GitLab

How to convince leadership to adopt CI/CD

A developer’s guide to advocating for new technology

Table of Contents

03 INTRODUCTION

03 HOW WE DEFINE CI/CD

05 THE VALUE OF CI/CD FOR DEVELOPERS
- More reliable deployments
- Faster feedback loops
- More time for creative projects
- Less context switching

06 MAKING THE CASE FOR CI/CD IN YOUR ORGANIZATION
- Communicate value
- Start small
- Measure the results
- Tie your efforts to business outcomes
- Use industry examples

10 OVERCOMING BLOCKERS TO CI/CD ADOPTION
- Legacy infrastructure
- Not enough time
- Lack of expertise
- Not enough budget
- Lack of visibility

11 CONCLUSION

12 ABOUT GITLAB

Introduction

Continuous integration and continuous delivery (CI/CD) are a foundational component of modern software development. Though most organizations use some form of CI/CD, many are still using slow, error-prone, manual processes to build, test, and deploy their software. What if you could get your code to production simply by merging to master while automation takes care of the rest?

Teams look to tools and other solutions to help them achieve this ideal state of development. Developers often have their hand on the pulse of new technology, but IT leaders and executives need to be convinced that making a large investment in updating tools, training, and culture will have a positive return.

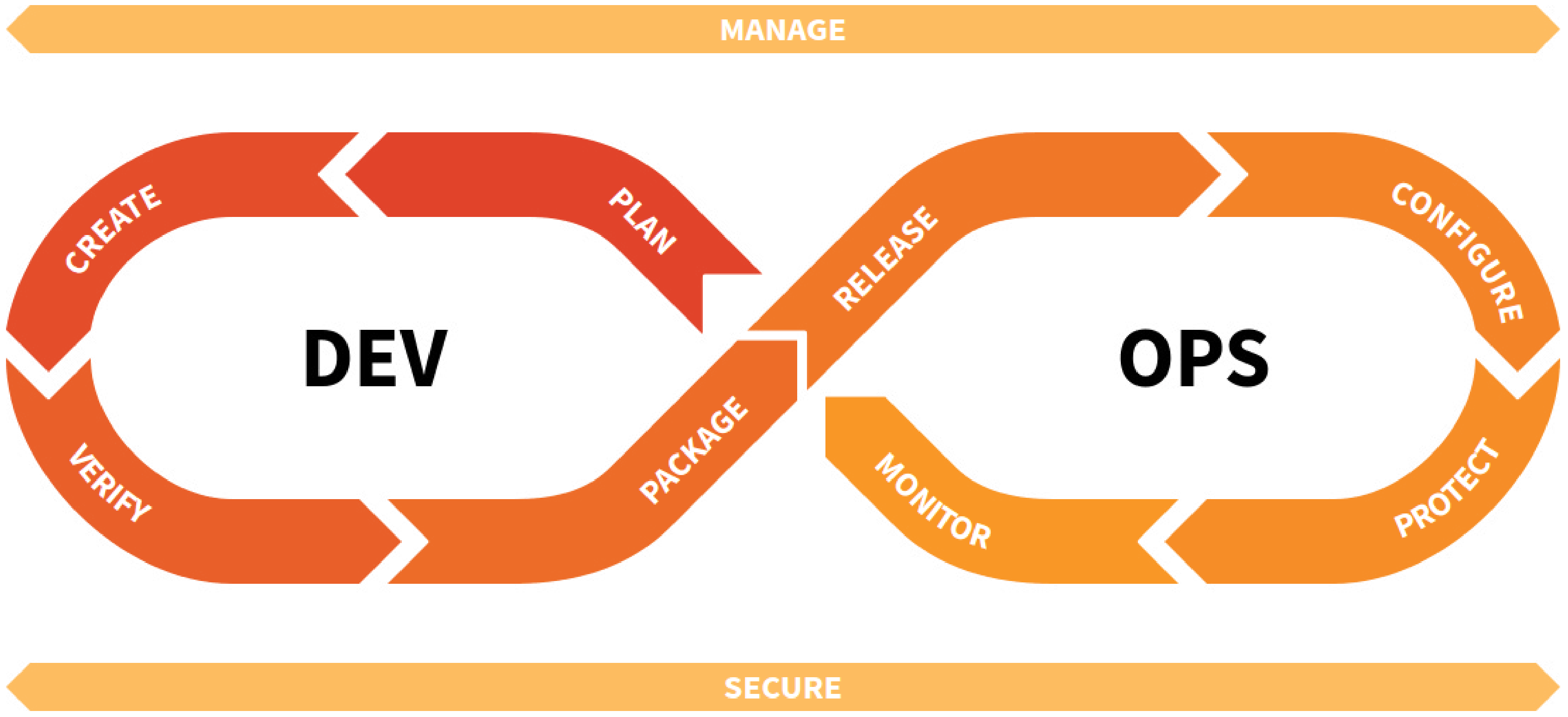
Learning how to advocate for budget and organizational change can be a beneficial skill. In this eBook, we'll show helpful strategies in making the case for CI/CD in your organization and how to overcome blockers to adopting CI/CD best practices. We'll also discuss five tips to successfully advocate for new software to leadership teams.

How we define CI/CD

We'll first define CI/CD to make sure we're staying consistent. When it comes to CI/CD, there isn't much variance in the industry, but there are some nuances to consider. At GitLab, when we're talking about CI/CD we are talking about stages of the DevOps lifecycle. These stages are **Plan**, **Create**, **Verify**, **Package**, **Release**, **Configure**, **Protect**, and **Monitor**. On either side of these stages are ongoing processes: **Manage** and **Secure**.

Software moves continually through these stages in the DevOps infinity loop. For GitLab, CI and CD live in the **Verify**, **Package**, and **Release** phases of the DevOps lifecycle. When we think about continuous integration, this is performing all of your code tests, unit tests, and integration tests. Martin Fowler and others who are early thought leaders on these topics have additional criteria, such as merging to your main branch or main trunk at least once a day. Sometimes we use the term CI just to talk about the tests we run, and that test might just be on a feature branch or smaller part of the code, but we can still call that CI. If we're building software and testing that build, that's the verification phase.

We also include packaging when talking about CI/CD. Not only are you building your release artifacts, you're also storing them and doing package management. Every team needs a place to store their packages and dependencies. GitLab allows teams to package their applications and dependencies, manage



containers, and build artifacts all from our single interface. If you're doing cloud native development or containerized applications, there are also container registries and artifact repositories built-in.

Finally, CD is in the **Release** phase of the lifecycle. Teams can release into any number of environments such as development environments, QA or testing environments, staging environments, and, of course, simply releasing software into production.

For many teams, they may not be doing continuous delivery yet but are automating some tests. Some may deploy to a staging environment multiple times a day, but only deploy to production once a month. Maybe there are still manual tests with some automation incorporated into build and test phases. What CI and CD looks like from team to team can vary, and that’s okay.

The value of CI/CD for developers

If you're in the engineering organization, you want to remove the tedious and error-prone manual effort that makes your job harder. If you're doing manual builds, manual tests, manual deployments, it’s not necessarily fun work and can involve a lot of troubleshooting and difficulty. Manual processes make releases more risky, prevent you from getting feedback on your code, and are often to blame for those late nights when a deployment imploded on itself and you had to fix it. CI/CD alleviates those all too common pain points.

MORE RELIABLE DEPLOYMENTS

If you have one big release every six months, there is a lot of risk involved in that release. If one thing goes wrong, it can break everything. With so many changes going live at once, it's very hard to do forensics on what broke and why.

When you have robust CI/CD practices, you can continually deploy to production because it's automated. You can make small changes with low risk and potentially deploy hundreds or thousands of times per day. If something breaks, it's a lot easier to know where it happened in the deployment, and to rollback or to remove that part of the code.

FASTER FEEDBACK LOOPS

For engineers who write code and build software, rapid feedback on code can be especially valuable. Few things are more frustrating or demoralizing than spending a lot of time and effort writing software, only for no one to use it. CI/CD allows for faster feedback loops so developers can make small changes, learn what's working and what's not working, and use that knowledge to make the right adjustments.

MORE TIME FOR CREATIVE PROJECTS

When builds, tests, and deployments are automated, engineers can focus on more creative work. Instead of working late on a Friday night fixing broken builds, developers can be proactive and work on projects that excite them and add long term value to the software they create. When deployments are boring and predictable, developers are happier because they can work on projects that not only excite them, but are more in line with why they became developers in the first place.

LESS CONTEXT SWITCHING

When writing code in one part, one feature, or one section of your codebase, you’re immersed in that environment. Once that project gets put on the release calendar, you have to clear your mind and work on a new project. If you discover there's a problem with this code, you have to get back into that mindset and into the framework of what you built in order to fix it, and that takes time and effort.

Constant context switching is frustrating for engineers. For example, if you ship something to your security team, but they don’t get back to you until a week or even a month later, you basically have to relearn all of the code. It's unproductive and ineffective.

Making the case for CI/CD in your organization

CI/CD provides tremendous value to engineers, but often if the engineering team wants a widespread implementation, they need to convince leadership that CI/CD is the way to go. These five tips for selling your organization on CI/CD have come from real-world experiences, influenced by conversations with GitLab customers, engineers, IT leaders, and business leaders.

What engineers will need to focus on in these discussions is the value CI/CD can bring to others: The value for leadership and for the business as a whole. By changing the scope of the conversation to the opportunities of CI/CD, rather than the pain points, engineers can drive these discussions and advocate for new technologies successfully.

COMMUNICATE VALUE

In GitLab's 2020 Global DevSecOps Survey, a majority of those surveyed (dev, sec, ops, and test pros) agreed that communication and collaboration will be the most important skills for their future careers. Given the increasing prevalence of cross-functional tech teams, the significance of soft skills like communication and collaboration makes sense. It's vital for teams to communicate well, particularly when business and tech pros work together.

When trying to persuade an organization on a new piece of technology, you'll need to communicate value, not only for your team, but for the organization as a whole. Identify stakeholders in the decision-making process and dive into their motivations and goals to find the most compelling way to communicate value to them.

In engineering orgs, it can be very easy to go to leadership and talk about late nights, the manual effort, tedious tasks, the chaotic environment, but all this doesn't communicate value. Instead of taking that approach, think about what leadership is concerned about. Maybe they care about driving revenue or lowering costs, or maybe they're trying to find a way to retain talent. Phrase your argument in a way that solves a problem for them. Luckily, a good CI/CD strategy can tackle those larger issues head-on.

Whether it's your leadership or your colleagues who need to understand CI/CD's value, try to put it in their words. For example, you can say that if the organization had more robust engineering practices it would be able to better attract and retain talent. Communicate the benefits of CI/CD, not just for your team, but for leadership as well.

START SMALL

The reason you would want CI/CD in your engineering org is so that you can break your code changes into small parts that are easier to manage. Smaller code changes are less risky, easier to prove right or wrong, and are easier to rollback if necessary. You can apply these same development principles to advocating for change in your organization.

When implementing CI/CD, you don't need to completely change your infrastructure to move things forward. Starting small with one team or one project is often the most successful way to implement change. If just one team adopts a new or better version of CI/CD, that team can become what we call "the lighthouse" because it sheds light for the rest of the org.

A historically low-cost endeavor is to propose an experiment. This will get your foot in the door, and will also let you measure the results for an even better argument for adopting CI/CD. By proposing a small change (just one team/initiative/project), and using an experiment to measure the results, you're making CI/CD a low-risk request for leadership.

MEASURE THE RESULTS

While appealing to motivations and goals is an important part of meaningful communication, you want to be able to prove your arguments. Successful data points around the benefits of CI/CD will make a strong case.

A powerful metric for leadership can be Mean Time to Repair (MTTR). For example, what's the average time from when there's a bug/failure discovered to when a fix is in place? You don't necessarily need a dashboard down to the millisecond, or robust observability tools with a graph that computes the MTTR for you. If you haven't adopted CI/CD yet, obtaining specific metrics might be a challenge. Use the best estimates available and ask these questions:

- When was the last time you had a failure in production?
- How long did it take to discover the problem?
- How long did it take to do forensics on the issue and develop a fix?
- How many customers/users were impacted by this problem?

If you have estimates for these questions, the lighthouse team, or the lighthouse project, can help you start to measure the new MTTR with CI/CD. For instance, if customers in your original MTTR were impacted by an outage for 24 hours, but with CI/CD that outage is only 30 minutes, that measurement can be very powerful.

If security and compliance, code coverage, and code quality are concerns for your organization, measure how much of the code is tested and when. If security testing happens when code is ready for production, it can be difficult to fix problems quickly. Shift left testing is an approach that moves testing to earlier in the development lifecycle. Use the lighthouse project to demonstrate how CI/CD can help the organization shift left with earlier security testing.

You don't need to have data for the sake of having data. Pick two or three data points that are powerful and relevant to your org.

TIE YOUR EFFORTS TO BUSINESS OUTCOMES

As an engineer, you may or may not be thinking about revenue on a daily basis. But, your leadership is. If you want to drive your argument for why you should adopt new technology in your organization, a great way to do that is to show how what you're doing actually helps a business drive revenue. If users can get features faster, then delivering more software sooner through more frequent deployments will push revenue forward.

Businesses are concerned with security and compliance. When deployments are boring, it reduces security and compliance risks. Automation reduces the chance of human error and CI ensures that code is tested continually.

For most organizations, it is well understood that engineering talent is very difficult to attract and retain. Having better engineering practices can help with that. For organizations that want to be competitive and hire talent, being able to run modern practices like CI/CD and even Progressive Delivery is a way to retain and attract talent, and that makes an impact on the business's bottom line.

The business is also going to care about driving revenue, driving market share, and acquiring customers. As engineers, if you care a little bit about these kinds of things, like revenue and customer acquisition, then use that to drive your engineering initiatives forward.

USE INDUSTRY EXAMPLES

Sometimes you can get very insular thinking about yourself, your challenges, and your organization's problems. Looking to the industry or at other organizations can be effective for more motivation.

DevOps Research and Assessment (DORA) is an organization that does quantitative research on the impact of DevOps on organizations, and they have correlated DevOps best practices with business success across enterprises. Every year, DORA releases an Accelerate State of DevOps report with data points you can share.

DORA has shown that four metrics correlate directly to business success.

- **Lead time:** Early feedback and build/test automation help decrease the time it takes to go from code committed to code successfully running in production.
- **Deployment frequency:** Automated tests and builds are a prerequisite to automated deploy.
- **Time to restore service:** Automated pipelines enable fixes to be deployed to production faster reducing Mean Time to Repair (MTTR).
- **Change failure rate:** Early automated testing greatly reduced the number of defects that make their way out to production.

Overcoming blockers to CI/CD adoption

When selling CI/CD to your organization, it's important to keep potential blockers in mind and learn how to show value even when arguments block your path. Just as CI/CD alleviates pain points in engineering, use the five tips above to show how CI/CD can break through potential barriers from leadership.

LEGACY INFRASTRUCTURE

Sometimes the problem with adopting great CI/CD is that the business may not think they have the infrastructure to support it yet. The argument is that before you can even start implementing CI/CD tooling and best practices, you'll need to address how you're doing infrastructure and the technical prerequisites first.

FASTER FEEDBACK LOOPS

Redirecting resources and engineers away from daily tasks can have an impact on already delayed release cycles. Organizations don't want to stop shipping features to implement continuous integration and may not think they have the time to adopt it.

LACK OF EXPERTISE

Lack of expertise can be a "chicken or the egg" argument: If we're not running a modern technology stack, it can be difficult to attract talent with modern technology skills. But if you don't have the right talent in place, how do you implement the right technology?

NOT ENOUGH BUDGET

Sometimes budgets involve not only the budget for the tooling itself, but the time to train and get people up to speed. Even if the technology can provide long revenue-generating opportunities in the future, sometimes the cost in the short term seems too big to overcome.

LACK OF VISIBILITY

Sometimes there's a chaotic development environment, but the business doesn't necessarily see that every day. There can be a lack of communication across the org, and that lack of visibility can be a blocker because if they don't know where the problems are, then there's no urgency to address them.

If there is a blocker, keep in mind that it’s only a blocker for now. Any business that cares about their customers, that cares about driving market share, will eventually want to adopt the best practices that help them succeed. While you might have trouble convincing them this quarter, persistence is one of the keys to successful advocacy.

Conclusion

Sometimes it can take a long time to build a case, sometimes you need to run experiments and build up data over time, but that only makes your case stronger the next time you present it. Leadership understands that new technology is coming along all the time, and making developers happy usually means higher output.

CI/CD has powerful benefits for engineering orgs, but also provides overall business benefits. Organizations with robust CI/CD are innovating faster and setting the standards for everyone else. As developers and engineers that work closely with this technology, you can be powerful agents of change in your organization. With these tips for success, you can advocate for new technologies and make long-lasting improvements across the entire organization.

Curious and want to explore GitLab’s industry-leading CI/CD?

Try GitLab free for 30 days

*This material was originally presented as a webcast entitled “Making the case for CI/CD in your organization.” In this iteration, we’ve included links to additional resources and have expanded sections to include more details. You can view the original webcast **here**.*

About GitLab

GitLab is a DevOps platform built from the ground up as a single application for all stages of the DevOps lifecycle enabling Product, Development, QA, Security, and Operations teams to work concurrently on the same project. GitLab provides a single data store, one user interface, and one permission model across the DevOps lifecycle. This allows teams to significantly reduce cycle time through more efficient collaboration and enhanced focus. 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 from startups to global enterprises, including Ticketmaster, Jaguar Land Rover, NASDAQ, Dish Network, and Comcast trust GitLab to deliver great software faster. GitLab is the world's largest all-remote company, with more than 1,300 team members in more than 65 countries and regions.

GitLab

GitLab