DZone

TREND REPORT

FEBRUARY 2023

# DevOps

## CI/CD, Application Delivery, and Release Orchestration

BROUGHT TO YOU IN PARTNERSHIP WITH

SauceLabs

# Welcome Letter

**By Jason Cockerham, Community Engagement Manager at DZone**

Ah, DevOps. It's one of those things that every developer can agree has revolutionized how we build, test, and deploy software, but no one can really agree on *what* it is.

Sure, there are definitions out there, but I (and nearly every developer colleague I ask about it) have yet to find one that perfectly captures what it is or even one that most everyone can agree on.

But that's OK. We don't really need a complete definition of it because it's not really a *thing* that needs to be defined. It's more of a concept, a methodology, a mindset — if you will. What's important about DevOps is the ideas we can take from it, how we use them to make our jobs easier, and how that can change the way we think about and approach what we do.

It's almost like a really great movie or book. We've all seen or read one that had a profound impact on us, or at least made us think about something important.

Everyone else who has seen or read it probably walked away with something different from it even though the story, the characters, the ending, etc., were all exactly the same.

What really matters is how we take what was learned from it to help ourselves think about things differently.

DevOps is the same way. It's an idea, a concept, a set of practices that helps you think about how to approach building, testing, deploying, and maintaining software differently. And the overall goal is to help everyone on your team do their job better, ultimately delivering the customer (whether internal or external) a better product.

What's interesting about DevOps (also movies and books) is that as we evolve throughout our lives, these ideas and concepts, and what we learn from them, can evolve as well.

Think about when you went back and watched that movie or read that book again several years later and how it impacted you in a different way. Similarly, the DevOps world of 2023 will differ and teach us other things than the DevOps of the past.

And just like a good movie or book, there are several varying parts and aspects of it that must all work together to tell a complete story. Every part of the DevOps story is important, and they impact each of us differently.

In our 2023 *DevOps: CI/CD, Application Delivery, and Release Orchestration* Trend Report, we'll explore how the nuances of all the different aspects of DevOps — Infrastructure as Code (IaC), shift left, source code management, CI/CD, and more — are continuing to change how we build, test, and deploy code today.

So grab the popcorn and an ice-cold soda, and enjoy the DZone 2023 *DevOps* Trend Report!

Thanks for joining us,

Jason Cockerham

ORIGINAL RESEARCH

# Key Research Findings

## An Analysis of Results from DZone's 2023 DevOps and CI/CD Survey

**By G. Ryan Spain, Freelance Software Engineer, Former Engineer & Editor at DZone**

In January 2023, DZone surveyed software developers, architects, and other IT professionals in order to understand the state of DevOps and continuous integration and continuous delivery (CI/CD).

**Major research targets were:**

1. Software delivery practices and metrics
2. Continuous delivery adoption
3. Automated vs. manual deployment processes

**Methods:** We created a survey and distributed it to a global audience of software professionals. Question formats included multiple choice, free response, and ranking. Survey links were distributed via email to an opt-in subscriber list, popups on DZone.com, the DZone Core Slack workspace, and various DZone social media channels. The survey was opened on January 10th and ended on January 25th; it recorded 201 complete and partial responses.

In this report, we review some of our key research findings. Many secondary findings of interest are not included here.

## Research Target One: Software Delivery Practices and Metrics

**Motivations:**

1. The frequency with which software is deployed is a vital element in that software's success, and it is a fundamental consideration to the DevOps pipeline. But there is no "one-size-fits-all" solution to how often releases need to occur; each application has its own requirements that factor into how frequently it must be deployed. We wanted to see what kind of delivery schedules organizations were on, and whether software professionals thought these schedules were too quick, too slow, or just right.
2. DevOps is often as much about organizational structure as it is about automation and toolchains. Who is in charge of ensuring proper test coverage? Of approving pull requests? Of monitoring deployments? We tried to get a sense of who handled deployments and how that impacted the development pipeline.
3. Understanding what is succeeding and what is failing in the delivery pipeline requires metrics tracking. To help learn where organizations' priorities lay, we asked respondents what deployment metrics their organization tracked. Furthermore, we asked respondents what metrics they believed to be most important for continuous delivery (CD).

### DEPLOYMENT FREQUENCY

Finding the right release schedule for an application can be a tightrope walk; it's a difficult balance between releasing too slowly and risking falling behind competitors in terms of new features or letting bugs remain unfixed, or releasing too quickly and introducing half-finished features or adding more bugs to the application. This can become even more complex when an organization is working with multiple applications, all with their own unique release requirements. We wanted to see how often organizations were releasing their applications to production, as well as how appropriate respondents felt new feature release schedules were, so we asked the following:

*How often does your organization release to production?*

and

*Overall, your organization releases new software features: {Too quickly, Too slowly, At the optimal rate, I have no opinion}*

Results (n=192 for both):

*SEE FIGURES 1 AND 2 ON NEXT PAGE*

Figure 1

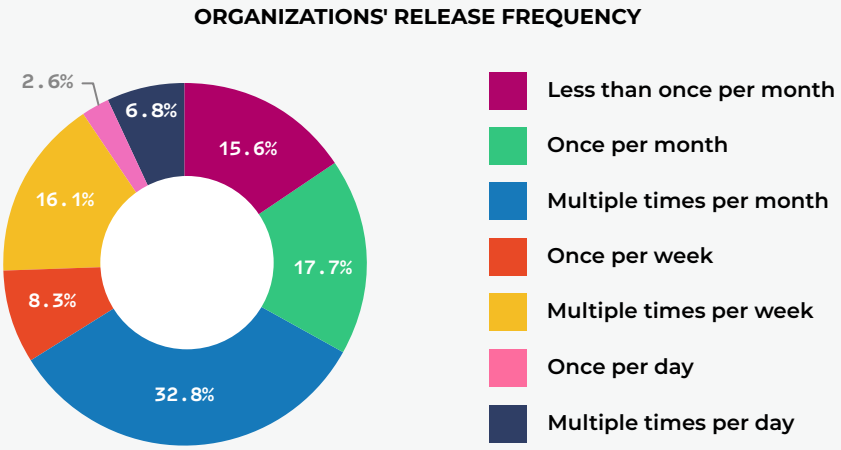
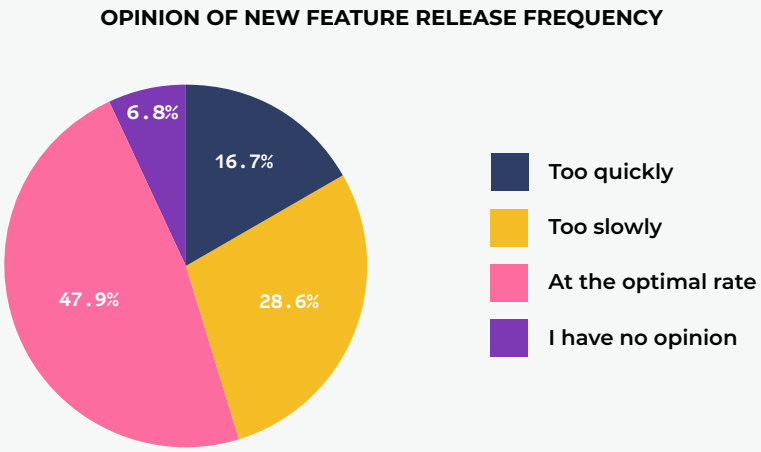



Figure 2




**Observations:**

1. One third of respondents' organizations release to production on a weekly basis or faster, one third release not quite weekly but still multiple times per month, and one third release once or less than once per month. Of course, applications and organizations will have individual needs which may dictate how often production releases should be done.

   Respondents at mid-sized organizations, on average, indicated faster release schedules than others; 61% of those at organizations with 20-49 employees and 50% of those at orgs with 50-99 employees release once a week or more; only 21% of respondents at organizations with 10,000+ employees reported releasing at this frequency.

   The type of software also had some correlation with release frequency. Respondents who said they are developing boxed software with over-the-web updates (57%) or native mobile apps (48%) were much more likely than the average respondent to release once a week or more. 48% of respondents developing embedded software said that they performed releases once a month or less.

2. Almost half (48%) of respondents thought their organization was releasing new software features at the optimal rate, 29% thought new-feature releases were coming too slowly, and 17% thought new-feature releases were coming too quickly. Comparing these responses to the production release frequency discussed earlier shows 52% of those at organizations deploying less than once per month believe that new-feature releases are too infrequent, while 32% of those at organizations deploying once per week or faster believe that these releases come too quickly.

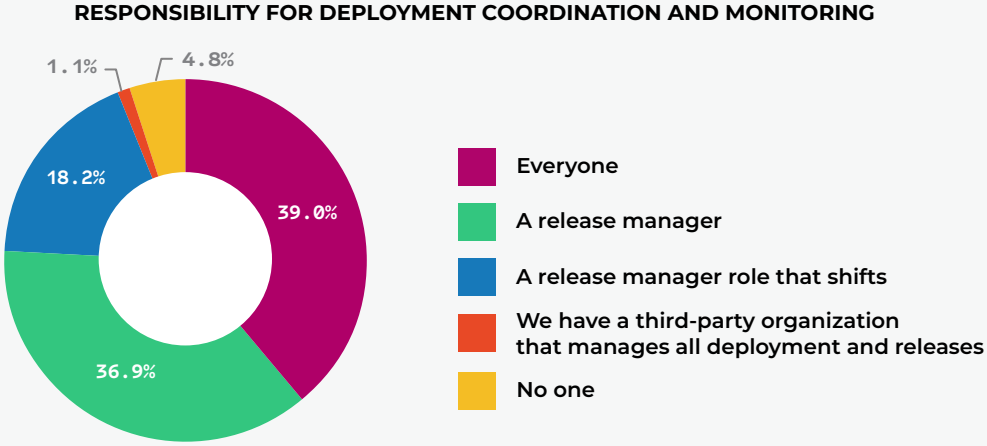
## DEPLOYMENT RESPONSIBILITY

Frequency of production deployments is just one piece of the puzzle when it comes to software release practices. Who (if anyone) in a team or organization is responsible for handling deployments can have an enormous impact on how smoothly those deployments go. In order to determine where responsibilities lay regarding deployments, we asked:

*Who helps coordinate and monitor deployments for your team?*

Results (n=187):

**Figure 3**




**Observations:**

1. Most responses were fairly evenly split between "Everyone" (39%) and "A [dedicated] release manager" (37%). 18% of respondents said their team has "A release manager role that shifts."

2. Respondents' perception of their organization's focus on dev vs. ops seems to have little to no weight on who is coordinating and monitoring deployments. Respondents who said that everyone helps with deployments, on average, thought their org's focus was 66% on dev and 34% on ops. Respondents who said a dedicated release manager ran deployments, on average, thought this focus was 63% on dev and 37% ops — a minor discrepancy within the margin of error. So having dedicated managers in charge of releases does not appear to be indicative of an organization that emphasizes a focus on operations.

3. Having a dedicated release manager did correlate with less frustration regarding environment drift. Respondents who said that everyone coordinates and monitors deployments were much more likely to say that their organizations' different environments drifted "Far enough to make every deployment nerve-wracking" (25%) than those saying they had a release manager coordinating deployments. Likewise, those with a release manager answered that their organization's environment drift was only "Far enough to cause occasional, mild annoyance" (35%) vs. respondents saying that everyone coordinates deployments.

4. Having dedicated release managers also correlated with fewer incidents and rollbacks. When asked how often deployments resulted in incidents or rollbacks, respondents who said that a release manager handled deployments mainly answered "Rarely" (44%) or "Occasionally" (41%), rather than "Almost every deployment" (15%). This shows a significant departure from respondents saying that everyone coordinates releases: Regarding the frequency of deployment rollbacks, only 32% of these respondents answered "Rarely," while 35% answered "Occasionally" and 33% answered "Almost every deployment."

## DEPLOYMENT METRICS

Tracking delivery and deployment metrics is crucial to smoothing release hiccups (or preventing full-blown disasters). But too many metrics means the most important indicators can get lost in a sea of data. To find out which metrics organizations are tracking the most, as well as which metrics respondents found most vital to continuous delivery, we asked the following:

*What deployment-related metrics does your organization track?*

and

*What do you believe are the most important metrics for continuous delivery? Please rank in order of most important (top) to least important (bottom).*

Results (n=175 and n=183, respectively):

**Figure 4**

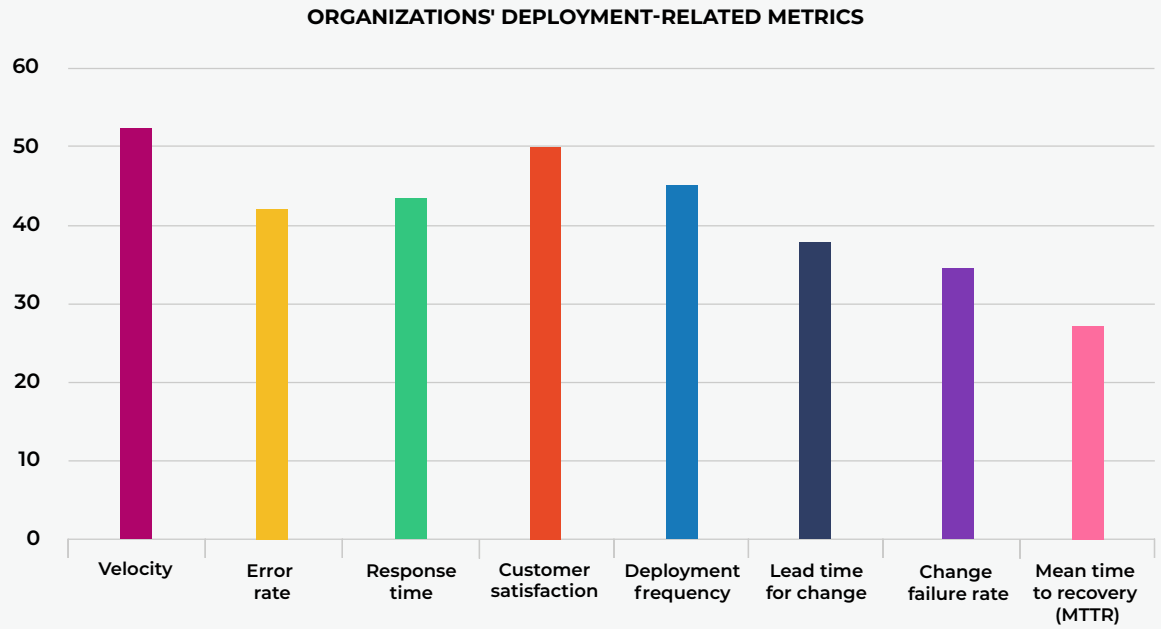



**Table 1**

**OPINION OF MOST IMPORTANT CD METRICS**

| Metric | Overall Rank | Score | n= |
|---|---|---|---|
| Deployment frequency | 1 | 1,155 | 149 |
| Production downtime during deployment | 2 | 1,029 | 136 |
| Mean time to recovery (MTTR) | 3 | 918 | 137 |
| Lead time | 4 | 871 | 126 |
| Change failure rate | 5 | 794 | 119 |
| Absolute number of bugs | 6 | 717 | 122 |
| Regression test duration | 7 | 675 | 121 |
| Mean time to discovery (MTTD) | 8 | 634 | 113 |
| Error rate | 9 | 602 | 116 |
| Error budget | 10 | 358 | 99 |

**Observations:**

1. "Velocity" (52%) and "Customer satisfaction" (50%) topped the list of deployment-related metrics tracked, ahead of "Deployment frequency" (45%), "Response time" (45%), and "Error rate" (43%), implying that speed of development and addressing customer concerns are moderately more important to organizations than how often code is put into production, how quick application transactions are, or how often deployments cause issues. This is a shift from the results of last year's survey, which reported "Deployment frequency" (55%) and "Error rate" (52%) as the most-tracked metrics, while "Velocity" (42%) was only sixth in regard to the number of responses, being surpassed by "Customer satisfaction" (50%), "Response time" (46%), and even "Lead time for change" (43%).

Given cultural changes surrounding work from home and other office policies in recent years, it's possible that this change in development velocity tracking indicates organizations have an increased concern with ensuring that tickets are being completed at an acceptable pace.

2. Respondents believed that "Deployment frequency" is the most important metric for continuous delivery, followed by "Production downtime during deployment." These were also ranked numbers one and two, respectively, in last year's survey. "Mean time to recovery" moved from fifth place last year to third this year, surpassing "Lead time" and "Change failure rate," and "Absolute number of bugs" rose from eighth place last year to sixth this year — but otherwise, rankings remained in the same order as they appeared in 2022.

   Generally, this seems to show an overall agreement by software professionals about the most important metrics needed for continuous delivery, even if those don't necessarily map exactly to the delivery metrics that organizations tend to track the most.

## Research Target Two: Continuous Delivery Adoption

**Motivations:**

1. There are many proposed or hypothetical benefits to continuous delivery, but which ones make the best arguments for software professionals to actually adopt CD? To find out, we asked respondents to rank potential continuous delivery benefits by importance.
2. Change can be quite difficult, so despite the advantages that CD can provide, there are still reasons an organization might not adopt (or fully adopt) continuous delivery practices or tools. We wanted to see what respondents thought were the main barriers to their organizations adopting continuous delivery.
3. Once CD has been, at least in part, adopted in an organization, we wanted to know if the results lived up to the hype. We asked respondents how they felt CD adoption affected the quality of their applications, and looked at other data regarding effects in the delivery pipeline, to determine how continuous delivery impacted software development and release.

### REASONS FOR ADOPTION

Adopting continuous delivery has the potential to provide a wide array of benefits: increased speed, lowered costs, less headache — the touted advantages of CD adoption cover multiple pain points in the SDLC. We tried to find out which perks of CD adoption respondents found most desirable, how those rankings compared to responses we received from last year's CI/CD survey, and how job role affected respondents' perception of value of possible CD adoption boons. We asked:

*What do you believe are the top reasons for adopting continuous delivery? Please rank in order of most important (top) to least important (bottom).*

Results (n=185):

**Table 2**

**TOP REASONS FOR ADOPTING CONTINUOUS DELIVERY**

| Reason | Overall Rank | Score | n= |
|---|---|---|---|
| Increased speed of feature delivery | 1 | 1,389 | 153 |
| Shortened development cycles | 2 | 1,206 | 138 |
| Reduced complexity of development cycles | 3 | 1,129 | 144 |
| Improved developer/team flow/productivity | 4 | 1,121 | 142 |
| Increased release frequency | 5 | 1,105 | 139 |
| Reduced overhead costs | 6 | 970 | 137 |
| Reduced number of bugs post deployment | 7 | 902 | 126 |
| Reduced time to complete QA feedback loops | 8 | 863 | 126 |
| Reduced deployment error rate | 9 | 856 | 124 |
| Reduce maintenance costs | 10 | 853 | 129 |
| Reduced mean time to discovery | 11 | 655 | 117 |
| Reduced error budget | 12 | 506 | 112 |

**Observations:**

1. Like last year, "Increased speed of feature delivery" and "Shortened development cycles" were ranked number one and two, respectively. "Reduced complexity of development cycles" jumped from fifth to third since 2022, while "Increased release frequency" switched places with it, falling from third to fifth. Also switching places in the rankings were "Reduced overhead costs" (sixth this year, ninth last year) and "Reduced deployment error rate" (ninth this year, sixth last year). Other rankings remain unchanged year over year.

2. Job role appeared to play some part in the top factors for CD adoption. Developers/engineers and developer team leads ranked "Reduced number of bugs post deployment" much higher than most, with each role ranking that reason in fourth place. "Improved developer/team flow/productivity" was much more important to DevOps team leads (second place) and developer team leads (third place) than others. And C-level executives rated reasons affecting costs — "Reduced overhead costs" (second place), "Reduced maintenance costs" (fifth place), and "Reduced error budget" (eighth place) — as much more important than other respondents, on average. While this kind of disparity could indicate that continuous delivery "has something for everyone," it could also create tension between stakeholders over how CD is implemented.

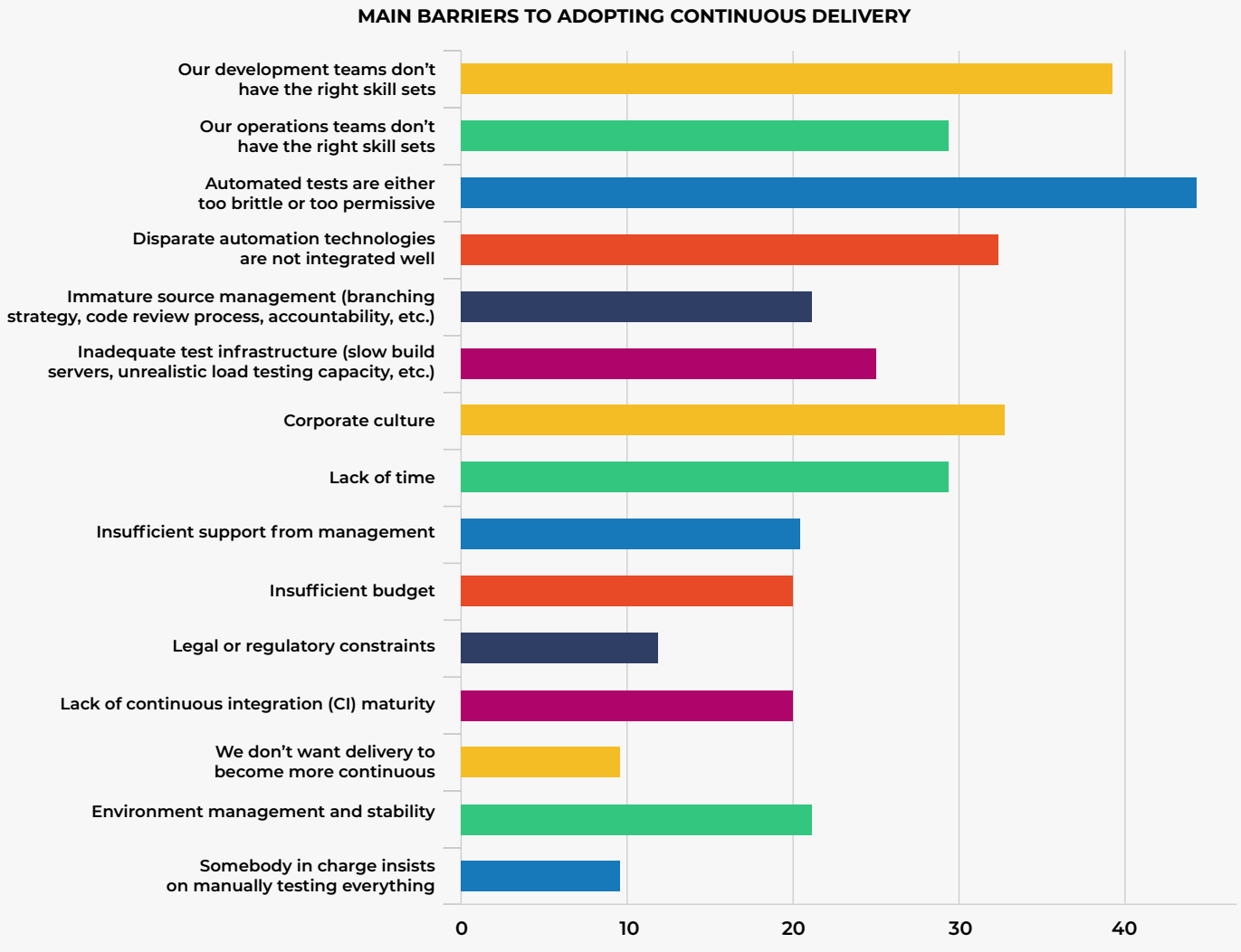
## BARRIERS TO ADOPTION

With so many reasons for adopting continuous delivery, why would any organization not buy in? Of course, nothing is ever really that simple. Barriers to adoption can be numerous and varied, and we wanted to see what software professionals thought were at the forefront, asking:

*What do you believe are your organization's main barriers to adopting continuous delivery? Select all that apply.*

Results (n=186):

**Figure 5**



**Observations:**

1. The top barriers to CD adoption comprised a mix of organizational and technological issues. 44% of respondents selected "Automated tests are either too brittle or too permissive," a technological concern, while 39% said "Our development teams don't have the right skill sets," and 33% claimed company culture was a barrier to adopting CD — two problems that stem from organizational shortcomings.

    "Disparate automation technologies are not integrated well" (32%), "Lack of time" (30%), and "Our operations teams don't have the right skill sets" (30%) were also popular responses, again suggesting multiple root causes rather than a single overarching root cause creating the majority of main adoption barriers.

2. The survey results for this year mostly remained in line with results we saw from last year's CI/CD survey, with a few notable exceptions. Respondents saying that a "Lack of continuous integration maturity" was a barrier decreased by 11% (31% in 2022 vs. 20% in 2023). On the other hand, responses of "Our development teams don't have the right skill sets" saw a 10% increase (29% in 2022 vs. 39% in 2023). There was also a 6% increase in responses that "Disparate automation technologies are not integrated well" (26% in 2022 vs. 32% in 2023). All other responses were within 3.5% of those from last year's survey.

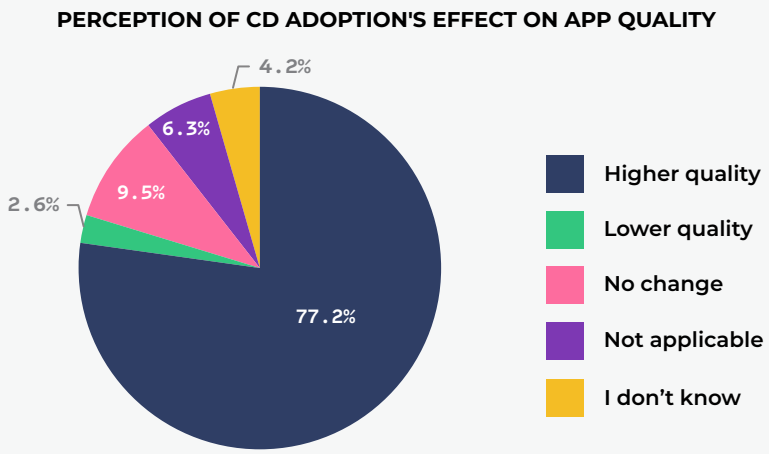
## ADOPTION RESULTS

After analyzing the benefits, overcoming barriers, implementing continuous delivery, and tracking the metrics, it's time to review the results and see if CD adoption actually improves the quality of applications. To find out how respondents felt continuous delivery affected software quality, we asked:

*Overall, adopting continuous delivery has made your applications: {Higher quality, Lower quality, No change, Not applicable, I don't know}*

Results (n=189):

**Figure 6**




**Observations:**

1. The vast majority of respondents (77%) believed that continuous delivery has made their applications higher quality, while 10% believed there was no change, and only 3% thought that CD lowered the quality of their applications. These results are not significantly different from those we saw in last year's survey, wherein 74% claimed that CD improved software quality, 9% observed no change, and 4% said quality was lowered by CD.

2. Respondents in the smallest companies (1-4 employees; n=6) saw the least change in quality from CD adoption. Subsets of respondents in companies with more employees (5+) all had between 78% and 87% saying that continuous delivery adoption made their applications higher quality, whereas only 17% of respondents in the smallest companies said the same. 67% saw no change in quality, and 17% said they did not know.*

    This would make sense, as it is much less likely that organizations this small would have the capacity to implement continuous delivery practices at the levels that larger companies could, and DevOps in general would likely be a lower priority given a much smaller divide between "dev" and "ops."

3. Respondents' perception of continuous delivery on software quality correlated with the amount of post-deployment incidents and rollbacks in their organization:
   - 36% of respondents who saw higher quality from CD adoption said their org's deployments rarely resulted in incidents and rollbacks, versus 28% who saw no quality change and 20% who saw lower quality.
   - 61% of respondents saying quality was unchanged also said they occasionally had incidents and rollbacks after deployment, compared to 38% of those who saw higher quality and 20% who saw lower quality.
   - 60% of respondents noting lower quality software from CD adoption also said that almost every deployment resulted in an incident or rollback, versus 22% seeing higher software quality and 6% seeing no change.*

   This correlation may seem obvious, but it may be an indicator that dissatisfaction with continuous delivery results regarding quality of applications may stem from improper or incomplete continuous delivery adoption. This is an area we will examine further in the next section as we analyze responses regarding deployment automation and its effects.

*Note: Given the extremely small sample sizes of some of these result subsets, the confidence level for these statistics is low, and results will need to be revisited in future research to verify or improve accuracy.

## Research Target Three: Automated vs. Manual Deployment Processes

**Motivations:**

1. "Automation" is a term thrown around a lot in the software industry, and it is one that can mean any number of things. What does automation entail, especially in the context of DevOps and continuous delivery? To find out, we asked respondents what they believed must be true for a production deployment to be "automated."
2. While semantics are generally vital to understanding, they mean very little without the support of concrete, practical application. So after examining how software professionals *defined* deployment automation, we wanted to see how it was *applied*. We looked into several stages of the deployment process — testing, building, provisioning, etc. — and sought to discover how often automation was utilized, and how often manual intervention was required.
3. Finally, to add to our analysis of the effects of continuous delivery adoption, we tried to dig deeper into the specifics by inspecting how particular aspects of automation affected software quality as well as the development process. We looked at what impacts, if any, automation had on technical debt, deployment frequency, incidents, rollbacks, and more.

### AUTOMATION REQUIREMENTS

Getting code from pre-production to production safely and effectively requires more than a few steps. And more time spent on these steps means a greater divide between the time code is ready and the time a product is in the customer's hands. Automating deployment steps is one of the primary ways to lessen that divide. We wanted to know what steps were necessary to deployment automation, so we asked:

*Check all things that must be true (i.e., join by Boolean AND) in order to automate production deployments:*

Results:

**Table 3**

**PREREQUISITES FOR AUTOMATED PRODUCTION DEPLOYMENTS**

| Prerequisite | 2023 | 2022 | Delta |
|---|---|---|---|
| Unit test coverage is > 75% | 58% | 57% | 1% |
| Code review process is robust | 57% | 75% | -18% |
| Production and pre-production environments are provisioned by the same code | 55% | 72% | -17% |
| Realistic load tests are performed on every build | 39% | 45% | -6% |
| There is an auditable approval process to promote changes from pre-production to production | 36% | 48% | -12% |
| Rollback deployments have been used successfully in production many times before | 34% | 49% | -15% |
| All whitelisted user paths are covered by automated UI tests | 30% | 42% | -12% |
| Most (a fuzzy definition of "most") user paths are covered by automated UI tests | 30% | 37% | -7% |
| Crucial features are called behind feature flags and can be turned off without a new deployment | 26% | 40% | -14% |
| Unit test coverage is > 50% | 19% | 15% | 4% |
| Unit test coverage is 100% | 17% | 25% | -8% |

**Observations:**

1. Most respondents agreed that the following factors are required in order to automate production deployments: a) "Unit test coverage must be greater than 75%" (58% of respondents), b) "Code review process must be robust" (57%), and c) "Production and pre-production environments must be provisioned by the same code" (55%).

   Other responses that were popular but not agreed on by a majority included "Realistic load tests are performed on every build" (39%), "There is an auditable approval process to promote changes from pre-production to production" (36%), and "Rollback deployments have been used successfully in production many times before" (34%). Automated UI tests, feature flags, and other unit test coverage percentage options (> 50% and 100% unit test coverage) were the least suggested of the list.

2. These results, for the most part, significantly differ from what we recorded from last year's survey. While unit test coverage more than 75% remained fairly static (+1% from 57% in 2022), the top two deployment automation requirements from last year — a robust code review process (75% in 2022) and production and pre-production environments provisioned by the same code (72% in 2022) — fell by 18% and 17%, respectively. Prerequisites also dropping dramatically from last year are successful past production rollback deployments (-15%), feature flags for crucial features (-14%), auditable approval processes from pre-prod to prod (-12%), and automated UI tests for all whitelisted user paths (-12%).

   This downward trend for almost every prerequisite listed seems to indicate a growing disagreement over what a fully automated production deployment looks like.

## AUTOMATING THE DEPLOYMENT PIPELINE

Once we discerned software professionals' opinions on deployment automation prerequisites, we wanted to see how many organizations were fully automating their production provisioning and deployments. For those whose organizations still required manual steps in the pipeline between development and production, we tried to find out which specific steps were still not automated. To gather this information, we asked the following questions:

*Do your organization's deployments to production require any manual steps?*

*Which of the following require manual intervention between dev and production? Select all that apply.**

and

*Your organization has automated provisioning and deployment: {For all pre-production environments, For some pre-production environments, For no pre-production environments}*

*Note: This question was only displayed to respondents who answered "Yes" to the previous question.

Results (n=190, n=116, and n=182, respectively):

**Figure 7**

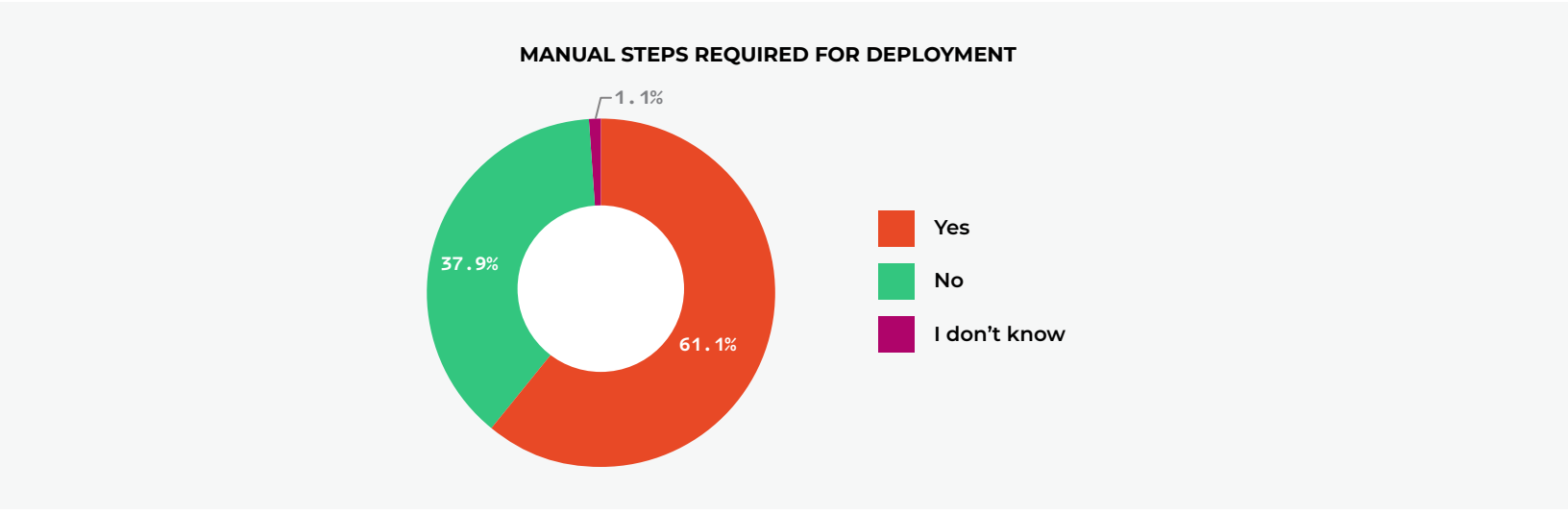


Figure 8

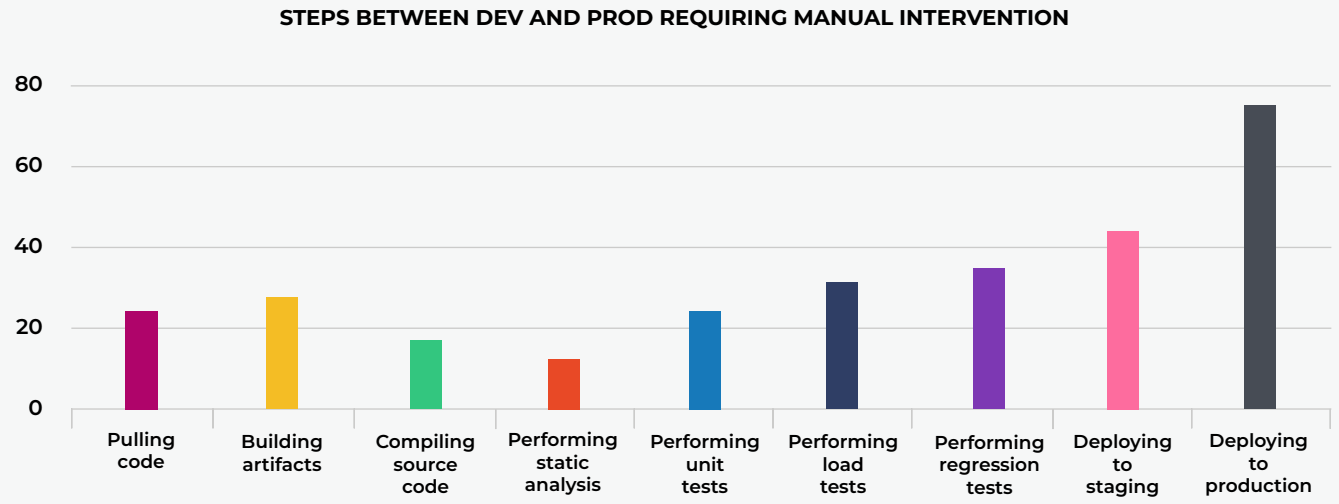
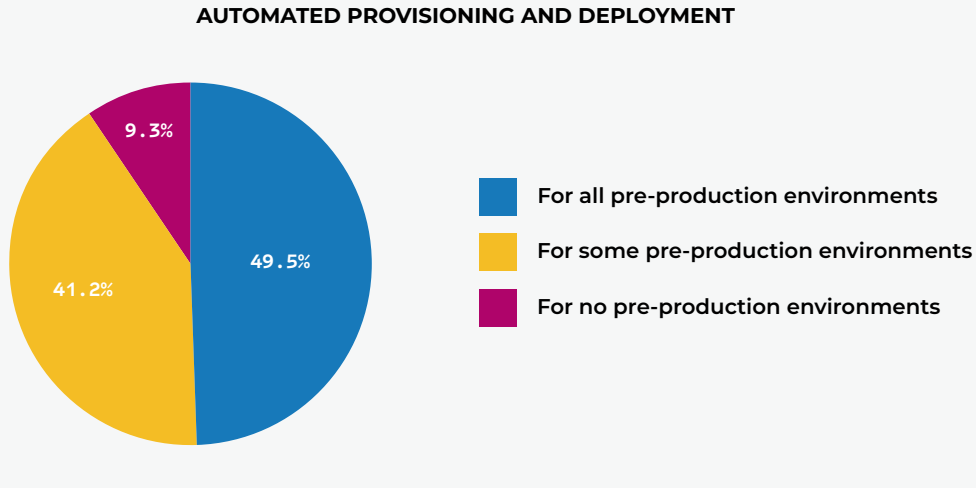



Figure 9




**Observations:**

1. 61% of respondents said their organization's production deployments require some manual steps, while 38% said their prod deployments are fully automated. This shows a moderate increase in automated deployments from last year's survey, where 67% said their production deployments required manual steps and only 29% said they were fully automated.

2. 75% of respondents with manual production deployment steps said that they manually performed the actual deployment to the production server, the most selected manual step by far. 41% said they manually deployed to staging, 34% said regression tests were manual, and 33% said load tests were manual. We do not have data from 2022 on manual production deployment steps.

3. 50% of respondents said their organization has automated provisioning and deployment for all pre-production environments, down slightly from last year's 55%. 41% said their organization only has automated provisioning and deployment for some pre-prod environments — a minor increase from 37% in 2022. And 9% of respondents said their org has no automated provisioning and deployment for pre-production environments, the same as last year.

## EFFECTS OF AUTOMATION

Lastly, we wanted to see how deployment automation affected other aspects surrounding the development cycle. We examined the automation statistics we discussed in the last section and analyzed their effects on things like technical debt, incidents/rollbacks, and deployment frequency.

Table 4

| MANUAL STEPS IN PRODUCTION DEPLOYMENTS AND PERCEPTION OF TECHNICAL DEBT | | | | |
|---|---|---|---|---|
| | Too much tech debt | Too little tech debt | The optimal amount of tech debt | I have no opinion |
| Manual steps required | 47% | 15% | 31% | 7% |
| No manual steps required | 46% | 11% | 32% | 7% |
| Two or fewer manual steps | 41% | 13% | 36% | 8% |
| Three or more manual steps | 57% | 17% | 19% | 6% |

Table 5

| FULLY AUTOMATED PRODUCTION DEPLOYMENTS AND RELEASE FREQUENCY | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Multiple times per day | Once per day | Multiple times per week | Once per week | Multiple times per month | Once per month | Less than once per month |
| Manual steps required | 6% | 2% | 16% | 5% | 31% | 21% | 20% |
| No manual steps required | 8% | 1% | 18% | 14% | 38% | 13% | 7% |

Table 6

| MANUAL STEPS IN PRODUCTION DEPLOYMENTS AND DEPLOYMENT INCIDENTS/ROLLBACKS | | | |
|---|---|---|---|
| | We rarely have to roll back due to deployment | Occasionally | Almost every deployment |
| Manual steps required | 38% | 44% | 12% |
| No manual steps required | 36% | 31% | 29% |
| Two or fewer manual steps | 42% | 38% | 21% |
| Three or more manual steps | 35% | 48% | 17% |

*Note: We did not include the two "I don't know" responses to "Do your organization's deployments to production require any manual steps?" in this analysis. Respondent subsets were divided into those who answered "Yes" and those who answered "No" to this question.

**Observations:**

1. There was no significant correlation between fully automated production deployments and respondents' perception on technical debt. However, when we looked at how many manual steps respondents said their organizations required, we found that those who said two or fewer steps (including zero steps for respondents with fully automated prod deploys) were significantly more likely to report that their organization has the optimal amount of technical debt than those with three or more manual steps (36% for two or fewer steps vs. 19% for three or more steps).

2. Respondents with fully automated production deployments had moderately quicker release frequencies than those without. The differences between these sets were insignificant for the fastest frequencies (multiple times per day, once per day, multiple times per week), but 14% of automated production deploys were once per week compared to only 5% when manual steps were required. 38% of automated deploys were multiple times per month versus 31% for deploys that weren't fully automated. Deployments requiring manual steps were more likely to release once per month (21%) or less than once per month (20%) when compared to fully automated deployments (13% and 7%, respectively).

3. Interestingly, respondents who said that their organization's production deployments did not require any manual steps were much more likely to report that "Almost every deployment" resulted in an incident or rollback (29%) compared to those with manual prod deploy steps, with the latter group being more likely to say deployments "Occasionally" resulted in incidents or rollbacks (44% manual deployments vs. 31% automated).

Grouping by the number of manual steps required, those with two or fewer manual steps only had a minor increase in "Almost every deployment" rollback responses — 21% versus 17% when three or more manual steps were needed — but had moderately more responses saying, "We rarely have to roll back due to deployment" (42% for two or fewer manual steps vs. 35% for three or more).

4. We believe that these discrepancies between the observations we found in the results above versus our expectations — that fully automated production deployments would produce much more optimal technical debt, much faster delivery times, and many fewer incidents and rollbacks — may be due, in part, to the lack of agreement on prerequisites to fully automated deployments that we discussed in a previous section.

## Future Research

Our analysis here only touched the surface of the available data, and we will look to refine and expand our DevOps and CI/CD survey as we produce further Trend Reports. Some of the topics we didn't get to in this report, but were incorporated in our survey, include:

- Specific methods of organizations' application infrastructure provisioning
- Manual vs. automated testing
- TDD, BDD, and other approaches to software development and design
- Rankings of programming paradigms and SOLID principles
- Toolchain orchestration solutions

Please contact publications@dzone.com if you would like to discuss any of our findings or supplementary data.


**G. Ryan Spain, Freelance Software Engineer, Former Engineer & Editor at DZone**
@grspain on DZone | @grspain on GitHub and GitLab | gryanspain.com

G. Ryan Spain lives on a beautiful two-acre farm in McCalla, Alabama with his lovely wife and adorable dog. He is a polyglot software engineer with an MFA in poetry; a die-hard Emacs fan and Linux user; a lover of The Legend of Zelda; a journeyman data scientist; and a home cooking enthusiast. When he isn't programming, he can often be found playing Minecraft with a glass of red wine or a cold beer.

# What Would You Do With More Device Coverage?

**Wow your customers**
Catch bugs early to release quality apps at speed

**Make life easier for your mobile teams**
Test anywhere, anytime, on any device

**Reduce operational costs & headaches**
Spend less (money), stress less (maintenance), scale more (mobile tests)

**Kill the security & compliance game**
Manage device access, meet security standards, safeguard your data

Automate testing on the most extensive range of devices and OSes with Sauce Labs.

TELL ME MORE → saucelabs.com/cloud-based-real-device-testing

SauceLabs

PARTNER CASE STUDY

# Case Study: Nederlandse Spoorwegen

Building and Releasing a New App in Just Three Months With Sauce Labs

## Challenge

Nederlandse Spoorwegen (NS), the Netherlands' national railway system, plays an essential role in millions of daily lives by providing train, bike, and car transportation options via their app or website.

NS wanted to better serve customers by providing a more streamlined, inclusive, and personalized experience. They were planning a move from waterfall methodology to agile methodology, which required its testers to significantly scale up test volume. To accomplish this, NS needed to increase test coverage overall. Unfortunately, testing was already a painfully slow and manual process, with testers relying on their own infrastructure and phones. These practices left testing incomplete and inconsistent, leading to negative customer experiences.

Another problem that NS struggled with was the onboarding of staff. Without a centralized system, onboarding and knowledge sharing was difficult.

## Solution

In a thorough evaluation of testing solutions, the NS testers were impressed with Sauce Labs, citing the solution's intuitiveness and dashboard as key differentiators. "The ease of working stood out," van Veenendaal says. "For example, working with the cross-browser tests really makes a difference." Ultimately, the company chose Sauce Labs because of its wide, reliable testing coverage, local data center, and outstanding customer support.

## Results

NS is extremely satisfied with Sauce Labs. It now uses a local data center, which helps the company avoid security issues while improving performance time by 50 percent. "Now, testing is faster and more thorough," van Veenendaal says. "If we have any issues, we find them quickly. It's simply more efficient to work with one dashboard and one tool." The tribal knowledge problems and painful experiences of onboarding are also a thing of the past.

"The developers use Sauce Labs from their local environments and are very happy with all the possibilities. They can test and check all the things they want to know, seeing in real time if a new feature is broken," van Veenendaal says.

Perhaps most important of all, NS can provide customers with a better experience. "We're able to offer our customers a more personalized experience," van Veenendaal explains. "It feels like a trusted environment, where they feel comfortable sharing personal and financial information."

**COMPANY**
Nederlandse Spoorwegen

**COMPANY SIZE**
30,000+ employees

**INDUSTRY**
Travel

**PRODUCTS USED**
Sauce Cross-Browser, Sauce Mobile

**PRIMARY OUTCOME**
NS built and released a new app in just three months thanks to Sauce Labs' commitment to automation, an easy onboarding process, and having the most coverage of devices and OS.

*"Perhaps most important of all, Nederlandse Spoorwegen can provide customers with a better experience. We're able to offer our customers a more personalized experience."*

— **Serge van Veenendaal**,
Non-Functional Test Specialist,
Nederlandse Spoorwegen

CREATED IN PARTNERSHIP WITH

SauceLabs

CONTRIBUTOR INSIGHTS

# A Beginner's Guide to Infrastructure as Code

## How IaC Works, Its Benefits, and Common Challenges

**By Marija Naumovska, Technical Content Writer at Microtica**

Infrastructure as Code (IaC) is the practice of provisioning and managing infrastructure using code and software development techniques. The main idea behind IaC is to eliminate the need for manual infrastructure provisioning and configuration of resources such as servers, load balancers, or databases with every deployment. As infrastructure is now an integral part of the overall software development process and is becoming more tightly coupled with application delivery, it's important that we make it easier to deliver infrastructure changes.

Using code to define and manage infrastructure and its configuration enables you to employ techniques like version control, testing, and automated deployments. This makes it easier to prevent all sorts of application issues, from performance bottlenecks to functionality failures.

This article will explain how IaC works, highlighting both approaches, as well as the benefits and challenges of delivering infrastructure as code within a DevOps environment.

### How Does IaC Work?

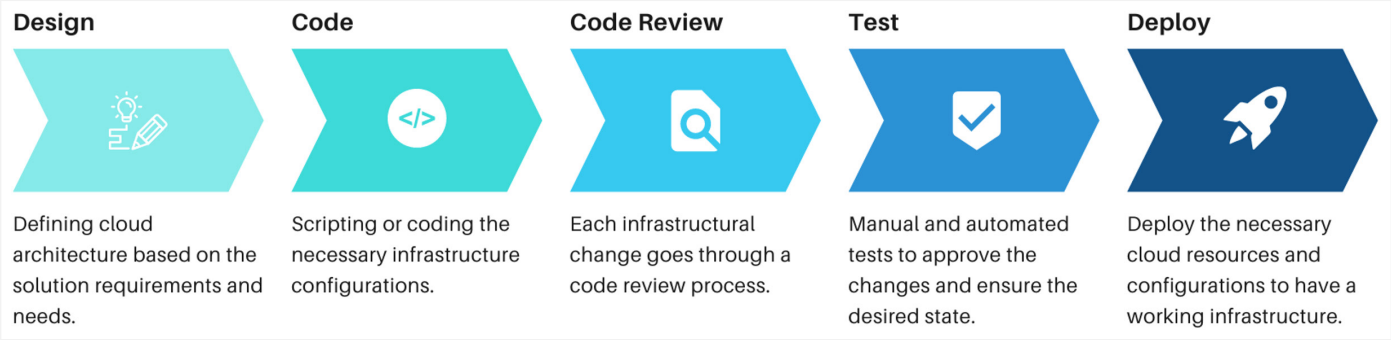
Historically, infrastructure management was mostly a manual process done by specialized system administrators. You needed to manually create virtual machines (VMs), manage their software updates, and configure their settings. This became highly expensive and time-consuming, especially with the rapid pace of modern software development.

IaC evolved as a solution for scalable infrastructure management. It allows you to codify the infrastructure to then be able to create standardized, reusable, and sharable configurations. IaC also allows you to define infrastructure configurations in the form of a code file. For example, Figure 1 demonstrates how you'd define the creation of an S3 bucket in AWS, using CloudFormation.

```
Resources:
  S3Bucket:
    Type: 'AWS::S3::BUCKET'
    DeletionPolicy: Retain
    Properties:
BucketName: DOC-EXAMPLE-BUCKET
```

As you define your Infrastructure as Code, you can implement the same practices that you use with application development code, like versioning, code review, and automated tests.

**Figure 1: The IaC workflow**



To implement IaC, you can use a variety of tools and technologies, like:

- Configuration management tools that make sure the infrastructure is in the desired state that you previously defined, like Ansible, Chef, or Puppet.
- Provisioning tools (e.g., CloudFormation templates) that allow you to define cloud resources in the form of a JSON or YAML file, and provision that infrastructure on a cloud platform.
- Containerization tools (e.g., Docker, Kubernetes) used to package applications and their dependencies into containers that can be run on any infrastructure.

## Approaches to IaC

There are two different Infrastructure as Code approaches: imperative (procedural) IaC and declarative (functional) IaC.

With the **imperative** method, the developer specifies the exact steps that the IaC needs to follow to create the configuration. The user commands the automation completely, which makes this method convenient for more specific use cases where full control is needed.

The main advantage of the imperative method is that it allows you to automate almost every detail of the infrastructure configuration. This also means that you need a higher level of expertise to implement this type of automation as it's mostly done by executing scripts directly on the system.

Here is an example of an imperative way to create an S3 bucket using the AWS CLI:

```
aws s3api create-bucket --bucket my-new-bucket --region eu-central-1
```

When you run this command, the AWS CLI will create a new Amazon S3 bucket with the name `my-new-bucket` in the `eu-central-1` region.

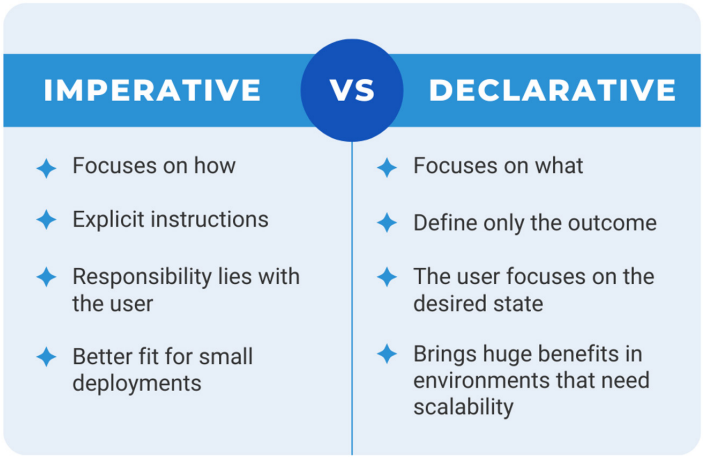
With the **declarative** method, the developer specifies the desired outcome without providing the exact steps needed to achieve that state. The user describes how they want the infrastructure to look through a declarative language like JSON or YAML. This method helps with standardization, change management, and cloud delivery. Features can be released faster and with a significantly decreased risk of human error.

Here's a simple example of a declarative Infrastructure as Code using AWS CloudFormation:

```
{
“Resources”: {
“EC2Instance”: {
“Type”: “AWS::EC2::Instance”,
“Properties”: {
“InstanceType”: “t2.micro”,
“ImageId”: “ami-0c94855bac71e”,
“KeyName”: “my-key”
                    }
             }
}
}
```

This script tells CloudFormation to create an EC2 instance of type `t2.micro`, and the CloudFormation will take care of all the required steps to achieve that state with the specific properties you defined.

Figure 2: Overview of IaC approaches




## The Benefits and Challenges of IaC

Infrastructure as Code is one of the key practices in DevOps and provides many benefits that save time and money, as well as reduce risk. But as with any tool, solution, or practice that you adopt in the organization, it is important to weigh the challenges that one may face when implementing IaC methods.

Table 1: IaC benefits vs. challenges — factors to consider

| Factor | Benefits | Challenges |
|---|---|---|
| Codification | Codifying infrastructure helps developers ensure that the infrastructure is configured well without any unintended changes whenever it is provisioned. | Infrastructure as Code can reduce visibility into the infrastructure if it is not implemented or managed properly. Improved visibility can be achieved by making sure that the code is well-documented, easily accessible, standardized, simple, and properly tested. |
| Configuration drift | IaC is idempotent, meaning that if a change happens that's not in sync with your defined IaC pipeline, the correction to the desired state occurs automatically. | Avoiding manual changes directly in the console reduces challenges with configuration drift in relation to IaC implementation. |
| Version control | Integrating with the version control system your team uses helps create trackable and auditable infrastructure changes and facilitates easy rollbacks. | As the size and complexity of an organization's infrastructure grow, it can become more difficult to manage using IaC. |
| Testing and validation | Infrastructure changes can be verified and tested as part of the delivery pipeline through common CI/CD practices like code reviews. | Performing code reviews to ensure infrastructure consistency is not always enough — there are usually a variety of testing options specific to a use case. |
| Cost | Automating time-consuming tasks like infrastructure configuration with IaC helps minimize costs and reallocate resources to more critical assignments. | Extra costs can easily ramp up if everyone is able to create cloud resources and spin up new environments. This usually happens during the development and testing phases, where developers create resources that can be forgotten after some time. To prevent this, it's a good idea to implement billing limits and alarms. |
| Speed | There may be a higher initial investment of time and effort to automate infrastructure delivery, but automating IaC brings faster and simpler procedures in the long run. | For organizations that run simple workloads, the process of automating and managing IaC can become more burdensome than beneficial. |
| Error handling | Automating with IaC eliminates human-made infrastructure errors and reduces misconfiguration errors by providing detailed reports and logs of how the infrastructure works. | In complex infrastructure setups, it can be extremely challenging to debug and troubleshoot the infrastructure, especially when issues arise in production environments. |
| Security | You can define and execute automated security tests as part of the delivery pipeline. Security experts can review the infrastructure changes to make sure they comply with the standards, and security policies can be codified and implemented as guardrails before deploying to the cloud. | As IaC is a much more dynamic provisioning practice that can be used to optimize infrastructure management, it can be misused almost as easily. IaC can make it easier to unintentionally introduce security vulnerabilities, such as hard-coded credentials or misconfigured permissions. |

## Conclusion

With IaC, systems can be easily reproduced and reused; the processes are consistent. As DevOps culture becomes more pervasive, maintaining a strategic advantage through IaC will likely become an increasingly important goal. If you work at an organization that aims to implement Infrastructure as Code within their existing processes, it can be helpful to understand the benefits and challenges that your team might encounter. The trick is to walk a fine line between understanding your business' infrastructure needs and recognizing the potential for improvement that IaC offers.

To ensure that IaC is implemented properly, it's important to start small. You want to gradually increase the complexity of the tasks, avoid over-complicating the codebase, continuously monitor the IaC implementation to identify areas for improvement and optimization, and continue to educate yourself about the different tools, frameworks, and best practices in IaC.

Check out these additional resources to continue learning about IaC:

- *Getting Started With IaC*, DZone Refcard
- *IaC Security: Core DevOps Practices to Secure Your Infrastructure as Code*, DZone Refcard
- "Infrastructure-as-Code: 6 Best Practices for Securing Applications" by Jim Armstrong
- "5 Best Practices for Infrastructure as Code" by Marija Naumovska
- "Best Infrastructure as Code Tools (IaC): The Top 11 for 2023" by Florian Pialoux

**Marija Naumovska, Co-Founder & Technical Writer at Microtica**
@marulka on DZone | @marulka on Twitter

As a co-founder of Microtica, Marija helps developers deploy their applications on the cloud in minutes. She's a software engineer with more than nine years of experience and now works as a product person and technical writer full time. She writes about cloud, DevOps, GitOps, and Kubernetes topics.

CONTRIBUTOR INSIGHTS

# Shift-Left: A Developer's Pipe(line) Dream?

## The Traditional SDLC Is Broken and Long Overdue for a "Shift" in Direction

**By John Vester, Lead Software Engineer at Marqeta**

The software development life cycle (SLDC) is broken. In fact, the implementation steps were flawed before the first project ever utilized Winston Royce's implementation steps.

**Figure 1: Winston Royce software implementation steps (Waterfall Model)**


Dr. Winston Royce stated "the implementation described above is risky and invites failure" in the same lecture notes that presented this very illustration back in 1970. Unfortunately, that same flaw has carried over into the iterative development frameworks (like Agile) too.

What is broken centers around **when** and **how** the quality validation aspect is handled. In Figure 1, this work was assumed to be handled in the testing phase of the cycle. The quality team basically sat idle (from a project perspective) until all the prior work was completed. Then, the same team had a mountain of source code which had to be tested and validated — putting the initiative in a position where it is difficult to succeed without any issues.

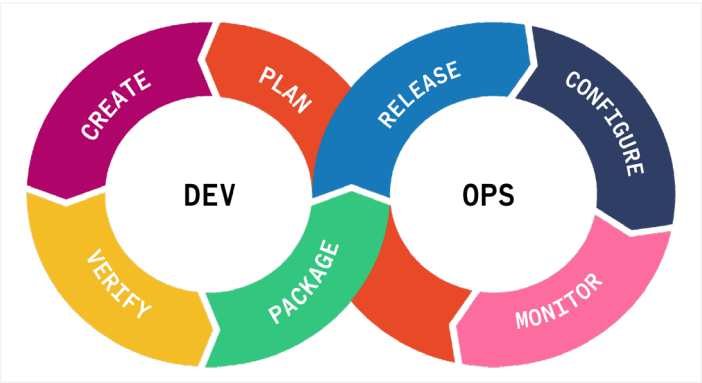
If it were possible to extract a top 10 list of lessons learned over the past 50+ years in software development, I am certain that the consequence of placing quality validation late in the development lifecycle is at the top of the list. This unfortunate circumstance has had a drastic impact on customer satisfaction — not to mention livelihood of products, services, or entire business entities.

Clearly, we are far past due for a "shift" in direction.

### How Shift Left Is a Game Changer

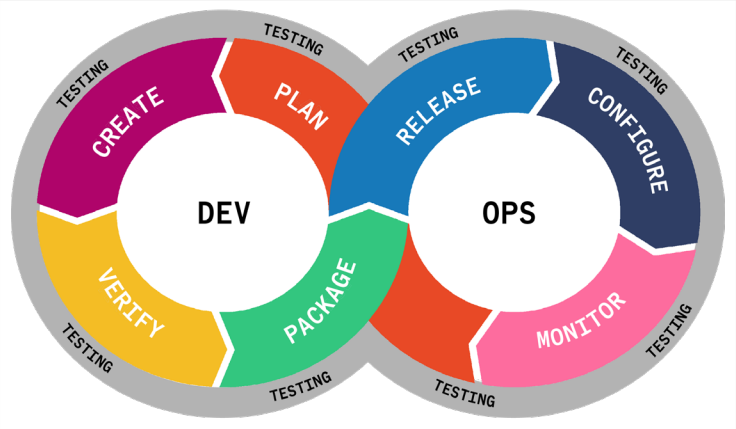
Shift left is an approach that moves — and really scatters — the testing phase of the development lifecycle to earlier in the process. In fact, the "test early and often" approach was first mentioned by Larry Smith back in a 2001 Dr. Dobb's Journal post. To demonstrate how shift left is a game changer, consider the basic DevOps toolchain shown in Figure 2, which has become quite popular today:

**Figure 2: DevOps toolchain**



Shift left adoption introduces a testing effort at every phase of the lifecycle, as is demonstrated in Figure 3:

**Figure 3: DevOps toolchain with shift-left adoption**




Employing a shift-left approach redistributes when the quality aspects are introduced into the lifecycle:

- **Plan** – Expose fundamental flaws sooner by leveraging and validating specifications like OpenAPI.
- **Create** – Establish unit and integration tests *before* the first PR is created.
- **Verify** – Include regression and performance/load tests *before* the first consumer access is granted.
- **Package and so on** – Assure the CI/CD pipelines perform automated test execution as part of the lifecycle. This includes end-to-end and sanity tests designed to validate changes introduced in the latter phases of the flow.

As a result, shift left yields the following benefits:

- Defects in requirements, architecture, and design are caught near **inception** — saving unnecessary work.
- Difficulty in trying to **comprehend** and **organize** a wide scope of use cases to validate is avoided by the "test early and often" approach inherent within shift left.
- Understand **performance** expectations and realities, which could potentially drive design changes.
- Determine **breaking points** in the solution — before the first production request is made.
- Avoid having to make design updates **late** in the lifecycle, which is often associated with unplanned costs.
- Higher staff levels required to fully test at the end of development are avoided by dedicating smaller staff levels to **participate throughout** the development lifecycle.

## Shift Left in Action

The "test early and often" approach often associated with shift left has the direct result of better quality for a given solution. Here are a few key points to expect when shift left is put into action.

### NEW CAREER PATH FOR QUALITY ENGINEERS

One might expect the pure focus of shift left to be on quality engineers assigned to the initiative. That is not really the case. Instead, this change becomes the role of every engineer associated with the project. In fact, more organizations today are merging tasks commonly associated with quality engineers and assigning them to software engineers or DevOps engineers. Forward-looking quality engineers should evaluate which of these engineering roles best suits their short- and long-term goals when thinking about their role in shift left-driven organizations.

### TEST COVERAGE

In a "test early and often" approach, actual tests themselves are vital to the process. The important aspect is that the tests are functional — meaning they are written in some type of program code and not executed manually by a human. Some examples of functional tests that can adhere to shift left compliance are noted below:

- **API** tests created as a result of an API-first design approach or similar design pattern
- **Unit** tests introduced to validate isolated and focused segments of code
- **Integration** tests added to confirm interactions across different components or services

- **Regression** tests written to fully exercise the solution and catch any missed integrations
- **Performance** tests designed to determine how the solution will perform and establish benchmarks

The more test coverage that exists, the better the solution will be. As a rule of thumb, API and unit tests should strive for 100% code coverage and the remaining tests should strive for 90% coverage since reaching full coverage is often not worth the additional costs required.

### PIPELINE CONFIGURATION

Adoption of the shift-left approach can require updates to the pipeline configuration as well. In addition to making sure the tests established above are part of the pipeline, **sanity** and **end-to-end** functional tests should be included. These sanity tests are often short-running tests to validate that each entry point into the solution is functioning as expected. The end-to-end functional tests are expected to handle the behavioral aspects of the application — validating that all of the core use cases of the solution are being exercised and completed within the expected benchmarks.

### RELEASE CONFIDENCE

The end-result of shift left in action is a high degree of confidence that the release will be delivered successfully — minimizing the potential for unexpected bugs or missed requirements to be caught by the customer. This is a stark contrast to prior philosophies that grouped all the testing at the end of the lifecycle — with a hope that everything was considered, tested, and validated.

## Too Idealistic?

Like any lifecycle or framework, there are some key challenges that must be addressed and accepted before shift left is adopted into an organization to avoid a "too idealistic" conclusion.

### MUST BE A TOP-DOWN DECISION

Shift left must be a top-down decision because the core philosophy changes that are being introduced extend beyond the team participating in the initiative. What this means is that managers of quality engineering staff will find themselves dedicating individuals to projects on a long-term basis instead of handling all the validation efforts after development has been completed. From a bigger-picture perspective, those quality engineers are likely going to find themselves adapting their role to become a software or DevOps engineer — reporting to a different management structure.

### LEVEL-SET EXPECTATIONS

The shift left approach will also require expectations to be established and clarified early on. This builds upon the last challenge because each step of the lifecycle will likely require more time to complete. However, the overall time to complete the project should remain the same as the projects that achieve full and successful testing at the end of the lifecycle.

It is vital to remember that defects found by shift left adoption will have less of an impact to resolve. This is because the testing is completed within a given phase, reducing the potential to address additional concerns that must be resolved in prior phases of the lifecycle.

### SACRIFICING QUALITY IS NO LONGER AN OPTION

A risky approach often employed for initiatives that are pressured to meet an imposed deadline is to shorten the amount of time reserved for the quality and validation phase of the lifecycle. When shift left is adopted, this is no longer a valid option since a dedicated testing phase no longer exists. While this approach is never something that should be considered, aggressive decision makers often would roll the dice and sacrifice the amount of time given to quality engineers to validate the initiative, hoping for a positive result.

## Conclusion

Throughout my life, I have always been a fan of the blockbuster movies.  You know those movies made with a large budget and a cast that includes a few popular actors. I was thinking about the 1992 Disney movie *Aladdin*, which builds upon a Middle-Eastern folktale and features the comic genius of the late Robin Williams. I remember there being a scene where the genie gives Aladdin some information about the magical lamp. Immediately, the inspired main character races off before the genie can provide everything Aladdin really needs to know. It turns out to be a costly mistake.

I feel like Dr. Winston Royce was the genie while the rest of us raced through the lifecycle without a desire to hear the rest of the story, like Aladdin. Decades and significant cost/time expenditures later, a new mindset has finally emerged which builds upon Royce's original thoughts.

Regardless of your team's solution destiny, shift left should be strongly considered because this is something we should have been doing all along. Taking this approach provides the positive side effect of transforming dedicated quality engineers to software engineers who can participate at every step of the lifecycle being utilized. The biggest benefit is identifying defects earlier in the lifecycle, which should always translate into a significant cost savings when compared to finding and fixing defects later in the lifecycle.

To succeed, implementing shift left must be a top-down decision — driven by an organization-wide change in mindset and supported by everyone. Not ever should a reduction in quality or performance testing be a consideration to shorten the time required to release a new feature, service, or framework.

Have a really great day!

References:


- "Shift Left," Devopedia
- "Successfully Implementing TDD/BDD to Enable Shift-Left Testing Approach" by Pavan Rayaprolu
- "Managing the Development of Large Software Systems" by Dr. Winston W. Royce
- "Shift-Left Testing" by Larry Smith



**John Vester, Lead Software Engineer at Marqeta, Freelance Writer, & DZone Core Member**

@johnjvester on DZone | LinkedIn, Twitter, GitLab, GitHub, dockerhub


IT professional with 30+ years expertise in app design and architecture, feature development, and project and team management. Currently focusing on enterprise architecture/app design utilizing object-oriented programming languages and frameworks. Prior expertise in building (Spring Boot) Java-based APIs against React and Angular client frameworks, CRM design, customization, and integration with Salesforce. Additional experience using C# (.NET Framework) and J2EE (including Spring MVC, JBoss Seam, Struts Tiles, JBoss Hibernate, Spring JDBC).

CONTRIBUTOR INSIGHTS

# How to Build an Effective CI/CD Pipeline

Practical Steps for Creating Pipelines That Accelerate Deployments

**By Justin Albano, Software Engineer at IBM**

Continuous integration/continuous delivery (CI/CD) pipelines have become an indispensable part of releasing software, but their purpose can often be misunderstood. In many cases, CI/CD pipelines are treated as the antidote to release woes, but in actuality, they are only as effective as the underlying release process that they represent. In this article, we will take a look at a few simple steps to create an effective CI/CD pipeline, including how to capture and streamline an existing release process, and how to transform that process into a lean pipeline.

## Capturing the Release Process

A CI/CD pipeline is not a magic solution to all of our release bottlenecks, and it will provide minimal improvement if the underlying release process is faulty. For software, the **release process** is the set of steps that a team uses to get code from source code files into a packaged product that can be delivered to a customer. The process will reflect the **business needs** of each product and the team creating the product.

While the specifics of a release process will vary — some may require certain security checks while others may need approvals from third parties — nearly all software release processes share a common purpose to:

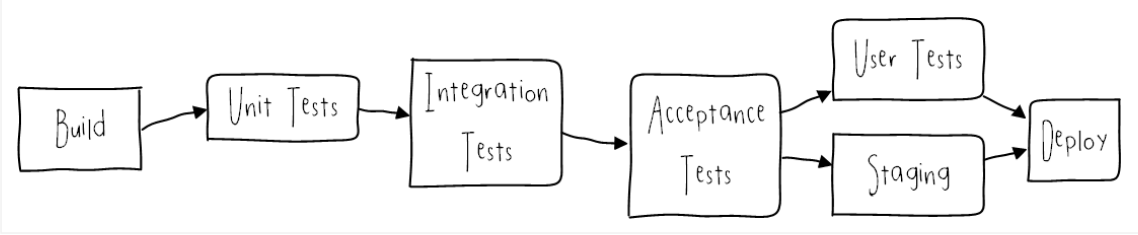
- Build and package the source code into a set of artifacts
- Test the artifacts with various levels of scrutiny, including unit, integration, and end-to-end (E2E) tests
- Test the critical workflows of the product from an end user's perspective
- Deploy the artifacts into a production-like environment to smoke test the deployment

Every team that delivers a product to a customer has some release process. This process can vary from "send the artifacts to Jim in an email so he can test them" to very rigid and formalized processes where teams or managers must sign off on the completion of each step in the process.

### PUTTING IT DOWN ON PAPER

Despite this variation, the first, most-critical step in developing an effective CI/CD pipeline is capturing the release process. The simplest way to do this is by drawing a set of boxes to capture the steps in the release process and drawing arrows from one step to another to show how the completion of one step initiates the start of another. This drawing does not have to be overly formal; it can be done on a sheet of paper, so long as the currently practiced process is captured. Figure 1 illustrates a simple release process that is common for many products:

**Figure 1: A basic release process**




*Capturing the steps of the current release process is the first step to creating a pipeline*

### SPEAKING THE SAME LANGUAGE

Once the current release process has been captured, the next step is to formalize the process. When speaking about a release process, and eventually a CI/CD pipeline, it is important to use a common vernacular or domain language.

For pipelines, the basic lexicon is:

- **Step** – A single action, such as *Build*, *Unit Tests*, or *Staging*, in the release process (i.e., the boxes).
- **Stage** – A single phase in the release process, containing one or more steps. Generally, stages can be thought of as the sequential columns in a pipeline. For example, *Build* is contained in the first stage, *Unit Test* in the second stage, and *User Tests* and *Staging* in the fifth stage. When there is only one step in a stage, the term step and stage are often used synonymously.
- **Pipeline** – A set of ordered steps.
- **Trigger** – An event, such as a check-in or commit, that starts a single execution of the pipeline.
- **Gate** – A manual step that must be completed before all subsequent steps may start. For example, a team or manager may need to sign off on the completion of testing before the product can be deployed.

A CI/CD pipeline is simply an automated implementation of a formalized release process. Therefore, if we wish to create an effective CI/CD pipeline, it's essential that we optimize our release process first.

## Optimizing the Release Process

Since our CI/CD pipelines are a reflection of our release process, one of the best ways to create an effective pipeline is to optimize the release process itself before deriving a pipeline from it. There are three critical optimizations that we can make to a release process that pay dividends toward an effective pipeline:

1. **Streamline the process** – We should minimize any bottlenecks or artificial steps that slow down our release process.
   - Remove any unnecessary steps.
   - Minimize the number of steps while fulfilling business needs.
   - Simplify any complex steps.
   - Remove or distribute steps that require a single point-of-contact.
   - Accelerate long-running steps and run them in parallel with other steps.
2. **Automate everything** – The ideal release process has no manual steps. While this is not always possible, we should automate every step possible.
   - Consider tools and frameworks such as JUnit, Cucumber, Selenium, Docker, and Kubernetes.
   - Capture the process for running each step in a script — i.e., running the build should be as easy as executing `build.sh`. This ensures there are no magic commands and allows us to run each step on-demand when troubleshooting or replicating the release process.
   - Create portable scripts that can be run anywhere the release process is run. Do not use commands that will only work on specific, special-purpose environments.
   - Version control the scripts, preferably in the same repository as the source code.
3. **Shorten the release cycle** – We should release our product as often as possible. Even if the end deliverable is not shipped to the customer or user (e.g., we build every day but only release the product to the customer once a week), we should be frequently running our release process. If we currently execute the release process once a day, we should strive to complete it on every commit.

Optimizing the release process ensures that we are building our CI/CD pipeline from a lean and efficient foundation. Any bloat in the release process will be reflected in our pipeline. Optimizing our release process will be iterative and will take continuous effort to ensure that we maintain a lean release process as more steps are added and existing steps become larger and more comprehensive.

## Building the Pipeline

Once we have an optimized release process, we can implement our pipeline. There are three important pieces of advice that we should follow in order to create an effective CI/CD pipeline:

1. **Don't follow fads** – There are countless gimmicks and fads fighting for our attention, but it is our professional responsibility to select our tools and technologies based on what is the most effective for our needs. Ubiquity and popularity do not guarantee effectiveness. Currently, options for CI/CD pipeline tools include GitHub Actions, GitLab CI/CD, and Jenkins. This is not a comprehensive list, but it does provide a stable starting point.
2. **Maintain simplicity** – Each step should ideally run one script with no hard-coded commands in the pipeline configuration. The pipeline configuration should be thought of as glue and should contain as little logic as possible.

For example, an ideal GitLab CI/CD configuration (`.gitlab-ci.yml`) for the release process in Figure 1 would resemble:

```
build:
  stage: building
  script:
    - /bin/bash build.sh

unit-tests:
  stage: unit-testing
  script:
    - /bin/bash run-unit-tests.sh

integration-tests:
  stage: integration-testing
  script:
    - /bin/bash run-integration-tests.sh
...

deploy:
  stage: deployment
  script:
    - /bin/bash deploy.sh --env production:443 --key ${SOME_KEY}
```

This ideal is not always possible, but this should be the goal we strive toward.

3. **Gather feedback** – Our pipelines should not only produce artifacts, but it should also produce reports. These reports should include:
   - Test reports that show the total, passed, and failed test case counts
   - Reports that gauge the performance of our product under test
   - Reports that show how long the pipeline took to execute — overall and per step
   - Traceability reports that show which commits landed in a build and which tickets — such as Jira or GitHub tickets — are associated with a build

   This feedback allows us to optimize not only our product, but the pipeline that builds it as well.

By following these tips, we can build an effective pipeline that meets our business needs and provides our users and customers with the greatest value and least amount of friction.

## Conclusion

CI/CD pipelines are not a magic answer to all of our release problems. While they are important tools that can dramatically improve the release of our software, they are only as effective as our underlying release processes. To create effective pipelines, we need to streamline our release processes and be vigilant so that our pipelines stay as simple and as automated as possible.

Further reading:

- *Continuous Delivery* by Jez Humble & David Farley
- *Continuous Delivery Patterns and Anti-Patterns* by Nicolas Giron & Hicham Bouissoumer
- *Continuous Delivery Guide* by Martin Fowler
- *Continuous Delivery Pipeline 101* by Juni Mukherjee
- ContinuousDelivery.com

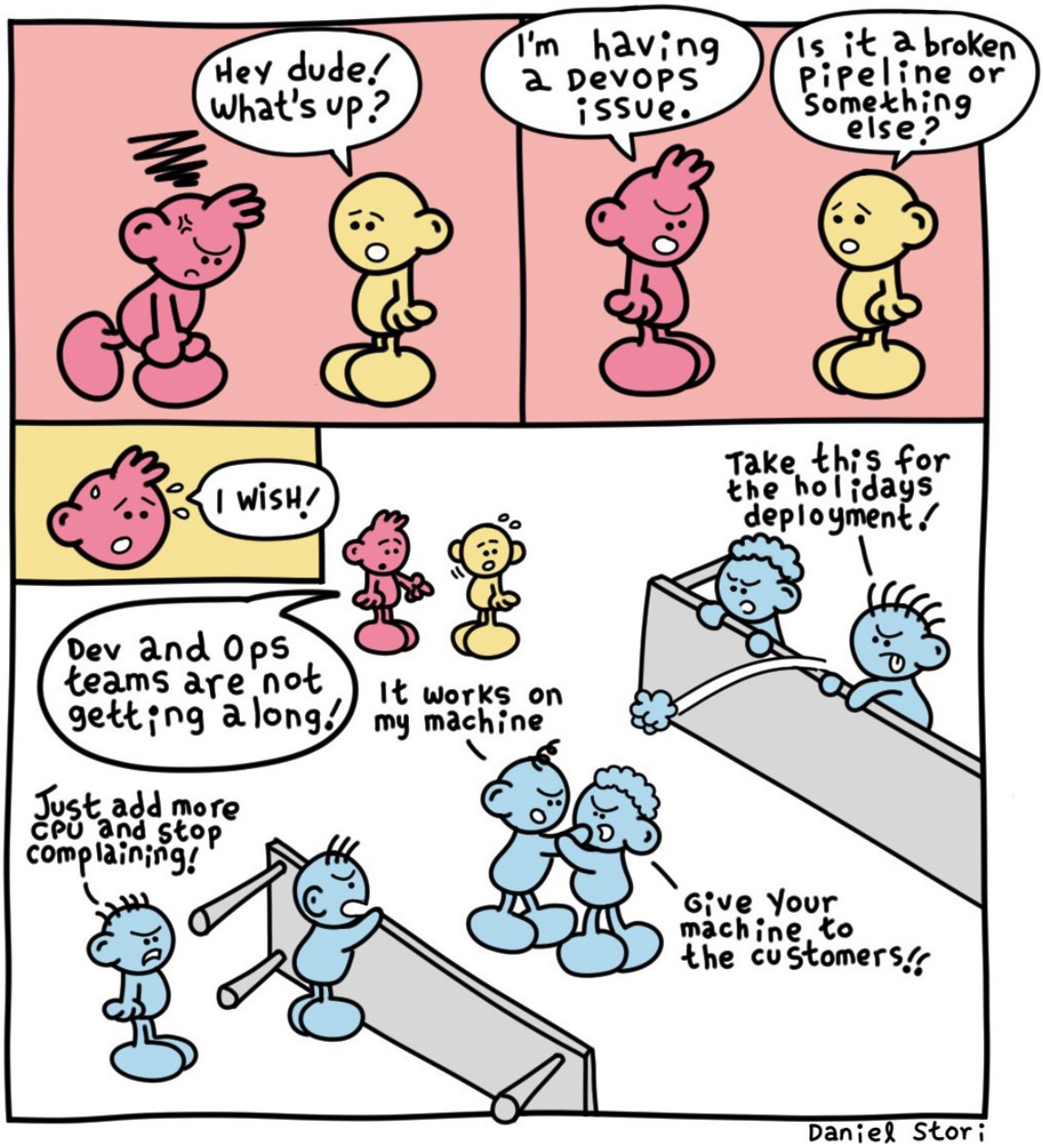
**Justin Albano, Software Engineer at IBM**
@albanoj2 on DZone | @justin-albano on LinkedIn

Justin Albano is a Software Engineer at IBM responsible for building software-storage and backup/recovery solutions for some of the largest worldwide companies, focusing on Spring-based REST API and MongoDB development. When not working or writing, he can be found practicing Brazilian Jiu-Jitsu, playing or watching hockey, drawing, or reading.

CONTRIBUTOR INSIGHTS

# Dev vs. Ops: Conflicted? So Are We.

By Daniel Stori, Software Development Manager at AWS




**Daniel Stori, Software Development Manager at AWS**
@Daniel Stori on DZone | @dstori on LinkedIn | @turnoff_us on Twitter | turnoff.us

Passionate about computing since writing my first lines of code in Basic on Apple 2, I share my time raising my young daughter and working on AWS Cloud Quest and AWS Industry Quest, a fun learning experience based on 3D games. In my (little) spare time, I like to make comics related to programming, operating systems, and funny situations in the routine of an IT professional.

CONTRIBUTOR INSIGHTS

# Source Code Management for GitOps and CI/CD

**By Yitaek Hwang, Software Engineer at NYDIG**

GitOps has taken the DevOps world by storm since Weaveworks introduced the concept back in 2017. The idea is simple: use Git as the single source of truth to declaratively store and manage every component for a successful application deployment. This can include infrastructure-as-code (e.g., Terraform, etc.), policy documents (e.g., Open Policy Agent, Kyverno), configuration files, and more. Changes to these components are captured by Git commits and trigger deployments via CI/CD tools to reflect the desired state in Git.

GitOps builds on recent shifts towards immutable infrastructure via declarative configuration and automation. By centrally managing declarative infrastructure components in Git, the system state is effectively tied to a Git commit, producing a versioned, immutable snapshot. This makes deployments more reliable and rollbacks trivial. As an added benefit, Git provides a comprehensive audit trail for changes and puts stronger guardrails to prevent drifts in the system.

Finally, it promotes a more consistent CI/CD experience, as all operational tasks are now fully captured via Git actions. Once the pipeline is configured, developers can expect a standard Git workflow to promote their changes to different environments.

Even though the benefits of GitOps are well documented, best practices for implementing GitOps are still being formed. Afterall, the implementation details will depend on the nature of the existing code repositories, size and makeup of the engineering teams, as well as practical needs for imperative changes (e.g., emergency rollback or break glass procedures).

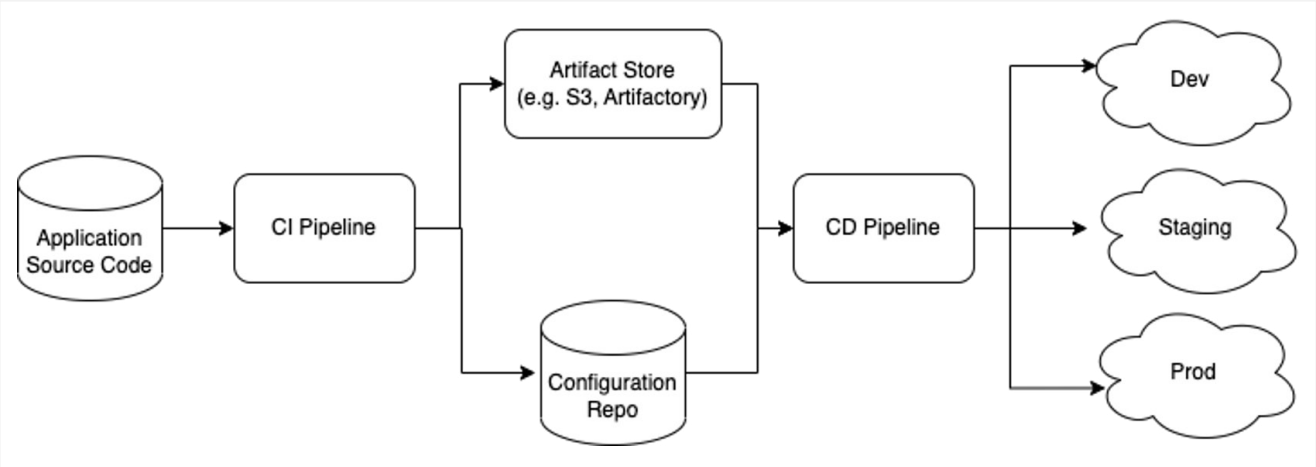
In this article, we'll look at how to choose the best strategy for embracing GitOps with the aforementioned considerations in mind.

## Source Code Management

The first consideration to adopting GitOps is deciding what artifacts to store in which Git repository. Some tech companies, such as Google and Facebook, are staunch supporters of monorepos, utilizing sophisticated tools to build and deploy code. Others take the opposite approach, using multiple repos to segregate applications or products to manage them separately.

Fortunately, GitOps is not tied to a particular framework, but unless your organization already has robust tooling to deal with monorepo builds, such as Bazel, the general recommendation is to at least separate application source code from deployment artifacts.

**Figure 1: Typical GitOps flow**



The benefits to separating the two repositories are multifold:

1. Deployment cadence for application and infrastructure changes can be more easily separated and controlled. For example, application teams may want every commit to trigger a deployment in lower environments, whereas infrastructure teams may want to batch multiple configuration changes before triggering a deployment (or vice-versa).
2. There may be compliance or regulatory requirements to separate who has access to deploy certain aspects of the application stack as a whole. Some organizations may only allow a Production Engineering or SRE team to trigger production deployments. Having separate repos makes access and audit trails easier to configure.
3. For applications with dependent components that necessitate deployment as a single unit, a separate configuration repo allows multiple application or external dependency repos to push changes independently. Then the CD tool can monitor a single configuration repo and deploy all the components at the same time.

With that said, the exact point of separation for what belongs in the application repo versus the deployment artifacts repo depends on the composition of your team. Small startups may expect application developers to be responsible for application code and other deployment artifacts (e.g., Dockerfile, Helm charts, etc.). In that case, keeping those manifests in the same repo and just keeping Terraform configs in another repo may make sense.

As for larger organizations with dedicated DevOps/infrastructure teams, these dedicated teams may own Kubernetes components exclusively and maintain those separately from application code.

### DEPLOYMENT REPO SETUP

The logical next question is, how many repos should hold deployment configs? The answer to this question is driven by similar considerations as above but with influence from the infrastructure topology.

- For **small teams** with a small number of cloud accounts/projects, it will be easier to have a single repo to host all deployment configs to trigger deployments to a small number of environments. As the infrastructure evolves, non-prod artifacts can be separated from prod repos.
- For **mid-sized teams** with slightly more sophisticated tooling and complex cloud infrastructure (e.g., multiple projects nested under organizations, hybrid-/multi-cloud), a repo per team may work well. This way different controls can be implemented based on security or compliance needs.
- At the other end of the spectrum, a repo per service and environment provides the most flexibility in terms of controls for **large teams** with robust tooling.

## Other CI/CD Considerations

The main goal of GitOps is to leverage the power of Git to store the single source of truth in terms of the deployed state. In practice, however, what constitutes the versioned, immutable artifact will be determined by the state of CI/CD tools for each team. For teams with slow pipelines, it may be undesirable to trigger frequent updates. Likewise for product teams needing to interface with external constituents, versioned releases may need to be coordinated with business partners. In such cases, designing the CI pipeline to update the configuration repo on tagged releases may be beneficial. On the other hand, for agile teams with robust CD tools for canary releases, frequent deployments may be favored to collect real-time user feedback.

Another critical component to call out with GitOps is secret management. Since secrets can't be checked into Git as plaintext, a separate framework is required to deal with secrets. Some frameworks like Bitnami Sealed Secrets check encrypted data into Git and use a controller/operator on the deployed environment to decrypt the secrets. Others separate secrets entirely and leverage secret stores such as Hashicorp Vault.

Finally, it's important to call out that even with GitOps, configuration drift may still occur. In fact, for small teams with immature tooling, it may be critical to leave some room for imperative changes. For time critical outages, manual fixes may be necessary to restore the service first before running through the official pipeline for a long-term fix. Before enforcing stringent policies, test rollback and restore strategies to ensure that GitOps systems do not hinder emergency fixes.

## Conclusion

*"GitOps is the best thing since configuration as code. Git changed how we collaborate, but declarative configuration is the key to dealing with infrastructure at scale, and sets the stage for the next generation of management tools."*

– Kelsey Hightower

GitOps applies the best practices that application developers have learned over the years from interacting with Git to declarative configuration and infrastructure. GitOps produces a versioned, immutable state of the system with an audit trail

via a tried-and-true system with which developers are already familiar. In this article, we walked through some considerations for implementing GitOps in your organization based on team size, composition, and state of your CI/CD tooling. For some, a monorepo holding everything in one place may work best, whereas for others, granular controls enforced at multiple repos are a requirement.

The good news is that GitOps is flexible enough to adapt as your needs change. There is a plethora of GitOps tooling available in the market today. Start with a simple tool to reap the benefits of GitOps as you grow your technology stack to boost developer productivity.

References:


- "5 GitOps Best Practices" by Alex Collins
- "Guide to GitOps," Weaveworks



**Yitaek Hwang, Software Engineer at NYDIG**

@yitaek on DZone | @yitaekhwang on LinkedIn | yitaekhwang.com

Yitaek Hwang is a software engineer at NYDIG working on Bitcoin technology. He often writes about cloud, DevOps/SRE, and crypto topics.

CONTRIBUTOR INSIGHTS

# CI/CD for Data Science and Machine Learning

Leverage the Benefits of Automation

**By John Nielsen, Developer at J.Nielsen**

Continuous integration (CI) and continuous delivery (CD) are two core principles of DevOps that can be applied to machine learning (ML) and data science projects, enabling teams to quickly iterate on their data and models, and they can deploy them in a more reliable, secure manner. CI/CD pipelines provide end-to-end automation, allowing data scientists and ML engineers to focus on the tasks for which they are best suited — such as data engineering, cloud architecture, analytics, model training, and more. With CI/CD in place, better quality control, faster iterations, and increased collaboration with other team members can be ensured.

## CI/CD Automation for Data Science and ML Applications

In the context of data science and ML applications, CI/CD is a tool to help streamline the development process. Data scientists and ML engineers can quickly develop and deploy their applications by automating the process of code writing, testing, and deployment. Let's explore the benefits and challenges of CI/CD within this scope.

### BENEFITS AND CHALLENGES

Automation is one of the key benefits of applying CI/CD to data science and machine learning. The process of building, testing, and deploying models can be automated, which reduces the risk of human error and increases the speed and reliability of the overall process. This leads to faster iteration, more frequent releases, and improved collaboration among data science and ML teams. Additional benefits include:

- **Increased efficiency and speed of model development** – automatically test and deploy new models, reducing the time it takes to bring a new model into production
- **Improved collaboration and coordination** – work on different parts of a project simultaneously, automatically merging changes without conflicts
- **Enhanced reproducibility and traceability** – track the entire model development process, from code to data to results, making it easy to reproduce and understand the decisions that led to a particular model.
- **Better model governance** – improve management and governance of organizations' ML models to reduce risk and increase trust through the creation of a clear and auditable process for model development and deployment

Despite the numerous benefits, setting up CI/CD for data science and ML applications can be a daunting task. It is not as easy as setting up a CI/CD pipeline for Go, Node.js, or any other programming language. There are certain things that need to be addressed before moving forward, such as the following challenges:

- **Data dependencies** – The management of complex data dependencies in data science and ML applications can make it challenging to automate the process of building, testing, and deploying models.
- **Data versioning** – This is a crucial part, as it allows for tracking changes to data and reproducing experiments.
- **Environment setup** – Data science and ML applications often have complex environment dependencies, such as specific versions of libraries and frameworks, that make it challenging to set up and manage the environments needed to run the models.
- **Reproducibility** – Ensuring reproducibility of the models and experiments is a key requirement in data science and ML, but it's hard to achieve in a CI/CD pipeline.
- **Security** – Sensitive data is often involved, and organizations need to ensure that the data is protected throughout the CI/CD pipeline.
- **Scalability** – Data science and ML applications can require large amounts of data and computational resources, which can make it challenging to scale the CI/CD pipeline.

It is essential for organizations to do proper research on what they are looking for with a CI/CD approach and what the achievable goals would be. Hiring the right talent, providing adequate training, and making informed decisions are some of the most important factors for ensuring success and reaping the benefits.

## Automating Data Science and ML Applications With CI/CD

CI/CD can be applied to data science and ML applications to automate the process of building and deploying models. By using CI/CD, data scientists can easily test and deploy their models in a controlled and efficient manner. This can be done by creating a pipeline that integrates the various steps of the data science process, such as data preparation, model training, and evaluation. The pipeline can be configured to automatically run tests and checks on the code and data before deploying the model to production, which allows developers to quickly identify and fix any bugs that may be present in the code.

Let's dig a little deeper and see how we can properly implement CI/CD for data science and ML applications. There are several steps you can take to approach the implementation:

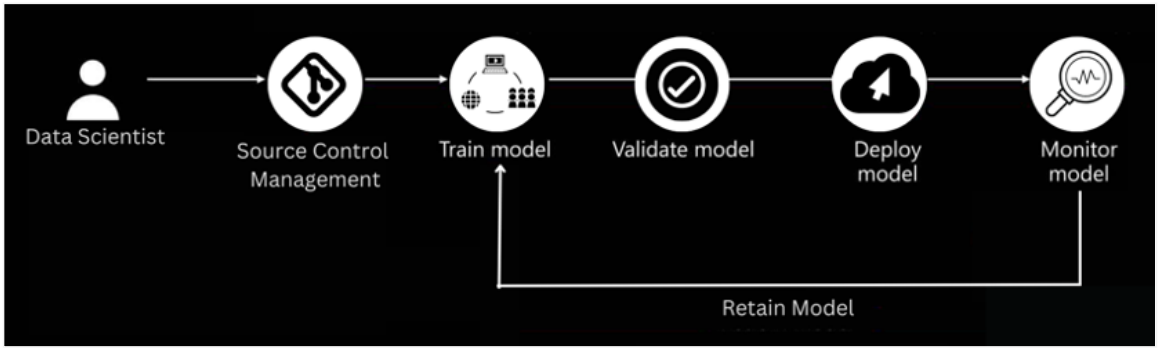
- Set up a version control system, such as Git, to track changes to your code and data.
- Use a CI tool to automatically build, test, and validate your code and data.
- Create automated tests for your code and data to ensure that changes do not break existing functionality.
- Use a CD tool to automatically deploy your code and data to a staging or production environment.
- Set up monitoring and logging to track the performance of your deployed models and detect any issues.
- Implement a feedback loop to track and use the results of your deployed models to improve them over time.
- Use containerization or virtualization technologies like Docker to package your code and dependencies for easy deployment.
- Use cloud-based services like AWS, GCP, or Azure to deploy your models and infrastructure.

It's important to remember that CI/CD is an iterative process, so you may need to experiment to find the best approach for your specific use case.

### A PRACTICAL CASE STUDY

One example of applying CI/CD in an ML application is a project called "MLflow on Kubernetes." This project aims to automate the deployment of ML models using a CI/CD pipeline, with MLflow as the ML platform and Kubernetes as the container orchestration system.

**Figure 1: Data/ML pipeline**




The pipeline starts with a developer committing code changes to a version control system (VCS) like Git. Next, a CI tool automatically builds and tests the code, ensuring it is stable and ready for deployment. If the tests pass, the pipeline continues to the CD stage, where the code is automatically deployed to a Kubernetes cluster. The deployed code includes the MLflow model and a Kubernetes deployment file, which describes how to run the model in a Kubernetes pod.

Once the model is deployed, it can be accessed via an API endpoint or a web UI, which allows users to send requests to the model and receive predictions in real time. Additionally, the pipeline can also include monitoring and logging tools to track the performance of the model and gather feedback from users.

Overall, this pipeline allows for efficient and automated deployment of ML models, reducing the effort required by developers and allowing them to focus on improving the model.

A practical CI/CD case study for data science applications would involve setting up a pipeline that automates the process of building, testing, and deploying a data science model. This pipeline would typically include the following steps:

- **Code integration** – The code for the data science model is integrated with the pipeline. This can be done using a version control system like Git, which allows multiple developers to work on the code simultaneously and keeps a record of all changes made to the code.
- **Building** – The code is built by compiling and packaging it so it can run on different environments. This can be done using a tool like Docker, which allows you to build a containerized environment that can be easily deployed on different machines.
- **Testing** – The model is tested to ensure it is working as expected. This can be done using unit tests, integration tests, and end-to-end tests. Automated testing frameworks like pytest can be used for this. Even CI/CD tools can help you build and test your applications.
- **Deployment** – The model is deployed to a production environment, where it can be used by end-users. This can be done using tools like Kubernetes or AWS Elastic Beanstalk.
- **Monitor** – After deploying, it is important to monitor the model's performance to understand how well it is doing in real-world scenarios.

Overall, a CI/CD pipeline for data science applications helps to ensure that the model is reliable, maintainable, and can be easily updated with new features and improvements.

## Conclusion

As the whole world is going crazy over data science and ML, there is a huge demand for the CI/CD approach. Applying CI/CD and DevOps principles to data science and ML projects can help streamline and automate the entire workflow — from data collection, training, and testing to model deployment. By automating the process, organizations can achieve faster results, improved scalability, and higher quality models. Gain a competitive advantage over others by adopting CI/CD and reap the enormous benefits that this approach provides. Happy DevOpsing!

**John Nielsen, Developer at J.Nielsen**
John is a technology writer who covers various DevOps topics. In addition to writing articles, he enjoys attending conferences and teaching developers. He splits his free time between playing cricket and singing.

CONTRIBUTOR INSIGHTS

# A Deep Dive Into AIOps and MLOps

**By Hicham Bouissoumer and Nicolas Giron, SREs at KumoMind**

Monitoring and managing a DevOps environment is complex. The volume of data generated by new distributed architectures (such as Kubernetes) makes it difficult for DevOps teams to effectively respond to customer requests. The future of DevOps must therefore be based on intelligent management systems. Since humans are not equipped to handle the massive volumes of data and computing in daily operations, artificial intelligence (AI) will become the critical tool for computing, analyzing, and transforming how teams develop, deliver, deploy, and manage applications.

## What Are Machine Learning Operations?

Machine learning operations (MLOps) refers to the lifecycle management of machine learning (ML) projects. It is a key concept of modern machine learning application development, and its purpose is to make the training, deploying, and maintaining of machine learning applications seamless and efficient. MLOps is not a set of specific technologies but rather an umbrella term for activities focused on building reliable and well-functioning machine learning models. It includes both development work practices and ways of working as a project team — essentially functioning as a set of best practices for machine learning application development.

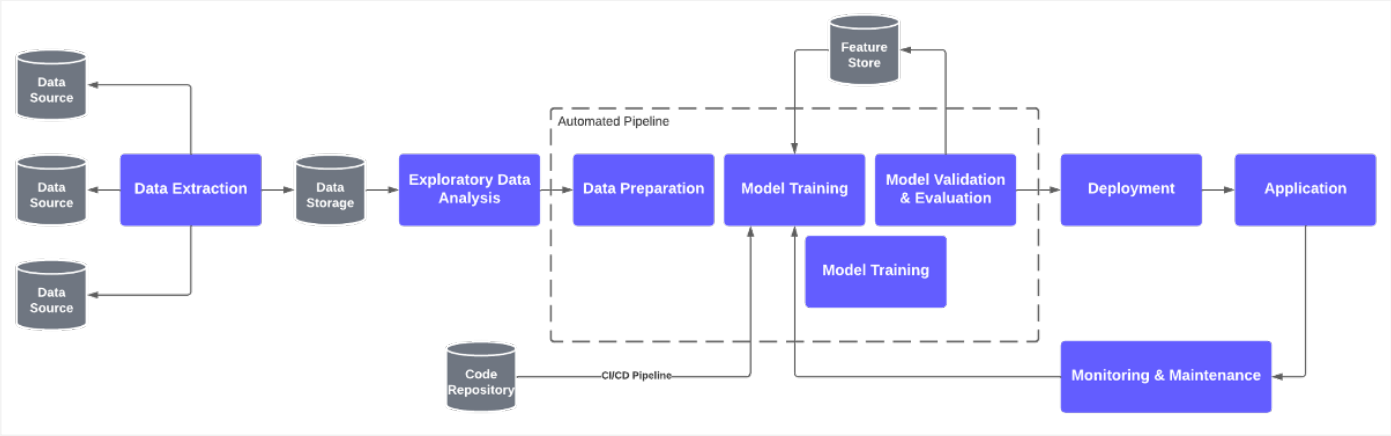
With the application of MLOps principles, data scientists can focus on the core development of machine learning models while the MLOps practices take care of tasks such as data cleaning, quality control, and model versioning.

Applying MLOps equally benefits business owners and clients. Automation increases the velocity of development, leading to faster results and more reliable machine learning models. This leads to shorter development times that, in turn, bring faster end-result delivery and cost effectiveness. Finally, automated quality control ensures more reliable solutions that are ensured and tested to function as intended, reducing the risk of faulty deployments.

### LIFECYCLE OF A MACHINE LEARNING MODEL

The lifecycle of a machine learning project differs from a traditional application. The diagram in Figure 1 details the steps to deploy a machine learning project in production:

**Figure 1: Machine learning model lifecycle**




1. **Data Extraction** – ingesting data from various sources
2. **Exploratory Data Analysis** – understanding the data format
3. **Data Preparation** – cleaning and processing th data for easy processing
4. **Model Training** – creating and training a model to process the data
5. **Model Validation and Evaluation** – evaluating the model on test data to validate the performances

6. **Model Versioning** – releasing a version of the model
7. **Model Deployment** – deploying the model in production

### CORE ELEMENTS OF MLOPS

There are several machine learning frameworks that allow you to deploy, manage, and monitor models — for example, KubeFlow is a toolkit that simplifies model management on the Kubernetes platform. A toolkit should be composed of:

- A version control to keep track of any changes in the datasets or the models
- A feature store to centralized data and frequently used features
- A tracker to monitor the performance of models in training
- A tool to train models using a set of optimal hyperparameters automatically
- A platform to deploy models in production
- A monitoring tool to track and govern machine learning models deployed in production

## What Are Artificial Intelligence Operations?

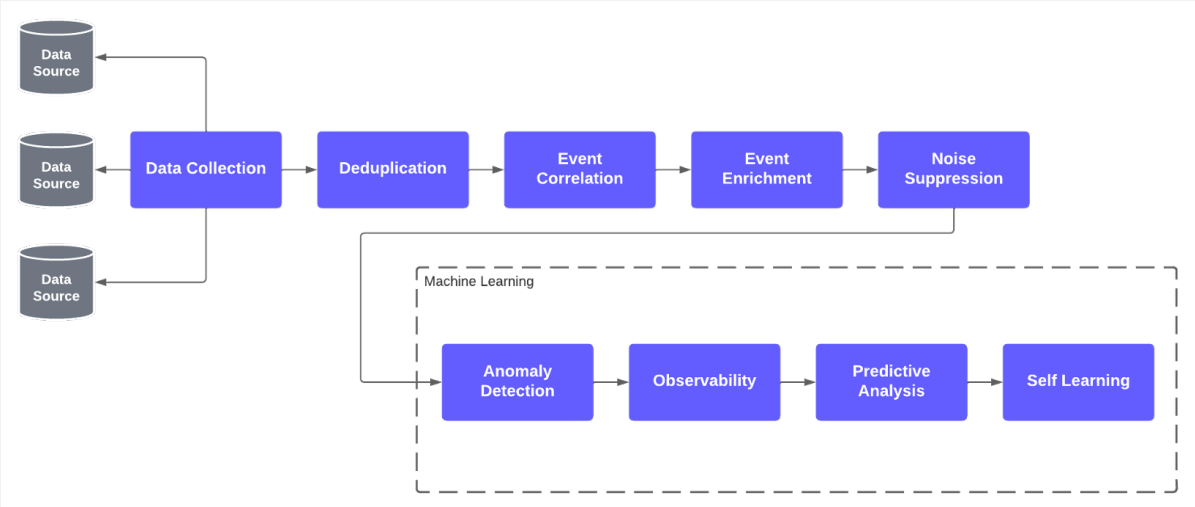
Gartner, creator of the term "artificial intelligence operations" (AIOps), defines it as the utilization of big data and machine learning to automate IT operations tasks, such as event correlation, identifying unusual events, and determining cause and effect. AIOps uses big data, analytics, and AI algorithms to analyze vast amounts of data generated by IT systems and applications in real time. This data includes log files, performance metrics, and security events, among others. The AI algorithms process this data to identify patterns, detect anomalies, and generate insights that can help IT teams resolve incidents quickly and prevent potential problems before they occur.

AIOps solutions can also automate manual tasks such as event correlation, root cause analysis, and incident resolution, freeing IT teams to focus on more strategic initiatives. AIOps can also help organizations achieve faster problem resolution, reduced downtime, and improved overall IT operations efficiency. It helps teams to work faster and smarter by unleashing the power of AI.

The core capabilities of AIOps that enable efficient digitization of workflows are:

1. **Process optimization** – Enhances efficiency throughout the enterprise by giving a comprehensive understanding of the connections and effects between systems. After identifying a problem, it facilitates refinement and ongoing monitoring of processes.
2. **Performance analytics** – Anticipates performance bottlenecks by examining trends and making necessary improvements as needed.
3. **Predictive intelligence** – Utilizes machine learning to categorize incidents, suggest solutions, and proactively alert critical issues.
4. **AI search** – Offers precise, personalized answers through semantic search capabilities.
5. **Configuration management database** – Enhances decision-making with visibility into the IT environment by connecting products throughout the digital lifecycle, allowing teams to comprehend impact and risk.

**Figure 2: AIOps lifecycle**



### CORE ELEMENT OF AIOPS

AIOps definitions vary among enterprises as each enterprise has unique needs and approaches to implementing AI solutions in IT operations. The primary objective of AIOps is to identify and respond to real-time issues efficiently. Some core components of AIOps can assist in the implementation of AI in IT operations:

1. **ML-based pattern discovery** – AIOps or IT analytics involves identifying patterns. Machine learning leverages the computational capability of computers to identify these patterns in IT data.
2. **Anomaly detection** – Unusual system behavior, such as downtime or poor customer experience, can occur from changes in normal behavior. AIOps enables the detection of any deviations from typical activities.
3. **Predictive insights** – AIOps introduces predictability in IT operations, enabling IT staff to proactively address issues before they occur, ultimately reducing the number of service desk tickets.
4. **Automated root cause analysis** – Simply having insights isn't enough. It's important to take action. In traditional IT management, staff monitor systems and take action as needed. However, with the growing volume of IT infrastructure issues, it can be difficult for staff to manage and resolve issues in a timely manner, especially when multiple systems are involved and root cause analysis can be time consuming. AIOps automates this process in the background.

### AIOPS TOOLSET

AIOps tools gather data from multiple sources to provide a comprehensive view of IT operations. They collect data such as application logs and measure system performance, breaking down silos of IT information and bridging the gap between software, hardware, and cloud issues. AIOps solutions aid IT operations by providing tools for root cause analysis, event correlation, and cloud mapping to support automation:

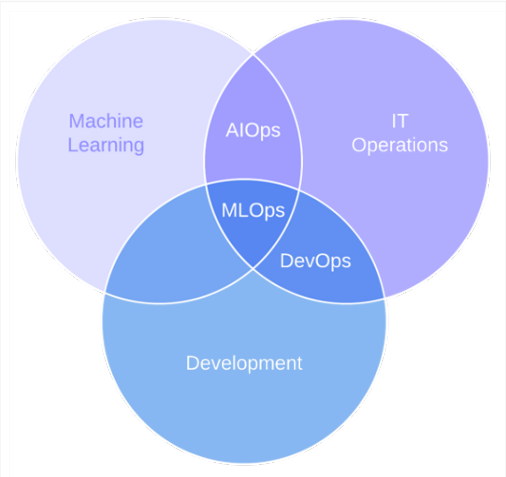
1. **Intelligent observability** – AIOps employs advanced monitoring techniques with the use of contextual information, AI, and automation to gain a complete understanding of IT issues. Precise root cause analysis with actionable insights is provided.
2. **Continuous automation** – Reduces manual effort in deployment, configuration, and management and automatically identifies and assesses the severity of issues in terms of user and business impact. Achieving continuous discovery, effortless deployments, and automatic dependency mapping is made possible.
3. **AI-assistance** – Performs efficient and error-free root cause analysis. Precise and reproducible results are achieved with the AI engine integrated into every aspect.

## What Is the Difference Between MLOps and AIOps?

Coupled with the increasing complexity of architectures of modern applications, the demands of this digital economy have made the role of IT operations much more complex. As a result, ML and AI have emerged to automate some manual business processes to increase efficiency.

MLOps and AIOps both aim to serve the same end goal: business automation. While MLOps bridges the gap between model building and deployment, AIOps focus on supporting and reacting to issues in real time and providing analytics to the operations team. AIOps combines big data and machine learning to automate performance monitoring, event analysis, correlation, and IT automation.

**Figure 3: AIOps vs MLOps vs DevOps**



MLOps, on the other hand, focuses on managing training and testing data that is needed to create machine learning models effectively. It is all about monitoring and managing ML models. **In other words, MLOps standardizes processes whereas AIOps automates machine monitoring**.

There are parallels in the teams and abilities needed to properly execute AIOps and MLOps, despite the obvious distinctions. It is worthwhile to consider where they intersect to determine which resources can support both disciplines.

## Conclusion

Organizations throughout the world are increasingly looking to automation technologies as a means of improving operational efficiency. This indicates that tech leaders are becoming more and more interested in MLOps and AIOps.

Machine learning systems can simplify data collection from various parts of the DevOps system like velocity, defects found, and burn rate. MLOps takes care of the continuous integration and deployment of the models. It allows users to shed light on important patterns and exploit data to extract meaningful information. It also implies surveillance and continuous model training in production in order to ensure the reliability and stability of those models.

AIOps can play a crucial role in accelerating DevOps efficiency. It is defined as the usage of big data and machine learning to automate operations such as event correlation, determining cause and effect, and identifying unusual events.

In other words, MLOps and AIOps can work together. Artificial intelligence will help boost performance by enabling instant development and operations cycles, and by delivering a compelling customer experience on these features. Machine learning will enable companies to gather metrics such as the number of integrations, the time between them, their success rate, and defects per integration, which are only valuable when they are accurately evaluated and correlated.


**Co-written by Hicham Bouissoumer, SRE at KumoMind**

Hicham is an engineer with about 8 years of experience working as a quality assurance engineer, DevOps engineer, and cloud architect. Always on the lookout for new challenges and problems to solve but also keen on sharing his knowledge, he decided to co-found KumoMind with Nicolas Giron to write tech-related content for the community.

**Co-written by Nicolas Giron, SRE at KumoMind**

Nicolas is an IT engineer with over 10 years of experience as a developer, cloud architect, DevOps, and SRE. His background and curiosity have allowed him to develop his technical skills in different fields beyond his areas of expertise. As co-founder of KumoMind with Hicham Bouissoumer, they aim to share their deep expertise in open-source technologies, cloud computing, and emerging technologies.

ADDITIONAL RESOURCES

# Diving Deeper Into DevOps and CI/CD

## REFCARDS

**Continuous Integration Patterns and Anti-Patterns**
Reap the full benefits of enhanced code quality, better testing practices, and early error detection with proper implementation of CI processes. This Refcard explains detailed patterns and anti-patterns for core areas of CI, including version control, the build stage, pipeline monitoring, documentation, as well as communication and collaboration across teams and within the organization.

**Getting Started With CI/CD Pipeline Security**
The increasingly distributed nature of CI/CD frameworks has made organizations more vulnerable to attacks. In this Refcard, you'll learn about the primary focus areas of CI/CD pipeline security, review common pipeline threats and security challenges, as well as walk through seven steps to get started with securing your pipelines.

## MULTIMEDIA

**The Pipeline: All Things CD & DevOps**
Covering a variety of topics in the CD and DevOps realms, host Jacqueline Salinas interviews industry professionals to bring you up-to-date news and all the information you need surrounding buzzy ideas. In addition, the experiences shared in the interviews will provide you with useful knowledge to navigate challenges in your projects.


**DevSecOps Talks**
Brought to you by DevSecOps practitioners Julien Bisconti, Andrey Devyatkin, and Mattias Hemmingsson, the DevSecOps Talks podcast was created to help anyone struggling with security on a daily basis. With an unbiased take, the hosts will help you through your security challenges and information overload as well as provide you with DevSecOps news — sans marketing content.


**The Azure DevOps Podcast**
If you're a developer or a DevOps professional working with Microsoft technologies, check out this weekly podcast with episodes covering full-system testing, content development, the evolution of DevOps, microservices, and more. Jeffrey Palermo brings you interviews with industry professionals to uncover better methods and to share success stories.

## TREND REPORTS

**DevOps:** CI/CD and Application Release Orchestration
In DZone's 2021 *DevOps* Trend Report, we provide insight into how CI/CD has revolutionized automated testing, offer advice on why an SRE is important to CI/CD, explore the differences between managed and self-hosted CI/CD, and more. The goal is to offer guidance on how to best adopt DevOps practices to help scale the productivity of teams.

**CI/CD:** Automation for Reliable Software Delivery
In 2020, DevOps became more crucial than ever as companies moved to distributed work and accelerated their push toward cloud-native and hybrid infrastructures. Our Trend Report examines what this acceleration looked like for development teams across the globe and dives deeper into the latest DevOps practices that are advancing CI, CD, and release automation.

**KodeKloud**
Want to learn more about DevOps? This YouTube channel is for you. KodeKloud will teach you more about DevOps practices and help you build your career through video tutorials and tips. Videos range from just a few minutes to over an hour, making it easy to find something that will fit your schedule.

**Jeff Geerling**
With numerous years of experience in programming and DevOps, Jeff Geerling brings experience, passion, and fun to his YouTube channel. His videos span numerous subjects but be sure to check out his Kubernetes 101 and Ansible playlists. Interested in more? Don't miss the link to his personal site where he writes blog posts and provides links to his books.

**Kubernetes in the Enterprise Virtual Roundtable**
DZone's Virtual Roundtable brings together the *Kubernetes in the Enterprise* Trend Report's SMEs and partners to discuss research findings and the future of Kubernetes. Panelists address numerous topics and questions, including the question: With the expectations of Kubernetes becoming more entrenched into systems, what do the adoption and deployment methods look like compared to previous years?

ADDITIONAL RESOURCES

# Solutions Directory

This directory contains tools for automation, IaC, pipelines, testing, and more to help manage application development and deployment, CI/CD pipelines, and DevOps practices. It provides pricing data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

**DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY**

2023 PARTNERS

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Buildkite | Buildkite CI/CD | Saas CI/CD solution | Free tier | buildkite.com/ci-cd |
| JFrog | JFrog Software Supply Chain Platform | Universal binary repository, software supply chain security, DevOps platform | Free tier | jfrog.com/platform |
| Progress Chef | Chef | Configuration management DevOps tool | Open source | chef.io/products |
| Release | Release | Automated, full-stack environments as a service | Trial period | release.com |
| Sauce Labs | Sauce API | Contract testing, mock testing, data-driven functional testing, load testing, and test management | Free tier | saucelabs.com/products/api-testing |
| | Sauce Cross-Browser | Cross-browser testing on every browser and OS combination | Trial period | saucelabs.com/products/cross-browser-testing |
| | Sauce Error Reporting | Error monitoring and reporting with Backtrace | | saucelabs.com/products/error-reporting |
| | Sauce Insights | Real-time visibility to identify and remediate high-impact issues first | | saucelabs.com/products/sauce-insights |
| | Sauce Low-Code | AI-powered, low-code testing for the enterprise | By request | saucelabs.com/products/low-code-testing |
| | Sauce Mobile | Mobile test and error reporting on real devices and virtual devices | Trial period | saucelabs.com/platform/mobile-testing |
| | Sauce Performance | Front-end performance insight before app or website goes live | | saucelabs.com/products/sauce-performance |
| | Sauce Visual | Automated UI and visual testing | By request | saucelabs.com/products/visual-testing |
| Slack | Slack | Team collaboration | By request | slack.com/solutions/engineering |
| **Company** | **Product** | **Purpose** | **Availability** | **Website** |
| AccelQ | Automate Mobile | No-code mobile test automation | Trial period | accelq.com/products/test-automation-mobile |
| | Automate Web | Test automation tool for web, API, mobile, desktop, and more | | accelq.com/products/test-automation-web |
| Amazon Web Services | CloudFormation | Model, provision, and manage resources by treating IaC | Free tier | aws.amazon.com/cloudformation |
| | CodeBuild | Build and test code with automatic scaling | | aws.amazon.com/codebuild |
| | CodePipeline | CD pipeline automation and management | | aws.amazon.com/codepipeline |
| Apache Software Foundation | Apache Ant | Java library and command-line tool | Open source | ant.apache.org |
| Appcircle | Appcircle | Mobile CI/CD platform with testing and store deployment | Free tier | appcircle.io |

## DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| AppVeyor | AppVeyor | Continuous integration for Windows developers | Free tier | appveyor.com |
| ARCAD Software | ARCAD for DevOps | End-to-end CI/CT/CD cycle automation for IBM i | By request | arcadsoftware.com/products/arcad-for-devops |
| | DROPS | Application release automation | | arcadsoftware.com/products/drops-application-release-automation-solution |
| Argo | Argo CD | Declarative continuous delivery with a fully loaded UI | Open source | argoproj.github.io/cd |
| Armory | CD-as-a-Service | Declarative deployments with a GitOps experience | Free tier | armory.io/products/continuous-deployment-as-a-service |
| | Continuous Deployment Managed | Continuous delivery platform | By request | armory.io/products/continuous-deployment-managed |
| Atlassian | Bamboo | Continuous integration, deployment, and release management | Trial period | atlassian.com/software/bamboo |
| AutoCloud | AutoCloud | Build and manage secure cloud platforms | Free tier | autocloud.io |
| AutoRABIT | ARM | CI/CD for Salesforce | By request | autorabit.com/products/automated-release-management |
| | CodeScan | Static code analysis for Salesforce | | autorabit.com/products/codescan |
| | Vault | Data protection for Salesforce | | autorabit.com/products/vault-data-backup-recovery |
| Basis Technologies | ActiveControl | DevOps automation for SAP | By request | basistechnologies.com/products/activecontrol |
| | Testimony | Robotic test automation for SAP | | basistechnologies.com/products/testimony |
| BitRise | BitRise | Fully hosted DevOps and CI/CD for mobile apps | Trial period | bitrise.io |
| BMC | Control-M | Integrate, automate, and orchestrate application workflows | Trial period | bmc.com/it-solutions/control-m.html |
| | Helix Operations Management | Service-centric monitoring, event management, and AI/ML-based root cause isolation | | bmc.com/it-solutions/bmc-helix-operations-management.html |
| Broadcom | Automic Automation | Service orchestration and automation platform | By request | broadcom.com/products/software/automation/automic-automation |
| | Automic® Continuous Delivery Automation | Real-time app deployment monitoring and management | | broadcom.com/products/software/continuous-delivery/automic-continuous-delivery-automation |
| | Continuous Delivery Director | Release planning and management | | broadcom.com/products/software/continuous-delivery/automic-continuous-delivery-director |
| Buddy | Buddy | DevOps automation | Free tier | buddy.works |
| Buildbot | Buildbot | Job scheduling system | Open source | buildbot.net |
| Bunnyshell | Bunnyshell | Create and manage full-stack environments | Free tier | bunnyshell.com |
| Cerberus Testing | Cerberus Testing | Scalable test automation platform | Free tier | cerberus-testing.com |
| CFEngine | CFEngine | Infrastructure, security, and compliance automation | Free tier | cfengine.com |
| CircleCI | CircleCI | Continuous integration platform | Free tier | circleci.com |

## DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Clarive | Clarive | Environment and deployment management and automation | Free tier | clarive.com |
| Cloud Foundry Foundation | Korifi | Cloud-native app deployment on Kubernetes | Open source | cloudfoundry.org/technology/korifi |
| Cloud Native Computing Foundation | Keptn | Cloud-native application lifecycle orchestration | Open source | keptn.sh |
| | OpenKruise | Application management automation on Kubernetes | | openkruise.io |
| CloudBees | CloudBees | Software delivery platform | By request | cloudbees.com |
| Cloudify | Cloudify | Deployment, configuration, and day-2 operation automation | By request | cloudify.co |
| Cloudnosys | Cloudnosys | Cloud security and compliance at scale | By request | cloudnosys.com |
| Cloudsmith | Cloudsmith | Artifact management | Trial period | cloudsmith.com |
| Cloudwise | Cloudwise | Full-stack intelligent business operation platform | Trial period | cloudwise.com |
| Codefresh | Codefresh | CI/CD platform | Free tier | codefresh.io |
| Codemagic | Codemagic | CI/CD for mobile | Free tier | codemagic.io/start |
| Concourse | Concourse | CI/CD automation | Open source | concourse-ci.org |
| Copado | CI/CD | Deployment automation | By request | copado.com/ci-cd |
| | Essentials | Salesforce DevOps | Free tier | copado.com/essentials |
| Coralogix | Coralogix | Full-stack observability | Trial period | coralogix.com/platform |
| D2iQ | D2iQ Kubernetes Platform | Kubernetes platform | Trial period | d2iq.com/kubernetes-platform |
| | DKP Enterprise | Kubernetes platform | | d2iq.com/products/enterprise |
| | DKP Essential | Kubernetes platform | | d2iq.com/products/essential |
| DBmaestro | Database Release Automation | Database CI/CD | By request | dbmaestro.com/database-release-automation |
| | Database Source Control | Database code, structure, and content management | Trial period | dbmaestro.com/database-source-control |
| Digital.ai | Deploy | Application deployment automation and scaling | By request | digital.ai/products/deploy |
| | Release | Software release automation and orchestration | | digital.ai/products/release |
| | TeamForge | Governance, compliance, and code security | | digital.ai/products/teamforge |
| DuploCloud | DuploCloud | DevOps automation | By request | duplocloud.com |
| easyCIS | easyCIS | Build automation and maintenance | Free | easycis.aspone.cz |
| Flexagon | FlexDeploy | DevOps value stream delivery platform | By request | flexagon.com/flexdeploy/devops-vsdp-platform |
| Flosum | DevOps | Mange and govern deployments in Salesforce environment | By request | flosum.com/products/devops |
| Flux | Flux | Open and extensible continuous delivery solution for Kubernetes | Open source | fluxcd.io |
| Gearset | Gearset | Salesforce DevOps | Trial period | gearset.com |

**DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| GitLab | GitLab | DevSecOps | Free tier | about.gitlab.com |
| Google Cloud | Deployment Manager | Infrastructure deployment | Trial period | cloud.google.com/deployment-manager/docs |
| Gradle | Gradle | Build automation tool for flexibility and performance | Open source | gradle.org |
| Graham Campbell Technology | StyleCI | Continuous integration service that enforces code style preferences | By request | styleci.io |
| Gridlastic | Gridlastic | Selenium grid testing infrastructure | Free tier | gridlastic.com |
| Harness | Harness | Software delivery platform | Trial period | harness.io/products/platform |
| HashiCorp | Terraform | Infrastructure automation | Free tier | terraform.io |
| HCL Software | Secure DevOps | Automated testing and security scanning, value stream management | By request | hcltechsw.com/devops |
| | SoFy | Cloud-based software deployment | | hcltechsw.com/sofy |
| IBM | Cloud Continuous Delivery | UI- and CLI-based DevOps workflows | Trial period | ibm.com/cloud/continuous-delivery |
| | Cloud Pak | DevOps management with AI analysis | By request | ibm.com/products/cloud-pak-for-watson-aiops |
| | Engineering Lifecycle Management | Software for product and application lifecycle management | Trial period | ibm.com/products/engineering-lifecycle-management |
| | Instana | Application performance monitoring | | ibm.com/products/instana |
| | UrbanCode | CI/CD and release management | By request | ibm.com/cloud/urbancode |
| Inedo | BuildMaster | Application release automation tool | Free tier | inedo.com/buildmaster |
| InfluxData | InfluxDB | Kubernetes monitoring solution | Trial period | influxdata.com/solutions/kubernetes-monitoring-solution |
| | Telegraf | Data collection agent | Free tier | influxdata.com/telegraf |
| Ionic | Appflow | Cloud-based app delivery platform | Trial period | ionic.io/appflow |
| Jenkins | Jenkins | Automation server | Open source | jenkins.io |
| Jenkins X | Jenkins X | Cloud-native CI/CD | Open source | jenkins-x.io |
| JetBrains | Qodana | Code quality platform for CI | Free tier | jetbrains.com/qodana |
| | Space | Complete software development platform | | jetbrains.com/space |
| | TeamCity | CI/CD software platform | Trial period | jetbrains.com/teamcity |
| JHipster | JHipster | Web apps and microservices development platform | Open source | jhipster.tech |
| Knapsack Pro | Knapsack Pro | CI infrastructure test management | Free tier | knapsackpro.com |
| LaunchDarkly | LaunchDarkly | Feature flag and toggle management | Trial period | launchdarkly.com |
| Liquibase | Liquibase | Database CI/CD and automation | Free tier | liquibase.com |
| Mend | SAST | Static application security testing | By request | mend.io/sast |
| | SCA | Open-source software management | | mend.io/sca |

**DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Micro Focus | CODAR | Application release automation | By request | microfocus.com/en-us/products/codar-continuous-deployment/overview |
| | Hybrid Cloud Management | Continuous delivery and release automation | | microfocus.com/en-us/products/hybrid-cloud-management-native/overview |
| | Operations Bridge | AIOps | | microfocus.com/en-us/products/operations-bridge |
| | ValueEdge | End-to-end VSM platform | | valueedge.microfocus.com |
| Microsoft Azure | Artifacts | Fully integrated package management for CI/CD pipelines | Trial period | azure.microsoft.com/en-us/products/devops/artifacts |
| | Automation | Process automation for cloud management | | azure.microsoft.com/en-us/products/automation |
| | Monitor | Full observability into apps, infrastructure, and network | | azure.microsoft.com/en-us/products/monitor |
| | Pipelines | Continuously build, test, and deploy to any platform and cloud | | azure.microsoft.com/en-us/products/devops/pipelines |
| MidVision | RapidDeploy | Release automation, orchestration, and management | By request | midvision.com/rapiddeploy-overview |
| NS1 | Managed DNS | Application delivery and performance optimization | By request | ns1.com/products/managed-dns |
| Octopus Deploy | Octopus Deploy | DevOps automation | Trial period | octopus.com |
| Opsera | Opsera | Continuous orchestration of DevOps tools, pipelines, and insights | By request | opsera.io |
| OpsHub | OpsHub Azure DevOps Migrator | Azure DevOps migration tool | By request | opshub.com/products/opshub-azure-devops-migrator |
| | OpsHub Integration Manager | Integration platform | | opshub.com/products/opshub-integration-manager |
| Outsystems | Outsystems | Low code for app development | Free tier | outsystems.com/low-code-platform |
| Pantheon | Pantheon | WebOps platform | By request | pantheon.io |
| Perforce | Perfecto | Web and mobile app testing | Trial period | perfecto.io |
| | Puppet Enterprise | Infrastructure automation | | puppet.com/products/puppet-enterprise |
| Plutora | Plutora | Value stream management platform | By request | plutora.com |
| Prisma Cloud | Bridgecrew | Code-to-cloud security automation with DevSecOps | Free tier | bridgecrew.io/platform |
| Probo | Probo | Continuous integration and collaboration tool | Trial period | probo.ci |
| Prodly | Prodly DevOps | Salesforce DevOps automation | By request | prodly.co/products/salesforce-devops |
| Provar | Provar | Salesforce automated testing | By request | provartesting.com |
| Pulumi | Pulumi | Infrastructure as Code tool | Open source | pulumi.com/product |
| | Pulumi Service | Fully managed cloud engineering | Free tier | pulumi.com/product/pulumi-service |
| Quali | CloudShell | Infrastructure automation | By request | quali.com/cloudshell |
| | Torque | Infrastructure control plane | | quali.com/torque |
| Rainforest QA | Rainforest QA | No-code test automation | Free tier | rainforestqa.com |

**DZONE'S 2023 DEVOPS SOLUTIONS DIRECTORY**

| Company | Product | Purpose | Availability | Website |
|---|---|---|---|---|
| Razorops | Razorops | Container-native CI/CD platform | Free tier | razorops.com |
| Red Hat | Ansible Automation Platform | IT automation | By request | ansible.com |
| Redgate | Redgate | DevOps for the database | Trial period | red-gate.com/products/flyway |
| Rendered Text | Semaphore | Continuous integration and deployment service | Trial period | semaphoreci.com |
| Scrutinizer | Scrutinizer | Software quality management | Trial period | scrutinizer-ci.com |
| Spinnaker | Spinnaker | Multi-cloud continuous delivery platform | Open source | spinnaker.io |
| Splunk | Infrastructure Monitoring | Infrastructure performance monitoring and troubleshooting | Trial period | splunk.com/en_us/products/infrastructure-monitoring.html |
| | Splunk IT Service Intelligence | AIOps for monitoring and observability | By request | splunk.com/en_us/products/it-service-intelligence.html |
| StrongDM | StrongDM | Infrastructure access platform | Trial period | strongdm.com |
| SUSE | SUSE Manager | Infrastructure management | By request | suse.com/products/suse-manager |
| ThoughtWorks | GoCD | CI/CD system | Open source | gocd.org |
| Travis CI | Travis CI | Cloud-based CI/CD | Trial period | travis-ci.com |
| Venafi | CodeSign Protect | Secure code signing operations | By request | venafi.com/codesign-protect |
| VMWare | Aria Operations for Applications | Multi-cloud environment observability | Trial period | vmware.com/products/aria-operations-for-applications.html |
| | Salt | Configuration management and distributed remote execution | Open source | saltproject.io |
| VSoft Technologies | Continua CI | Scalable continuous integration server | Free tier | finalbuilder.com/continua-ci |
| Weaveworks | Weave GitOps Core | Continuous delivery for Kubernetes | Open source | weave.works/product/gitops-core |
| Worksoft | Certify | Codeless continuous automated testing platform | By request | worksoft.com/products/worksoft-certify-test-automation |
| xMatters | xMatters | Service reliability platform | Trial period | xmatters.com |



DZone

600 Park Offices Drive, Suite 300
Research Triangle Park, NC 27709
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.