DZone

THE DZONE GUIDE TO

DevOps

Continuous Delivery and Automation

VOLUME IV

RESEARCH PARTNER SPOTLIGHT

ca technologies

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

497 respondents completed our 2017 Continuous Delivery survey. The demographics of the respondents include:

- 19% of respondents work at organizations with over 10,000 employees; 20% at organizations between 1,000 and 10,000 employees; and 26% at organizations between 100 and 1,000 employees.
- 45% of respondents work for organizations headquartered in Europe, and 30% for organizations based in the US.
- On average, respondents had 15 years of experience as IT professionals; 27% had 20 years or more of experience.
- 42% of respondents identified as developers/ engineers, and 27% identified as developer team leads.
- 82% of respondents work at companies using the Java ecosystem, and 70% at companies using client-side JavaScript.

TIME TO FACE THE STRANGE

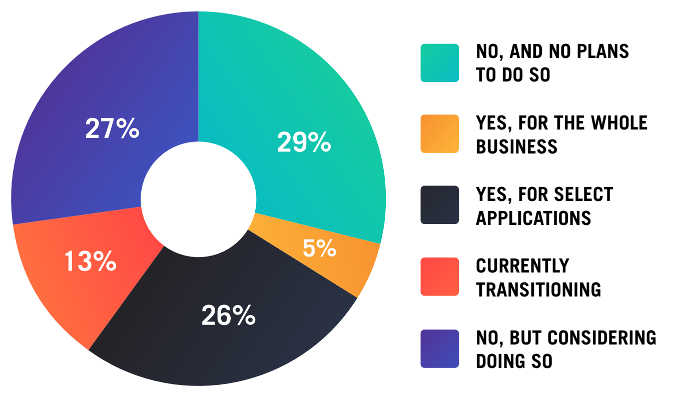
DevOps has had some steady growth over the past year, as more and more developers and organizations work towards automation and cross-departmental collaboration. 41% of respondents said their organization has a dedicated DevOps team, up 7% over last year's statistic (which had not changed from the year before). Performance issue detection in the software delivery process increased 5% year over year, while automated performance testing increased 6% and automated feature validation increased 4%. The number of respondents who said they believe their organization has achieved Continuous Delivery "for some projects" increased 9% from 2016, and there was an 8% swing in respondents who said that CD is a focus for their organization. Microservice architectures are used 7% more compared to last year, and container adoption is up 8%. The use of version control tools reported in QA and Production have increased 15% and 18% respectively, and the use of CI tools in those departments increased 17% and 13%.

Despite this growth, there are some areas of stagnation in CD results. From 2016, there was no statistically significant change in respondents' estimate of their mean time to recovery (between hours and days) or mean time between failures (between hours, days, and months). Most CD pipeline pain points also remained the same from last year, with the exception of automated testing, which dropped 7% as a pain point, and the deployment process and regression testing, which each dropped 4%.

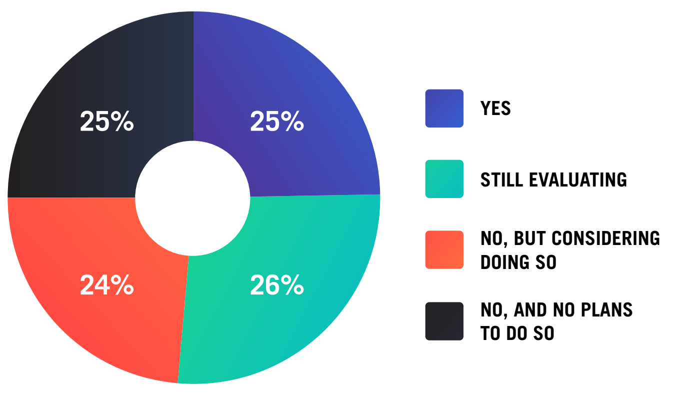
SIZE MATTERS

With regards to having Continuous Delivery implemented in an organization, and having Continuous Delivery implementation be a focus for an organization, company size plays a sizable role. Respondents' belief that their company has achieved Continuous Delivery trends upwards as the size of their organizations increase. 51% of respondents working at companies under 100 employees think their company has either fully or partially achieved Continuous Delivery, versus 60% of respondents who work at companies larger

HAS YOUR ORGANIZATION MOVED TOWARDS A MICROSERVICE ARCHITECTURE?



HAVE YOU OR YOUR ORGANIZATION ADOPTED CONTAINER TECHNOLOGY (E.G. DOCKER)



than 10,000 employees. And only 41% of respondents working at sub-100 employee organizations say that CD is a focus for their company, 15% less than those who work at companies between 100 and 9,999 employees and 30% less than those who work at 10,000+ organizations.

This goes hand-in-hand with larger companies' ability, and likely need, to have dedicated DevOps teams. Only about one in four respondents (27%) at an organization with fewer than 100 employees said their company had a dedicated DevOps team, compared to almost half (45%) of respondents between 100 and 9,999 employees and 62% over 10,000.

STOP... RECOVERY TIME

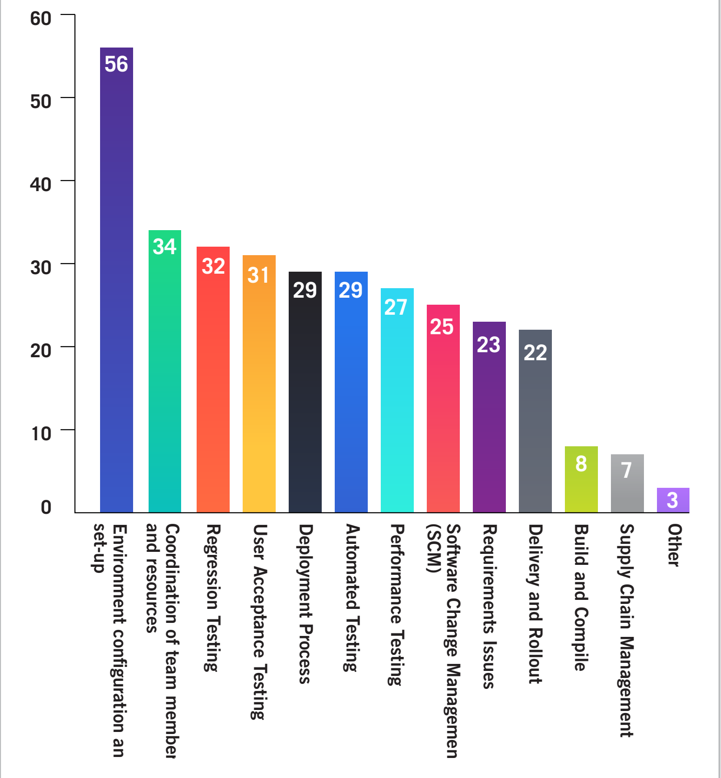
Overall, respondents' estimated an average mean time to recovery of just under 19 hours, with estimates ranging from just minutes to 40 days, but with most estimates falling somewhere between 2 and 24 hours. Several factors come into play here, which can drastically change the mean-time-to-recovery estimates. Respondents whose organizations have push-button/automated deployment estimated recoveries happen twice as fast as organizations that don't (12 hours versus 24 hours). Those respondents using or evaluating container technologies estimated about 20% less time to recover than non-container users (17 hours versus 21 hours). Microservice usage greatly affected these estimates. Respondents who said their organization has not moved to microservice architectures estimated, on average, a 29-hour mean time to recovery; respondents at organizations currently transitioning estimated 12 hours; and respondents at organizations using microservices for some or all of their applications estimated a 7 hour mean time to recovery.

BRING THE PAIN POINTS

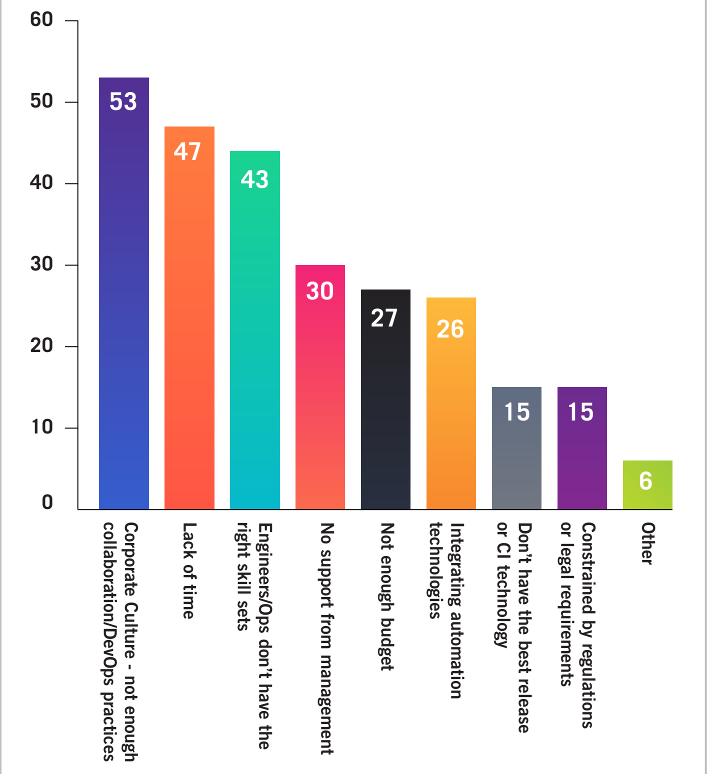
We asked our survey-takers who said they believed their organization had implemented Continuous Delivery in some capacity what their biggest pain points were in the CD pipeline, and likewise asked those respondents who did not think their organization had achieved CD status what they thought were the main barriers to adopting CD. As mentioned earlier, most pain points appeared to be just as painful this year as they were last year. The most common pain points were environment config and setup (56%), coordination of team members and resources (34%), and regression testing (32%). Most other pain points were experienced by roughly a quarter of respondents, with the exceptions of build and compile (8%) and supply chain management (7%).

Regarding barriers to adoption, this year's respondents again answered similarly to last year's results, though all barriers did drop somewhat. The biggest changes here were "no support from management," which dropped 7% from last year, and "Engineers/Ops don't have the right skill sets," which dropped 5%. All others dropped between 2 and 4 percent from last year. So, while progress is being made to make CD easier to adopt and manage, there is still certainly plenty of room for improvement.

WHAT ARE YOUR BIGGEST PAIN POINTS IN THE CONTINUOUS DELIVERY PIPELINE?



WHAT ARE YOUR MAIN BARRIERS TO ADOPTING CONTINUOUS DELIVERY?



HOW TO AVOID DEAD-END DELIVERY

In this year's survey of DZone's audience, 48% of respondents believe they have not adopted Continuous Delivery, and 38% believe they have adopted Continuous Delivery only for some projects. Just over half of respondents (54%) are currently focused on implementing Continuous Delivery in their companies, so what's keeping them from reaching that goal, and what's keeping the other 46% from trying to implement it? Turns out, there are a lot of obstacles that can prevent developers or managers from making headway in their adoption efforts. To learn more about them, we're going to play a little game...

Imagine you're a plucky young startup with everything to prove, or perhaps part of a seasoned corporation that's been around the block and is ready for a transition to more modern methodologies. Can you achieve Continuous Delivery without running into any of these barriers? A-maze us!

CD
30%
Continuous Delivery 4 Dummies
45%
53%
47%
27%
27%

NO SUPPORT FROM MANAGEMENT

30%

While the benefits of Continuous Delivery are well-documented, the initial investment into tooling and training can put a lot of managers off the concept. For successful Continuous Delivery, it takes both management and frontline developers to believe in the benefits and be devoted to working towards them.

LACK OF SKILL

45%

Continuous Delivery is very difficult without adopting several new tools, and impossible without changing processes. Learning all these new technologies can be incredibly difficult, especially if there's no prior knowledge on your team.

CORPORATE CULTURE

53%

Company culture can be difficult to establish, and even more difficult to change. If a culture has built silos that separate teams from each other, it's going to be very difficult to foster the collaboration, flexibility, and speed that Continuous Delivery demands.

LACK OF TIME

47%

Jamie Zawinski once famously said, "Linux is only free if your time has no value." Unfortunately, in the Enterprise, whether you go for an open source or proprietary tool, implementing DevOps tools and processes take a lot of time that you may not have, especially if you have delivery dates looming.

INTEGRATING AUTOMATION TECHNOLOGY

27%

The knowledge to put the pieces of your build pipeline together may not exist in your organization, and even if it does it could take a lot of work to integrate these tools, especially if those tools are open source and you don't have budget to spring for a proprietary product.

LACK OF BUDGET

27%

If your organization doesn't have time to go the open source route, you'll need to use proprietary solutions, which you may not have the budget for, especially if you're a startup without VC or time to spare. No money, mo' problems.

Better Code for Better Requirements

BY **ALEX MARTINS**
CTO/ADVISOR - CONTINUOUS QUALITY AT **CA TECHNOLOGIES**

QUICK VIEW

01 Code may be of the highest quality, but if it's not reflecting what was specified in the requirements, you may have built perfectly useless code.

02 By preventing defects from being written into the code, quality is thus built into the application from the onset.

03 The use of a CAD-like tool in software engineering not only accelerates the software lifecycle, but also ensures developers are building the right things.

Quality is a very hot topic in the DevOps and Continuous Delivery era. "Quality with speed" is the theme of the hour. But most development and testing teams have different views on what quality means to them.

Looking back at my days as a professional developer, I remember being tasked to follow the company coding style guide. This described the design principles and the code convention all the developers should follow so that we wrote consistent code. Thus when a change request came in, anyone could read the code and make the edits, and we could minimize maintenance.

Then there were the weekly reviews where we would get together with a peer and go through the code to ensure we understood it and were following the style guide. If the code checked out, we then thought we had quality code.

But did that mean the applications we built were high quality? No!

SETUP

I've worked with plenty of agile dev teams that have adopted DevOps and achieved Continuous Delivery. These teams typically create basic, sometimes throwaway code just so they can quickly push a build out to users to get feedback and make quick adjustments. Of course this approach generates technical debt; however, at this stage, speed is more valued than code that is perfectly written according to any style guide.

Upon seeing positive feedback from users, these teams start constantly refactoring the code to keep technical debt at manageable levels. Otherwise, all the speed they've gained to roll out the first builds is lost as the code grows and becomes hard to change due to the technical debt accrued. The ultimate consequence: team capacity and velocity for future iterations is decreased, taking everyone back to square one – with not only less-than-adequate code, but also an application that users don't like.

So to keep improving their code in such a mature environment, these teams use code quality tools to profile the code and determine where to focus refactoring efforts first. This helps them **build things right**. But no matter how good the code gets, the user may still think the application sucks, simply because they were not **building the right things** in the eyes of the user. There is a difference between the two, and in my experience, this is a huge gap in most Continuous Delivery initiatives.

So what's the missing link? Requirements. The code may be of the highest quality, but if it's not reflecting what was specified in the requirements, you may have built perfectly useless code.

Louis Srygley has an apt description for this:

"Without requirements and design, programming is the art of adding bugs to an empty text file."

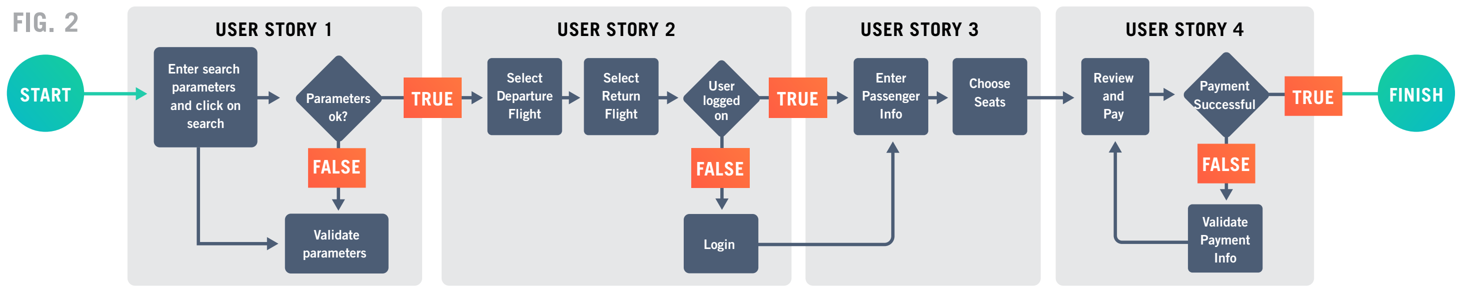
BUILDING THINGS RIGHT VS. BUILDING THE RIGHT THINGS

The use of diagrams such as visual flowcharts to represent requirements is something that helps analysts, product owners, developers, testers, and op engineers. Diagrams are a great communication tool to remove ambiguities and prevent misinterpretations by each of these stakeholders – ultimately leading to fewer defects in the code, as the visual flowcharts enable all stakeholders to have a common understanding from the get-go.

The key is to change our mindset of using "testing" as the only means to achieve application quality.

With Continuous Delivery we're realizing that although we can run unlimited automated tests at all levels to find defects, this approach will always be reactive and more costly than tests that always pass because there were no defects. That means we have **prevented**

FIG. 2



defects from being written into the code, which consequently means we have **built quality into the application** itself.

Martin Thompson says it best:
"It took us centuries to reach our current capabilities in civil engineering, so we should not be surprised by our current struggles with software."

We are on the right track. Tools have evolved and continue to evolve at a never-before-seen pace. The area that has been lagging in terms of advanced and easy-to-use tooling is the requirements-gathering and definition process. Martin Thompson also has a good quote on that:

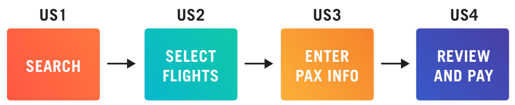
"If we look to other engineering [disciplines], we can see the use of tooling to support the process of delivery rather than imposing a process on it."

Look at civil engineering. CAD (computer-aided design) software revolutionized the designing of buildings and structures. We've been missing a CAD-like tool for software engineering, but now we are at a point where we have highly advanced and easy-to-use solutions to fill that gap.

BUILDING QUALITY INTO THE CODE = APPLICATION QUALITY

It is very common today for a product owner to draw an initial sketch on a whiteboard describing what she wants built. That sketch is then further refined through multiple iterations until the product owner is satisfied and **accepts** it.

That initial sketch for a simple Flight Booking Path example could look something like this:



Then, through multiple conversations with the product owner, developers, testers, and other stakeholders, the person assigned to formally model the Epic could come up with the following model shown at the top of the page.

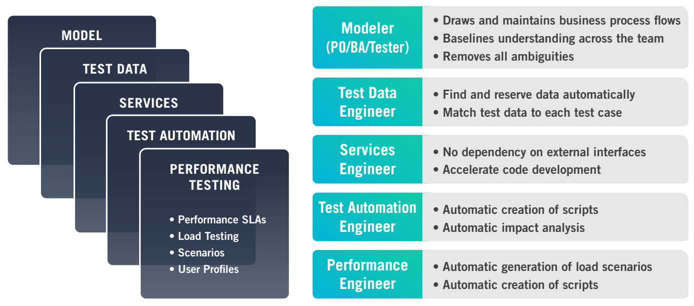
As you can see, those conversations caused a few additional process steps to be added as the model was formalized. We now know that the product owner wants the user to select the departure flight first and then select the return flight. It is also clear that before going to the passenger information step, the user must be prompted to log in. Lastly, it was clarified that the seats must be chosen only after the passenger information has been entered in the application.

Through the mere representation of the Epic in a visual model, ambiguities are removed and defects are prevented from entering the application code. Which means testing is truly "shifting left" in the lifecycle. And we're already starting to "build quality in" the application.

The visual model of the Flight Booking Path Epic becomes the foundational layer for other stakeholders in the lifecycle.

A CAD-like tool in software engineering helps us build a multilayered visual model of the requirements. These layers are tied together, and just like the CAD tools in civil engineering, the tool maintains full traceability across all layers as shown below.

LAYERED APPROACH TO CONTINUOUS DELIVERY


MODEL
TEST DATA
SERVICES
TEST AUTOMATION
PERFORMANCE TESTING
• Performance SLAs
• Load Testing
• Scenarios
• User Profiles
Modeler (PO/BA/Tester)
• Draws and maintains business process flows
• Baselines understanding across the team
• Removes all ambiguities
Test Data Engineer
• Find and reserve data automatically
• Match test data to each test case
Services Engineer
• No dependency on external interfaces
• Accelerate code development
Test Automation Engineer
• Automatic creation of scripts
• Automatic impact analysis
Performance Engineer
• Automatic generation of load scenarios
• Automatic creation of scripts

So if there is a change to any of those layers, the impact is automatically identified and communicated to the owner of each impacted layer, prompting the owner for a decision to address that impact.

From that visual model, the tool can then automatically:

1. Generate manual test cases.
2. Find, copy, mask, or synthetically generate the test data required for each test case.
3. Generate request/response pairs as well as provision virtual services for test cases to be able to run.
4. Generate test automation scripts in any language according to the test automation tools being used by the team.

So while developers must continue to invest in increasing code quality to **build things right**, the use of a CAD-like tool in software engineering not only accelerates the software lifecycle (i.e., speed), but it also ensures developers are **building the right things** (i.e., quality) from the beginning by providing unambiguous requirements to all stakeholders across the SDLC.

ALEX MARTINS has more than 18 years of experience in largescale application design, development and testing. For the last 13 years Alex has been focused on software quality engineering and testing discipline as the pillars for DevOps transformations. Going through all levels, from Tester to Practice Leader in various technology companies such as EDS, IBM, HP and Cognizant Technology Solutions, Alex built and ran several Enterprise Testing Organizations in Latin America and the US for multiple clients. Alex now works as a client advisor in the Continuous Delivery BU at CA Technologies and is also responsible for the Continuous Quality Center of Excellence. When not talking tech, you will either find Alex enjoying time with his family or on a beach somewhere surfing or kitesurfing.

It's amazing to think of the change in economic behavior over past 20 years. In 1995—the year both Amazon and eBay launched—virtually all commerce was conducted in the physical realm.

Today that's changed. In the application economy, customers' impressions are overwhelmingly shaped by their interactions with your web and mobile applications. The battleground for consumer loyalty is no longer in the physical world: it takes place on your web and mobile apps. This means that whatever products or services your company sells—and whether you realize it or not—your company is in the software business.

To compete in the application economy, your organization has to create software the way a modern factory manufactures goods. Specifically, software needs to be developed faster, at lower costs, and with high degree of quality. And this is true across all virtual touchpoints: your public-facing web and mobile apps, as well as your backend systems, are all equally critical to delivering a superior customer experience.

Agile development methodologies are a step in the right direction. DevOps takes things one step further. But the ultimate goal of any organization should be transforming into a factory capable of continuously delivering software.

Continuous delivery is not an easy task. It requires automation throughout the software development lifecycle, as a bottleneck anywhere can back up the entire assembly line. That means development, testing and release automation all must occur continuously—and concurrently. Testing is often the last hurdle to continuous delivery, and achieving continuous testing means shift-left testing practices, test automation, and testing at the API level.

CA offers an open and integrated portfolio of continuous delivery solutions that automate software delivery—from planning through production. These solutions help you accelerate the delivery of innovative, high-quality applications to drive competitive advantage and win in the application economy.

WRITTEN BY BRENDAN HAYES
DIRECTOR OF DEVOPS SOLUTIONS MARKETING, **CA TECHNOLOGIES**

PARTNER SPOTLIGHT

Continuous Delivery solutions By CA Technologies

ca technologies

Where development, testing, and release teams can work in a unique and open integrated ecosystem with proven results.

CATEGORY	NEW RELEASES	OPEN SOURCE
DevOps and Continuous Delivery	Continuous	Yes

CASE STUDY

GM Financial, the finance arm of General Motors, was striving to deliver applications and updates faster, to better serve its customers—and make it easier for them to get loans. Rapid growth within in the business and increased competition meant traditional development methods were no longer keeping pace.

GM Financial recognized the need to deliver higher quality software, faster—something that could only be achieved by automating much of the software development lifecycle.

A cohesive effort spanning dev, ops, and quality assurance did just this. With the help of CA's continuous delivery solutions, GM Financial was able to shorten a standard server deployment from several hours to a few minutes. Results like this have had a direct customer-facing impact: now loans can be processed in 1 day vs. 1-2 weeks.

Listen to their story here.

STRENGTHS

- Integrated and open, end-to-end continuous delivery ecosystem
- Develop continuously to release applications up to 20x faster
- Test continuously to gain up to a 25% reduction in testing cost and time

NOTABLE CUSTOMERS

- SunTrust
- Direct Line Group
- RaboBank
- Manheim
- AutoTrader
- Jewelers Mutual

BLOG blogs.ca.com **TWITTER** @CAInc **WEBSITE** ca.com/continuous-delivery

Continuous Delivery
Develop. Test. Release.
Deliver high-quality software, faster, with a fully automated software pipeline.
Only CA provides an integrated and open continuous delivery ecosystem.
Explore **ca.com/continuous-delivery**
ca technologies