Modernize your CI/CD

GitLab

Continuous integration and delivery helps DevOps teams ship higher quality software, faster. But is all CI/CD created equal? What does successful CI/CD implementation look like and how do you know you're on the right track?

To understand how CI/CD makes organizations more successful, it helps to look at the DevOps challenges a comprehensive CI/CD solves and how it can improve business revenue long term. When organizations prioritize a good CI/CD strategy, it's a competitive edge that has long-lasting benefits across many teams. When looking into modernizing your CI/CD, consider these four criteria:

» DevOps challenges a comprehensive CI/CD should solve.
» The revenue impact of a poor or non-existent CI/CD strategy.
» Competitive benefits and how to measure success.
» How to find the right CI/CD solution to meet your needs.

Modernizing your architecture and embracing CI/CD is what successful companies are doing to release better software and increase development speed. When organizations implement CI/CD best practices, they get the added benefit of generating more revenue in the long run.

What challenges do I face?

1. Maintenance and integration costs, predominantly human resources costs.
A large percentage of the overall IT budget goes to support teams of engineers needed to integrate and maintain a complex toolchain. An enterprise company with 1,000 developers could need up to 40 engineers just to maintain the DevOps toolchain instead of allocating these resources towards delivering business value.

2. Development is slowed/blocked by the operations team.
The quintessential challenge of the pre-DevOps world is that dev teams are incentivized to increase innovation velocity by shipping new features. Operations teams are incentivized for stability, uptime, and error reduction. The higher the development velocity, the greater the chance for downtime and errors – so these teams are naturally at odds with each other. Dev leaders don't always have enough enticing evidence or incentive to go to the Ops team to advocate for increased deployment velocity, and vice versa.

3. Developers doing ops.

Today, teams and individual developers base the code they produce on the capabilities of their environment rather than the needs of the business.

What do these look like in practice?

A big portion of resources and budget goes to undifferentiated integration and maintenance.

Teams are siloed by their tools – each team has their favorite and is optimized to work within these specialized tools only. It is difficult to collaborate and troubleshoot across the stack due to a lack of visibility.

Code sometimes never gets to production at all.

There is a delay between code being written and driving value. When problems or errors arise and need to be sent back to the developer, it becomes difficult to troubleshoot because the code isn't fresh in their mind (context switching). They have to stop working on their current project and go back to the previous code to troubleshoot. So much time might have passed that the code is no longer deployable in its current state. In addition to wasting time and money, this is demoralizing for the developer who doesn't get to see the fruit of their labor.

Developers worry about environments, not business logic.

Environment dependencies and configuration distracts developers from tasks they're better equipped to handle. They may even be spending time trying to decide what size VM they need to deploy to. In this order "DevOps" means "Developers have to do both dev and ops." Only a small percentage of developers actually enjoy this arrangement with most asking, "I'm a developer, please stop asking me to do operations."

What are the business impacts of poor CI/CD?

1. A large portion of IT budget is spent on undifferentiated engineering

Opportunity costs play a much larger role in the development process than we realize. Organizations can only afford so many engineers at one time, and systems that require extensive maintenance means fewer engineers are working on revenue-generating projects. This will lead to slower innovation and slower growth in the long term. Undifferentiated engineering means too many individuals are having to focus on one thing – maintenance.

2. Delayed (and even unrealized) revenue

This is the impact of lost opportunity costs. When there are too many dependencies, too many handoffs, and too many manual tasks, it causes delays between when code is written and when the business gets value from that code. In worst cases, code is written and the business never gets any value from it at all. Code can sit in limbo waiting for others to manually test it, and by the time it's finally reviewed it's already irrelevant. The opportunity cost essentially doubles: Engineers were paid to work on code that never deployed, and the business loses out on revenue the code could have generated.

3. Lower developer productivity, lower developer happiness, and less reliable software

Downtime = lost revenue. To avoid that dreaded downtime, developers are spending time working on infrastructure and configuration, and they're also not spending that time delivering business logic. In both cases, they're being less productive and working outside of their core competencies. Developer hiring and retention will inevitably suffer. Uptime and resiliency are also affected because people who aren't domain experts are put in charge of determining infrastructure. It's a self-fulfilling prophecy.

What does it look like if a magic wand were to solve it today?

1. More engineers are working on the app instead of maintenance

The organization has the right amount of developers devoted to driving business value and spends more time on innovation instead of undifferentiated heavy lifting. Less of the budget is spent on activities that don't generate revenue.

2. Developers see their code in production quickly

Infrastructure and deployment are fully automated. Everyone loves to see the output of their work, developers especially, and the business gets to see the benefits of this code right away. Deploying smaller chunks of code is less risky when developers can take advantage of test automation, so they have less overhead and coordination with a QA team forced to test manually.

3. Developers are focused on solving business problems

Code is written to be environment and cloud agnostic. Development teams own the uptime of their own services, but they are fully supported by the ops team. Ops owns the infrastructure, dev owns the service, and both teams can work according to their strengths.

What are some of the benefits of a good CI/CD strategy?

1. Increased speed of innovation and ability to compete in the marketplace

Two identical companies: One implements CI/CD technology and the other doesn't. Who do you think deploys applications faster? While this seems like a silly comparison, because *of course* the company with more automation deploys faster, there are organizations out there still convinced they don't need CI/CD because they're not looking at their competition. Organizations that understand the importance of CI/CD are setting the pace of innovation for everyone else.

2. Code in production is making money instead of sitting in a queue waiting to be deployed

Organizations that have implemented CI/CD are making revenue on the features they deploy, not waiting for a manual check to see if the code is up to par. They already know the code is good because they have tests that are automated, and continuous delivery means that code can be deployed at any time. They've removed human error and delays from the process.

3. Ability to attract and retain talent

Engineers that can focus on what they're best at will be happier and more productive, and that has a far-reaching impact. Turnover can be expensive and disruptive. A good CI/CD strategy means engineers can work on important projects and not worry about time-consuming manual tasks. They can also work confidently knowing that errors are caught automatically, not right before deployment. This kind of cooperative engineering culture inevitably attracts talent.

4. Higher quality code and operations due to specialization

Dev can focus on dev. Ops can focus on ops. Bad code rarely makes it to production because testing is automated. Developers can focus on the code rather than the production environment, and operations doesn't have to feel like a gatekeeper or a barrier. Both teams can work to their strengths, and automated handoffs make for seamless processes. This kind of cooperation makes DevOps possible.

What capabilities are required to make this happen?

1. Robust CI/CD

When we use the term "robust," it's all about avoiding half-baked or partial solutions. There are several CI/CD solutions out there but there are varying degrees of effectiveness. Continuous integration and continuous delivery go hand in hand, so having a solution that offers both is ideal. The tool you use should offer the automation you need, not just some. If your CI/CD tool is prone to failure or "brittle," it can be just one more thing to manage. This was precisely why the team at Ticketmaster replaced Jenkins CI and moved to weekly releases, decreasing their pipeline execution time from two hours to only *eight minutes* to build, test, and publish artifacts.

2. Containers and Kubernetes

Containers have made a huge impact on the way companies build and deploy code. While it was once difficult to develop applications with a microservices architecture, over the past five years it has become considerably easier with container orchestration tools like Kubernetes, comprehensive CI/CD tools that automate testing and deployments, and APIs that update automatically. Breaking up services so they can run independently reduces dependencies and creates better workflows.

3. Functionality for the entire DevOps lifecycle

Visibility is a huge asset when improving DevOps workflows. For some teams, they can have several tools handling different facets of the SDLC, which creates integration issues, maintenance issues, visibility issues, and is just plain expensive from a cost standpoint. A complex toolchain can also weaken security. In a Forrester survey of IT professionals, 45% said that they had difficulty ensuring security across the toolchain.

How would you measure success?

1. Cycle time

Cycle time is the speed at which a DevOps team can deliver a functional application, from the moment work begins to when it is providing value to an end user. See how the team at Axway was able to achieve a 26x faster DevOps cycle with GitLab.

2. Time to value

Once code is written, how long before it's released? This delay from when code is written to running in production is the time to value, and is a bottleneck for many organizations. Continuous delivery as well as examining trends in the QA process can help to overcome this barrier to quick deployments.

3. Uptime, error rate, infrastructure costs

Uptime is one of the biggest priorities for the ops team, and with a good CI/CD strategy that automates different processes, they should be able to focus more on that goal. Likewise, error rates and infrastructure costs can be easily measured once CI/CD is put in place. Operations goals are a key indicator of process success.

4. Team retention rate

Happy developers stick around, so looking at retention rates is a reliable way to gauge how well new

processes and applications are working for the team. It might be tough for developers to speak up if they don't like how things are going, but looking at retention rates can be one step in identifying potential problems.

The benefits of a good CI/CD strategy are felt throughout an organization: From HR to operations, teams work better and achieve goals. In such a competitive development landscape, having the right CI/CD in place gives any company an edge.

What is my CI/CD budget?

1. Free vs. Paid

Open source software is an incredible thing: Not only is it a great way to learn new skills, but its collaborative nature lets developers improve and support products they love. Many organizations have adopted open source software for good reason. Open source benefits from a thriving community that contributes new ideas, and creative minds solve problems creatively. Open source innovates, and enterprises get to take advantage of these efforts for free.

While no one can beat the low, low cost of "free" (or at least free in most cases), it's important to consider more than just cost.

Paid software does have cost attached to it, but it comes with distinct advantages. For one, you will receive better support with paid software, and higher-tier pricing models even have their own dedicated support team. When you pay for a service, you have the right to tell a provider, "I'm having trouble with this and I need your help to fix it." In the realm of CI/CD, where configuration plays such a big role, this kind of support pays for itself.

If a free product has everything a team needs, that's great. After all, GitLab Community Edition also offers a complete DevOps lifecycle with CI/CD built in, but there may come a time when an organization has to ask themselves: When is paying for a service the better decision in the long run?

2. Cost/benefit analysis

When evaluating a CI/CD solution, it's important to measure your organization's current needs vs. expected needs. All organizations have some sort of growth plan or expected growth trajectory and goals to go with them, such as headcount goals, expansion plans, additional products or services,

etc. Factoring in costs and benefits, investing in CI/CD has the potential to help you hit those numbers faster.

Room for growth
Will free software give you the room to grow or will it eventually limit you? Will you have the CI pipeline minutes you need for increased output?

Better code quality
Will you be able to produce better quality code and reduce code vulnerabilities?

Increased efficiency
Will you be able to reduce manual tasks and improve operational efficiency?

Weighing these factors, the cost/benefit analysis is largely positive when it comes to paying for CI/CD. Higher-cost plans may be able to offer additional security functionality, support for Kubernetes, additional pipeline minutes, and other perks that can help you maximize your CI/CD. When it comes to modernizing applications later, it can be a lot more expensive the bigger you are. Adopting technologies early, when teams are more nimble, is a much easier and cheaper endeavor.

A dollar isn't always a dollar, and sometimes the long term benefits far outweigh the cost of additional features. It's important to analyze your CI/CD budget and identify areas for revenue-generating opportunities.

How does GitLab CI/CD compare to other tools?

GitLab is the only single application for the entire DevOps lifecycle, with CI/CD already built right in. GitLab allows Product, Development, QA, Security, and Operations teams to work concurrently in a single application, allowing for maximum visibility and comprehensive governance. There's no need to integrate and synchronize outside tools as part of a large, complicated toolchain.

GitLab CI/CD is designed for a seamless experience across the SDLC, and while there are other CI/CD solutions out there, we're the only application with everything from source code management to

monitoring built in. Teams can collaborate in one environment.

Jenkins vs. GitLab

Jenkins is one of the most popular self-managed, open source build automation and CI/CD developer tools. It uses hundreds of available plugins, enabling it to support building, deploying and automating projects.

Weaknesses

Plugins are expensive to maintain, secure, and upgrade.

Compare Jenkins vs. GitLab

Travis vs. GitLab

Travis CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. Travis CI also offers a self-hosted version called Travis CI Enterprise, which requires either a GitHub Enterprise installation or account on GitHub.com.

Weaknesses

No continuous delivery and GitHub hosting only.

Compare Travis vs. GitLab

Bamboo vs. GitLab

Bamboo Server is a CI/CD solution, part of the Atlassian suite of developer tools. It's only available in a self-managed configuration and source code is available only to paid customers. Bamboo uses Bitbucket for understanding how source code has changed (SCM), as well as integrations with other tools.

Weaknesses

Those interested in auto-scaling must use Amazon Elastic Compute Cloud (EC2) and pay Amazon separately for their usage.

Compare Bamboo vs. GitLab

What the analysts say

The Forrester CI Wave™: Continuous Integration Tools report is a comprehensive evaluation of 10 CI tools based on 16 criteria, including ease of installation, security features, analytics, and more. Out of the 10 tools reviewed, we were rated #1 by Forrester and received the highest marks in several areas such as Current Offering as well as ease of installation/configuration.

> **"GitLab delivers ease of use, scalability, integration, and innovation... GitLab supports development teams with a well-documented installation and configuration processes, an easy-to-follow UI, and a flexible per-seat pricing model that supports self-service. GitLab's vision is to serve enterprise-scale, integrated software development teams that want to spend more time writing code and less time maintaining their toolchain. The company actively engages in the DevOps community, hosting regular webcasts and sponsoring developer events. It also stands apart from most other vendors in the space by making its planned enhancements road map available to the community through a public issue tracker."**
>
> *— THE FORRESTER WAVE™: CONTINUOUS INTEGRATION TOOLS, Q3 2017*

Having the right CI/CD in place is a competitive advantage in the current development landscape. Teams that utilize the right CI/CD strategy are going to produce better quality software much faster, and they're going to free up valuable resources to focus on long-term growth and innovation. When choosing the right solution for you, here are the top things to consider:

» **What is the cost vs. benefit?** A revenue-generating expense is not a dollar-for-dollar scenario. When it comes to budget considerations, it's important to look at the big picture and discuss value as well as cost. If you're paying the lowest price but not getting everything you need for scale, you're paying too much.

» **What are customers saying?** Word of mouth is a powerful tool. If you're interested in a particular CI/CD platform, learn about their customers and see what they have to say. Read case studies and look for customers with similar problems to yours so you can see how they solved them.

» **What do industry experts think?** Happy customers won't always point out shortcomings but industry experts can provide the vendor-neutral perspective you need to make an informed decision. Read reports and industry publications to learn how experts evaluate one CI/CD platform from another.

» **What do you think?** Independent research is an important last step. You've seen what customers think, you've seen what experts in the field say, and now it's time to form your own opinion. Join webinars to learn more about a product and ask questions, and compare product features carefully.

Start your free GitLab trial

About GitLab

GitLab is a complete DevOps platform, delivered as a single application. Only GitLab enables Concurrent DevOps, unlocking organizations from the constraints of today's toolchain. GitLab provides unmatched visibility, radical new levels of efficiency and comprehensive governance to significantly compress the time between planning a change and monitoring its effect. This makes the software lifecycle 200% faster, radically improving the speed of business.

GitLab and Concurrent DevOps collapse cycle times by driving higher efficiency across all stages of the software development lifecycle. For the first time, Product, Development, QA, Security, and Operations teams can work concurrently in a single application. There's no need to integrate and synchronize tools, or waste time waiting for handoffs. Everyone contributes to a single conversation, instead of managing multiple threads across disparate tools. And only GitLab gives teams complete visibility across the lifecycle with a single, trusted source of data to simplify troubleshooting and drive accountability. All activity is governed by consistent controls, making security and compliance first-class citizens instead of an afterthought.

Built on Open Source, GitLab leverages the community contributions of thousands of developers and millions of users to continuously deliver new DevOps innovations. More than 100,000 organizations, including Ticketmaster, ING, NASDAQ, Alibaba, Sony, and Intel trust GitLab to deliver great software at new speeds.


GitLab