, offering it in four languages other than English: French, German, Japanese and Malaysian. These languages were chosen because they are first languages in regions where we know that interest in DevOps is high.
- **Panel sample.** The snowball sample was supplemented with a panel sample. These were acquired from third-party panel providers, and their presence reduces bias in the overall sample. In this particular instance, our third-party panel provider nurtures and maintains a quality, engaged membership panel built to support its market research clients and to benefit non-profit organizations. The panel provider's unique approach to recruiting yields a highly engaged group of people who, as respondents, are dedicated to helping our market research clients fulfill their information needs. The panel provider's unique non-profit recruitment method enables the firm to source C-suite executives, directors, and managers who have key decision-making authority. In addition to their non-profit relationships, the firm also utilizes trade association partners to help drive certain audiences into online surveys. This approach provides access to the appropriate sample for each survey. The advantages offered by this panel are core to our differentiation.

Statistical analysis methods

- **Factor Analysis.** The five stages of DevOps evolution are derived with a data-driven approach, using factor analysis.
- **Regression analysis.** When predictions or impacts are cited linear regression (stepwise method) was used.
- **Study design.** This study employs a cross-sectional, theory-based design.

Author biographies
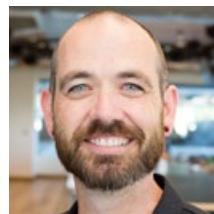
Andi Mann is chief technology advocate at Splunk and an accomplished digital business executive with extensive global expertise as a strategist, technologist, innovator, and communicator. For over 30 years across five continents, Andi has built success with Fortune 500 corporations, vendors,governments, and as a leading research analyst and consultant. Andi is also a sought-after commentator on business technology. He has been published in USA Today, The New York Times, Forbes, CIO, and The Wall Street Journal; presented at Gartner ITxpo, VMworld, CA World, Interop, Cloud Expo, and DevOps Summit; and participated and hosted interviews for radio, television, webcasts, podcasts, and live events.

Michael Stahnke is director of engineering at Puppet. He's held a few roles and been a part of the company growing from 35 to 520+ employees. While staying near the domains of release engineering, operations, and community, he's been in leadership for most of the last decade. His interests are building teams, mentoring team members, driving change with customers, and playing with his son. He came to Puppet from Caterpillar, Inc. where he was an infrastructure architect, infrastructure team lead, and open source evangelist. Michael also helped get the Extra Packages for Enterprise Linux (EPEL) repository launched in 2005, authored Pro OpenSSH (Apress, 2005), and writes with some frequency about technology and computers. Follow Michael @stahnma on Twitter and Medium.

Alanna Brown is director of product marketing at Puppet, where she's had the privilege of helping Puppet grow from a small startup to a global brand with thousands of customers around the world. She conceived and launched the first anual State of DevOps Survey in 2012, and has been responsible for the survey and report since then. In addition to heading up DevOps research, Alanna is also responsible for driving the go-to-market strategy for Puppet's product portfolio and cultivating relationships with customers to drive DevOps adoption.

Nigel Kersten is the chief technical strategist at Puppet, where he has held a variety of roles, including head of product, CTO and CIO. He came to Puppet from Google headquarters in Mountain View, Calif., where he was responsible for the design and implementation of one of the largest Puppet deployments in the world. Nigel has been deeply involved in Puppet's DevOps initiatives, and regularly speaks around the world about adoption of DevOps in the enterprise and IT organizational transformation.

About Puppet

Puppet is driving the movement to a world of unconstrained software change. Its revolutionary platform is the industry standard for automating the delivery and operation of the software that powers everything around us. More than 40,000 companies — including more than 75 percent of the Fortune 100 — use Puppet's open source and commercial solutions to adopt DevOps practices, achieve situational awareness and drive software change with confidence. Headquartered in Portland, Oregon, Puppet is a privately held company with more than 500 employees around the world.

Learn more at puppet.com.

About Splunk

Splunk Inc. (NASDAQ: SPLK) turns machine data into answers. Organizations use market-leading Splunk solutions with machine learning to solve their toughest IT, Internet of Things and security challenges. Join millions of passionate users and discover your "aha" moment with Splunk today: splunk.com

Report Sponsors

About AWS

For over 12 years, Amazon Web Services has been the world's most comprehensive and broadly adopted cloud platform. AWS offers over 125 fully featured services for compute, storage, databases, networking, analytics, machine learning and artificial intelligence (AI), Internet of Things (IoT), mobile, security, hybrid, virtual and augmented reality (VR and AR), media, and application development, deployment, and management from 55 Availability Zones (AZs) within 18 geographic regions and one Local Region around the world, spanning the U.S., Australia, Brazil, Canada, China, France, Germany, India, Ireland, Japan, Korea, Singapore, and the UK. AWS services are trusted by millions of active customers around the world—including the fastest-growing startups, largest enterprises, and leading government agencies—to power their infrastructure, make them more agile, and lower costs. To learn more about AWS, visit **aws.amazon.com**.

About Cloudability

Cloudability is a multi-cloud, True Cost™ management platform that delivers accurate cloud financials, empowering companies to run their cloud with financial and operational excellence. Our True Cost™ platform is designed to enable visibility, optimization and governance for every VM, container and serverless workload, application, department and user. Cloudability helps thousands of global enterprises and cloud-native companies leverage data science, analytics, machine learning and automation to improve margins, reduce cloud spend waste, and get solutions to market faster. Headquartered in Portland, Oregon, Cloudability is a venture-backed company with offices around the world. Learn more at **cloudability.com**.

Report Sponsors

About Cognizant

Cognizant is one of the world's leading professional services companies, transforming clients' business, operating and technology models for the digital era. Our unique industry-based, consultative approach helps clients envision, build and run more innovative and efficient businesses. Headquartered in the U.S., Cognizant is ranked 195 on the Fortune 500 and is consistently listed among the most admired companies in the world. Learn how Cognizant helps clients lead with digital at **cognizant.com** or follow us **@Cognizant**.

About CyberARK

CyberArk (NASDAQ: CYBR) is the global leader in privileged access security, a critical layer of IT security to protect data, infrastructure and assets across the enterprise, in the cloud and throughout the DevOps pipeline. CyberArk delivers the industry's most complete solution to reduce risk created by privileged credentials and secrets. The company is trusted by the world's leading organizations, including more than 50 percent of the Fortune 100, to protect against external attackers and malicious insiders. A global company, CyberArk is headquartered in Petach Tikva, Israel, with U.S. headquarters located in Newton, Mass. The company also has offices throughout the Americas, EMEA, Asia Pacific and Japan. To learn more about CyberArk, follow **@CyberArk** or visit **cyberark.com**.

Report Sponsors

About Diaxion

In a market grappling with demanding and complex transformation, Diaxion helps business leaders achieve high performing outcomes, with reduced risk. Leveraging our unique insight, Diaxion leads with the business challenge to unlock IT requirements that deliver the right outcome through people process and technology. We do this because we believe business goals should drive IT, not be limited by it. With over 18 years experience designing and implementing transformation projects, Diaxion is the choice of corporate Australian firms. Our highly skilled staff guarantee right fit IT to deliver high performing business outcomes, while our methodology, proven in over 2000 projects, ensures a robust and de-risked process. Diaxion, business powered IT. **diaxion.com**

About Eficode

Eficode is building the future of software development. We help our customers become high performance software organizations by providing DevOps platform and consultancy services. Eficode Root DevOps platform is a complete, state-of-the-art software production line tailored to perfectly fit your organization's needs. It comes as a turn-key solution with flexible options for support, maintenance, hosting and continuous production line development. Eficode is a privately held, fast growing international company with more than 240 employees. The company is headquartered in Helsinki with multiple branch offices outside of Finland. Learn more at **eficoderoot.com**

2018

State of DevOps Report

Presented by:

puppet + splunk>