circleci

The Data-Driven Case for CI: What 30 Million Workflows Reveal About DevOps in Practice

Authored by Ron Powell and Michael Stahnke

EXECUTIVE SUMMARY

What does a high performing technology delivery team look like? How do you know if your team is doing well? While there have been many widely reported and shared surveys on technology delivery team behavior that define the metrics for high performers (**Puppet State of DevOps Report 2019**, **2019 Accelerate State of DevOps**), at CircleCI we are privileged to have the vantage point of being able to review truly massive amounts of data on how technology delivery teams are behaving in the wild. Our cloud continuous integration and continuous delivery platform processes over 1.6 million job runs per day for more than 40,000 organizations and over 150,000 projects. We analyzed the data from 30 million workflows* to see how observed behavior compares to reported industry standards.

Our results show that continuous integration provides a clear path to becoming a high performing team:

- Teams using CI are incredibly fast: 80% of all workflows finish in less than 10 minutes.
- Teams using CI stay in flow and keep work moving: 50% of all recovery happens in under an hour.
    - 25% recover in 10 minutes.
    - 50% of orgs recover in 1 try.
- Highly efficient CI tooling is in service to the needs of your business, not the other way around. Pipelines are flexible and changes to them are successful: 50% of projects never had a failure when pipeline changes were performed during the 90 days of observation.
- **If you are looking for a path to engineering success, focusing on CI will deliver results.**

Teams using CircleCI are delivering at high velocity, with high confidence. This is the definition of being a high performance dev team. **CI is confidence, automated.**

While the question of whether CircleCI enables teams to be high performing or high performing teams choose CircleCI is yet to be determined, we can clearly see that **teams who commit to the practice of CI can produce the best software, consistently, confidently, and at high velocity.**

We know that implementing a robust, widely adopted CI practice is not easy. Delivering a great software product is not easy. But our data has great news: the average team using CI can achieve results that put them in the highest ranks of technology delivery teams. Your team does not have to have the world's best DevOps practitioners to be one of the best dev teams. You don't have to deploy a thousand times a day or spend a million dollars a month on infrastructure. **Simply starting and committing to the practice of CI/CD is a reliable path to engineering success.**

** a workflow is a set of rules for defining a collection of jobs and their run order*

INTRODUCTION:

Every technology delivery team wants to increase their code delivery velocity and the quality of the code they push to market. The industry-standard “Four Key Metrics” of DevOps drive digital transformation for software development and delivery, and show that it is possible to optimize for stability without sacrificing speed. But what do these metrics tell us about how dev teams are operating in real life? How can we create actionable insight from the data to help make dev teams more successful?

We wanted to know. As the world’s largest standalone CI provider, we looked to data from 30 million workflows on our platform to see how teams were moving from commit to release in practice, at scale. We were curious to answer a few key questions:

- What does “fast” really look like?
- How does delivery vary across a wide spectrum of teams?
- How do observed performance from our data, reported survey performance, and industry assumptions about benchmarks align or differ?

Let’s start by covering a few key terms. Continuous integration and continuous delivery (CI/CD) are the foundation of software delivery. CI/CD defines the automated steps behind building, testing, and deploying software. It is the embodiment of Agile development: DevOps covers the team structure, the tooling, and the organizational culture for developing software in a time when constant change is the norm. CI/CD provides the ability to automate in minutes what was historically a process that required many manual approval steps, which could take months to cycle through.

The four key stats, lead time, deployment frequency, mean time to recovery (MTTR), and the change fail percentage, map to clearly observable CI/CD metrics:

- Lead time for changes —> workflow duration
- Deployment frequency —> how often you kick off a workflow
- MTTR —> the time it takes to get from red to green
- Change fail percentage —> workflow failure rate

In the relations above, a workflow is being used to describe a set of rules for defining a collection of jobs and their run order. Optimizing these four key metrics leads to tremendous advantage over organizations that have not yet begun to adopt continuous integration and delivery. The data that supports this is clear.

Research shows that high performing DevOps teams continue to outperform their organizational counterparts with: 200x more frequent deployments, 24x faster recovery from failure, a 3x lower change failure rate, and 2,555x shorter lead time*.

It is important to emphasize here that speed alone is not the goal. The combination of automation, removing manual tasks and people-delays from the process, and shifting testing left enable speed, **and quality**. **Consistency of delivery is the unsung hero of automation: speed without reliable, consistent quality is not helpful.**

To achieve control of these key metrics, one must master the corresponding CI/CD aspects of their development pipeline. Let's explore each one.

* ***Puppet 2016 State of DevOps Report***

Lead time

Lead time is the length of time that it takes for your workflow to run from first trigger to completion. It is a measure of how quickly you can get a signal. A short lead time requires a maximally automated software development pipeline. That is why teams use continuous integration to automate building and testing. Automation of the pipeline via CI shrinks deployment timelines of weeks to months down to hours, if not minutes, and is how teams using CI are able to reduce lead time by four orders of magnitude over their counterparts.

There are many types of workflows. One could imagine a workflow that deploys an artifact to a hosting service that is then picked up by a separate workflow. So what we are describing here by lead time is not always “deploy to production”, but getting to the desired end-state of your workflow; the workflow’s outcome. The lead time measured is based on how long it takes for those workflows to run.

The length of time that your workflow runs defines the feedback cycle that is at the heart of CI/CD. The expectation is not that high performing teams are able to produce code without error and that all code changes produce green builds. No team should fear a red build — these are part of the development process for teams of every skill level. Rather, teams need to be able to act on a failure as soon as possible and with as much information as they can get from the failure. Teams should feel that they earn green builds, not that they get them by default. A workflow without tests will run quickly and return green. That is not helpful to anyone. Without tests, a fast-running green workflow is not generating a useful signal to the engineer as to whether the code being introduced accomplishes its goals to the standards required.

If a super-fast workflow is bad, is a longer workflow better? Is there a workflow length teams should aim for? A workflow that takes 10s of hours might be representative of a test suite that is complex and comprehensive. And while that run time may be relatively efficient for what it accomplishes, it is still a long time for a team to wait for a useful signal. **Optimal workflows will run on the order of minutes to 10's of minutes**, allowing signal to reach developers often enough for them to push code changes throughout the day and apply fast fixes to any workflow that returns a fail.

Deployment frequency

How often your team kicks off a workflow is an indication of how many discrete units of work are moving through the development pipeline. With a fully automated software development pipeline, even complicated updates to your codebase can be deployed automatically. Hotfixes are available immediately — and they go through the same rigorous testing as any other update. The important aspect of this metric is that it serves as an indicator of a team's ability to deploy at will.

Confidence in your CI pipeline, and thus the ability to automate deployments, removes the burden of manually validating your code, clearing the path for frequent deploys. Testing is performed on every commit and failed build notifications allow for immediate visibility into failures. When you get error information returned to you as soon as possible, changes can be pushed faster, too, further upping deployment frequency.

Mean time to recovery (MTTR)

Mean time to recovery is contingent on getting actionable information, fast. Getting a failed status returned quickly will enable you to get from red to green in the shortest amount of time. Adopting CI/CD practices enables rapid feedback loops, and is the best way to ensure fast signal for your developers. A robust CI/CD practice includes realtime artifacts such as logs and coverage reports from your test suite and gives devs the ability to troubleshoot in an environment equivalent to the production environment.

Change fail percentage

The highest performing teams rarely push bad code to their default branch. But this doesn't mean that these teams never write faulty code. Testing and security checks are performed on a separate branch and only when everything passes is a merge allowed to take place. A team should know that their code works well before it is merged into the default branch. **Topic branches are where you want to get your fastest signal and where it is safe to fail.** This practice is a development strategy that originated in Vincent Driessen's Git-flow model. Change failure rate on topic branches is higher since this is where the majority of the work is being done, and because failure on these branches won't bring down the default branch. Failure on these branches only impacts the people working on the same branches versus the entire codebase/product.

We observed other teams using trunk-based development, another common development strategy where team members develop right onto the default branch. This strategy is optimized for recovery time — when a build is red, everyone works to recover. However, we did not see this method used nearly as often in the data.

WHAT DOES THE DATA TELL US?

The following section explores data from CircleCI from over 30 million workflows observed between June 1 and August 30, 2019.

It represents:

- 1.6 million jobs runs per day
- More than 40,000 orgs
- Over 150,000 projects

Lead time

Remember that lead time is the length of time that it takes for your workflow to run from first trigger to completion. The minimum recorded lead time in our data set was 2.1 seconds. What does a workflow that finishes in 2.1 seconds do? It couldn't do much. Some scenarios that would produce a 2.1 second lead time would be sending a notification, or printing a message and returning exit 0.

The maximum lead time is 3.3 days. This scenario likely represents a large amount of testing: test suites, regression suites, integration suites, spinning up external dependencies on databases on LDAP directories, running performance tests, and possibly cross-compiling for multiple platforms.

Eighty percent of the 30 million workflows examined finished in under 10 minutes. The 50th percentile of lead times was 3 minutes and 27 seconds; the 95th percentile was 28 minutes. There are teams that would consider 28 minutes too long to wait for a signal. Other teams consider getting a workflow to run at 28 minutes a great success. The lead time for your workflow is going to be very dependent on what type of software you are building, how deeply integrated your tests are, how complicated your tests are, and how well you are testing. At CircleCI we do not believe there is a universal standard for the ideal workflow length because of the variance in factors listed above. We believe teams should worry less about hitting a universal benchmark, and instead be looking inward for opportunities to reduce their workflow length. **Any optimization resulting in the reduction of workflow time is cause for celebration.**

This also doesn't imply that the orgs with three day+ workflows are universally slow. Organizations can have multiple projects, sometimes hundreds of projects, and the dependency structure of one project to another is not reflected in this data. The point is that while lead time can vary greatly between minutes to 10s of minutes for different reasons, this is a massive reduction in lead time when compared to teams that are not using CI/CD. CI/CD automates the majority of approvals that were once manual such as security and compliance, removing the wait for approval boards and other manual gates, and greatly reducing lead time.

Deployment frequency

In 2009, John Allspaw and Paul Hammond from Flickr gave a **talk** and shared that their team deploys 10 times per day. Since then, many teams have worked hard to outdo this number. As such, we expected to see many teams deploying with very high frequency. Instead, we did not see evidence of very many teams deploying tens of times per day or more. While this behavior does exist, it is the exception, not the rule. The reason for this disparity is likely that deployment frequency is, and should be, heavily dependent on the software your team is developing.

On CircleCI 50% of orgs start six workflows per day across all of their projects. At a per project level, 50% of projects have three workflows started per day, and 50% only start one workflow on the default branch per day. And at the highest end, the 95th percentile, these numbers rise to 39 workflows on the default branch, 74 workflows per project, and 250 workflows across all projects for an org.

Since an organization using CircleCI can deploy whenever required, the number of deployments becomes a reflection of an organization's choices, not necessarily a reflection of whether they are a high performing dev team. Instead, how frequently an organization tests and integrates their code and how quickly they recover when they encounter errors are more indicative of high performing dev teams.

Mean time to recovery

The minimum time to recovery on CircleCI was recorded at less than one second. The only way for this to happen is for two workflows to be started nearly simultaneously and for one to fail while the other passes. It isn't possible to get a signal and respond to that signal in that amount of time in any other scenario.

The maximum time recorded was 30 days. This does not imply that recovery is occurring in 30 days, but rather our data set cut recovery off at thirty days making this the maximum possible.

The median time to recovery on CircleCI is 17.5 hours. This is about the length of time between the end of one workday and the start of another. This implies that when engineers get a failing signal at the end of their day, they wait until the following day to resolve it.

The interesting data here is not the minimum time nor the maximum time, but the median time. This 17.5 hour median value goes against the conventional wisdom that is reflected in polls and surveys which says that mean time to recovery is much more rapid.

What is driving this disparity between reported and observed behavior? Our hypothesis is that survey design itself is influencing this speedy reported MTTR. Survey questions are often worded as, “for the primary application or service that you work on, tell me about ___?” Our data covers all workflows for all branches, not just the default branch and not just on the primary application. While the median observed time to recovery is 17.5 hours, it is likely that the mean time to recovery on your default branch of your primary application is much faster, on the order of minutes to hours. Our data says that over 30% of active project branches never failed over the ninety-day time period that was observed (100% green), meaning that MTTR is 0 seconds — since they never fail, they never need to recover.

Other data to note regarding MTTR on CircleCI is that **50% of all recovery happens in under one hour, and 25% of orgs recover in under 15 minutes**. Fifty percent of orgs recover in one try and 75% within two tries. The top 10% of performers spend less than 10 minutes doing the work necessary to fix their build and get back to green. Also of interest is that there is a large frequency gap in the data between three hours and the median time of 17.5 hours, suggesting that if a red build is not recovered in under three hours, it will likely not be fixed until the following day.

Change fail rate

Overall, 27% of all workflows on CircleCI fail. There are some that believe that each branch should always be green, but red builds are OK as long as you are able to recover quickly. A failing build is not a bad sign. It means that your CI system is working and is giving you valid data. Let's look at failure by branch type for some more insight into what's going on. Topic branches have an average failure rate of 31%. However, if we only look at default branches, the change failure rate is down to 18%. It is no surprise that the failure rate on default branches is lower. As mentioned above, over 30% of active project branches never fail. This average supports our belief that topic branches are where the majority of change sets are being committed and validated prior to mainline integration. When you merge to the default branch after extensive pre-work on topic branches, you have a better understanding of the change that your code is going to produce.

We were surprised to see that 50% of projects never had a failure when there were configuration changes to the circle.yml file orchestrating the CI/CD process. One possible explanation for this is that configuration is reused. For instance, an organization with many projects with the same configuration will update one and duplicate it once the changes pass testing. Another explanation is the use of CircleCI orbs. Orbs are reusable, shareable configuration packages that teams can use to add functionality without fear of failure, because they are tested by the author and validated by the community. Regardless of how this low rate of failure comes about, it is significant for CircleCI customers — it counters the widely held notion that configuration changes are hard and require frequent updates.

WHAT DOES THIS DATA MEAN?

This data shows that teams that are using CircleCI as a part of their CI/CD tooling are among the highest performing dev teams.

Aspect of software delivery performance	Elite*	Average CircleCI users
Deployment frequency —> workflow initiation frequency	On demand: multiple deploys per day	**6 workflows per day**
Lead time for changes —> workflow duration	Less than one day	**3.5 minutes**
Mean time to recovery	Less than one hour	**Less than one hour**
Change failure rate —> workflow failure rate	0–15%	**18% on the default branch**

**Elite metrics sourced from the 2019 Accelerate State of DevOps Report.*

But does this data reflect all teams who adopt CI/CD tooling? Or is it more acutely reflective of teams that have adopted the DevOps principles in the chart above, and employed them through the use of CircleCI?

There are specific features that CircleCI offers that make it easy for our customers to rank among the best and most efficient teams following DevOps practices. For instance, parallelization and intelligent test splitting help teams run builds concurrently on CircleCI and dramatically reduces their lead times. Other advanced CircleCI features, like Docker layer caching and other dependency caching lets teams load in large dependencies and artifacts once, rather than multiple times, resulting in faster builds and reduced lead times.

Other CircleCI features further help to reduce lead time and improve MTTR. Having the option to build on many different resource classes allows teams to configure CPU, GPU and RAM resources to reduce lead time. Also, because teams can SSH into the specific machine that their build failed on, they can easily troubleshoot a red build in the exact environment where the issue occurred, reducing time to recovery. Finally, having a dashboard that provides easy access to logs and test results further reduces time to recovery for teams building on CircleCI.

A popular CircleCI feature, orbs, allow you to update your configuration with community-and-partner-developed out-of-the-box solutions that are stable, and reliable. Orbs help teams reduce their change fail rate, and improve the reliability and consistency of their pipelines when they make changes to their config file.. This is a big draw for CircleCI users —these reusable snippets of code help automate repeated processes to speed up project setup and make it easy to integrate with third-party applications.

TO IMPROVE DELIVERY OUTCOMES, START WITH CI.

While there is a broad consensus that these DevOps efficiency metrics are important and drive positive outcomes for engineering teams, the question of how to achieve them was unresolved. However, our data should be encouraging for teams looking to improve engineering output. You do not have to be an expert, nor do you have to do everything perfectly to see benefits from adopting CI/CD principles. In fact, average CircleCI customers (50th percentile users) measure as high performing, just by adopting CI/CD principles.

But, we also know that a single tool or practice is not a silver bullet to team success. The increased failure rate that inevitably comes with bringing in new practices to a team can make it scary to initiate change across an organization. However, bringing CI/CD into an organization can be a catalyst for culture change, and in this report, we showed how CI/CD practices correlate with success.

These changes are not easy, engineering great software is not easy, and adopting a CI/CD tool doesn't happen overnight. But making improvements in these DevOps practices is how the best teams are able to produce the best software, consistently, confidently, and at high velocity. And while aspiring to be a high performing team does not specifically mean that you need to use CircleCI, we do know that teams that host their CI on our platform are leaders in their space.

What will we explore next?

This analysis uncovered additional areas for further exploration:

- We found some language-specific nuances that we would like to dive into in future research. For instance, the fail rate on the default branch for projects written in PHP is only 7% compared with an average of 18% for all default branches. Is it that PHP projects are running fewer tests (or no tests)? Is it that there are more engineers that are experts in PHP? We found additional interesting patterns for JavaScript, Python, Ruby, Go, and Java that we intend to look more deeply at in future research.
- We would also like to further explore whether the number of branches impacts success rates and velocity and if clear patterns emerge for branching relative to team size.
- Lastly, we want to look into how our customers are using manual approval steps in their configuration and how that affects a team's lead time and change failure rate.

Methodology

The data from CircleCI was collected over 90 days from June 1 to August 30, 2019, and represents 31 million data points, more than 40,000 orgs, and over 150,000 projects.