GitLab

Ultimate Guide to CI/CD

From fundamentals to integrating security tests and AI

Table of Contents

Introduction

Continuous integration/continuous delivery (CI/CD) has revolutionized how software teams work together to turn code into applications. Gone are the days of code integration headaches and repeated manual processes. CI/CD makes modern software development possible — fast, reliable, and automated.

At its core, CI/CD is about creating a seamless pipeline that takes code from a developer's environment all the way to production and incorporates feedback in real time. CI helps teams catch issues early — before they become costly problems — by ensuring that code changes are frequently merged into a shared repository, automatically tested, and validated. CD extends this by automating deployments, making releases predictable and stress-free.

Rather than relying on manual processes and complex toolchains for software development, teams can use a robust CI/CD pipeline to build, test, and deploy software. And AI can streamline the process even further, automatically engineering CI/CD pipelines for consistent quality, compliance, and security checks.

The result is higher quality software, faster delivery times, and more frequent releases that can quickly respond to user needs and market demands.

This guide explains modern CI/CD pipelines, from basic principles to best practices to advanced strategies. What you learn in this guide will help you scale your DevSecOps environment to develop and deliver software in an agile, automated, and efficient manner.

With continuous integration, teams can identify and fix errors and security issues more easily and much earlier in the development process.

CI/CD fundamentals

What is continuous integration?

Continuous integration (CI) is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing changes when you commit or merge them, and automatically kicking off a build. With continuous integration, teams can identify and fix errors and security issues more easily and much earlier in the development process.

What is continuous delivery?

Continuous delivery (CD) – sometimes called continuous deployment – enables organizations to deploy their applications automatically, allowing more time for developers to focus on monitoring deployment status and ensure success. With continuous delivery, DevSecOps teams set the criteria for code releases ahead of time. When those criteria are met and validated, the code is deployed into the production environment. This allows organizations to be more nimble and get new features into the hands of users faster — and know their deployments passed all their security tests.

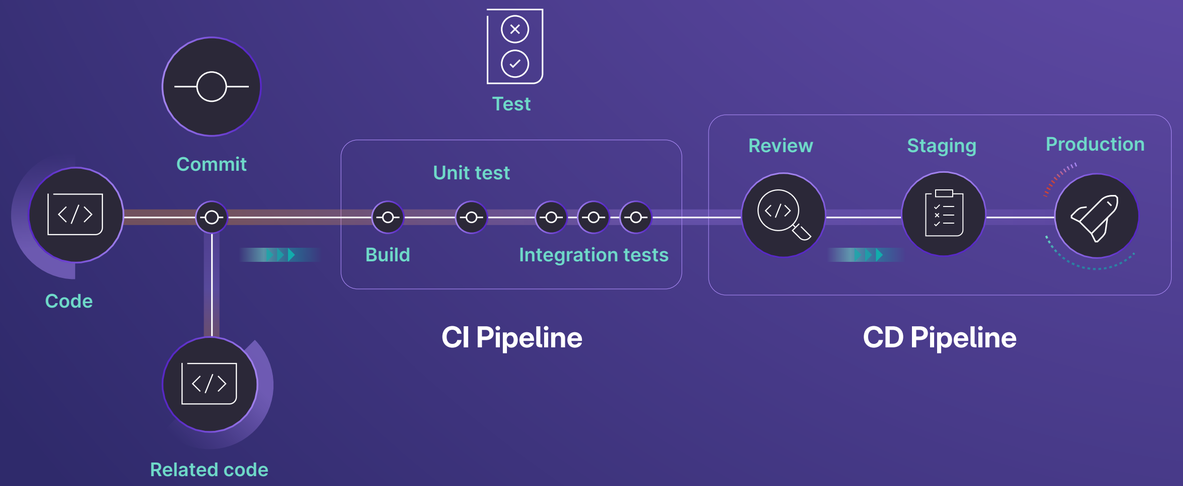
What is a CI/CD pipeline?

A CI/CD pipeline is a series of steps, such as build, test, and deploy, that automate and streamline the software delivery process. Each stage serves as a quality gate, ensuring that only validated code moves forward. Early stages typically handle basic checks like compilation and unit testing, while later stages may include integration testing, performance testing, compliance testing, and staged deployments to various environments.

The pipeline can be configured to require manual approvals at critical points, such as before deploying to production, while automating routine tasks and providing quick feedback to developers about the health of their changes. This structured approach ensures consistency, reduces human error, and provides a clear audit trail of how code changes move from development to production. Modern pipelines are often implemented as code, allowing them to be version controlled, tested, and maintained just like application code.

These are other terms associated with CI/CD that are important to know:

- **Commit:** A code change.
- **Job:** Instructions a runner has to execute.
- **Runner:** An agent or server that executes each job individually that can spin up or down as needed.
- **Stages:** A keyword that defines certain job stages, such as “build” and “deploy.” Jobs of the same stage are executed in parallel. Pipelines are configured using a version-controlled YAML file, .gitlab-ci.yml, at the root level of a project.



The benefits of CI/CD in modern software development

CI/CD brings transformative benefits to modern software development by dramatically reducing the time and risk associated with delivering new features and fixes. The continuous feedback loop gives DevSecOps teams confidence that their changes are automatically validated against the entire codebase.

Perhaps most importantly, CI/CD fosters a culture of collaboration and transparency within software development teams. When everyone can see the status of builds, tests, and deployments in real time, it becomes easier to identify and resolve bottlenecks in the delivery process. The automation provided by CI/CD also reduces the cognitive load on developers, freeing them to focus on writing code rather than managing manual deployment processes. This leads to improved developer satisfaction and productivity, while also reducing the risk traditionally associated with the entire software release process. Teams can experiment more freely knowing rapid code reviews are part of the process and they can quickly roll back changes if needed, which encourages innovation and continuous improvement.

Here are some of the key benefits of CI/CD:

More frequent deployments. Teams can safely deploy smaller changes more frequently, reducing the risk and complexity associated with large, infrequent releases. This enables faster feedback cycles and more responsive development.

Faster time to market. Automated build and deployment processes enable rapid delivery of new features and bug fixes to customers, significantly reducing the time between writing code and releasing it to production.

Faster testing. By automatically running tests on code changes and executing multiple test suites simultaneously across different environments, CI/CD significantly reduces testing time compared to sequential or manual approaches.

Improved code quality. With automated testing throughout the development process, bugs are caught as they arise and rolled back without ever making it into the main branch. This ensures better code quality overall and that every release works just as intended.

Faster recovery. CI/CD makes it easier to fix issues and recover from incidents, reducing mean time to resolution (MTTR). Continuous deployment practices mean frequent small software updates so when bugs appear, it's easier to pin them down. Developers have the option of fixing bugs quickly or rolling back the change so that the customer can get back to work quickly.

Simpler compliance and audits. Compliance tasks can be incorporated into the development lifecycle, reducing the risk of releasing non-compliant applications. And CI/CD systems maintain detailed records of all builds, tests, and deployments, giving you a comprehensive audit trail that makes it easier to troubleshoot issues, maintain compliance requirements, and complete audits.

Less context switching. Getting real-time feedback on their code makes it easier for developers to work on one thing at a time and minimizes their cognitive load. By working with small sections of code that are automatically tested, developers can debug code quickly while their minds are still fresh from programming. Finding bugs is easier because there's less code to review.

Consistent processes. CI/CD pipelines create a standardized process for integrating and deploying code, which makes it easier for teams to collaborate, onboard new developers, reduce risk, hit key release dates, and ultimately, deliver better software faster.

Happier (and more productive) developers. With more of the deployment process automated and less context switching, the team has time for projects that are more rewarding — to the developer and their organization.

Happier users and customers. With fewer bugs making it into production, and new features and bug fixes coming out faster and more often, it becomes much easier to improve customer satisfaction and retention — and win new customers.

Key differences between CI/CD and traditional development

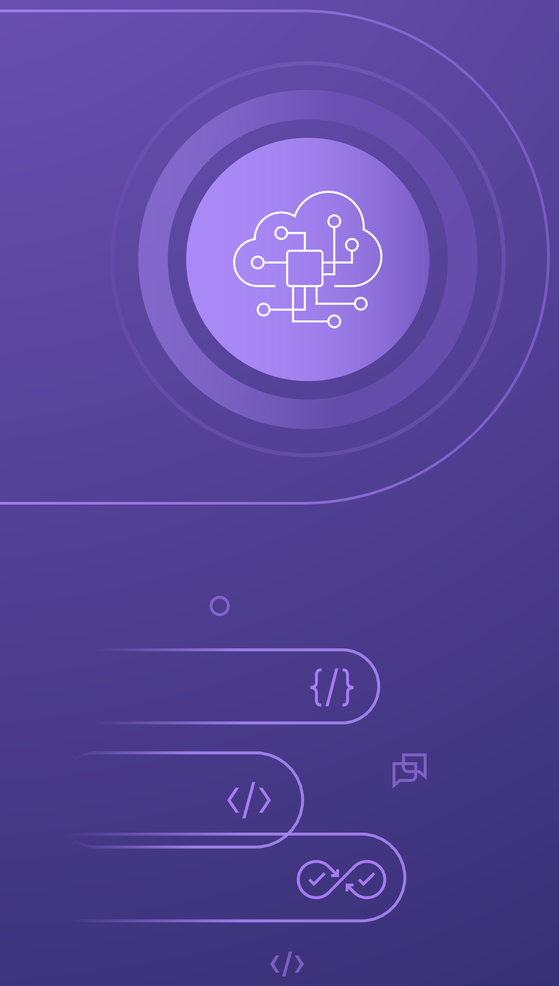
CI/CD differs from traditional software development in many ways, including:

Frequent code commits
Developers often work independently and infrequently upload their code to a main codebase, causing merge conflicts and other time-consuming issues. With CI/CD, developers push commits throughout the day, ensuring that conflicts are caught early and the codebase remains up to date.

Reduced risk
Lengthy testing cycles and extensive pre-release planning are hallmarks of traditional software development. This is done to minimize risk but often hinders the ability to find and fix problems. Risk is managed in CI/CD by applying small, incremental changes that are closely monitored and easily reverted.

Automated and continuous testing
In traditional software development, testing is done once development is complete. However, this causes problems, including delayed delivery and costly bug fixes. CI/CD supports automated testing that occurs continuously throughout development, sparked by each code commit. Developers also receive feedback they can take fast action on.

Automated, repeatable, and frequent deployments
With CI/CD, deployments are automated processes that reduce the typical stress and effort associated with big software rollouts. The same deployment process can be repeated across environments, which saves time and reduces errors and inconsistencies.



Best practices for CI/CD implementation and management

How successful you are with CI/CD depends greatly on the best practices you implement.

- Commit early, commit often.
- Optimize pipeline stages.
- Make builds fast and simple.
- Use failures to improve processes.
- Make sure the test environment mirrors production.
- Make sure to integrate with your source code management (SCM) system.
- Start where you are – you can always iterate.
- Understand the best continuous delivery is done with minimal tools.
- Track what's happening so issues and merge requests don't get out of hand.
- Streamline user acceptance testing and staging with automation.
- Manage the release pipeline through automation.
- Implement monitoring for visibility and efficiency.
- Begin experimenting with AI to make your process more efficient and secure.

Let's dive deeper into a few of these best practices.

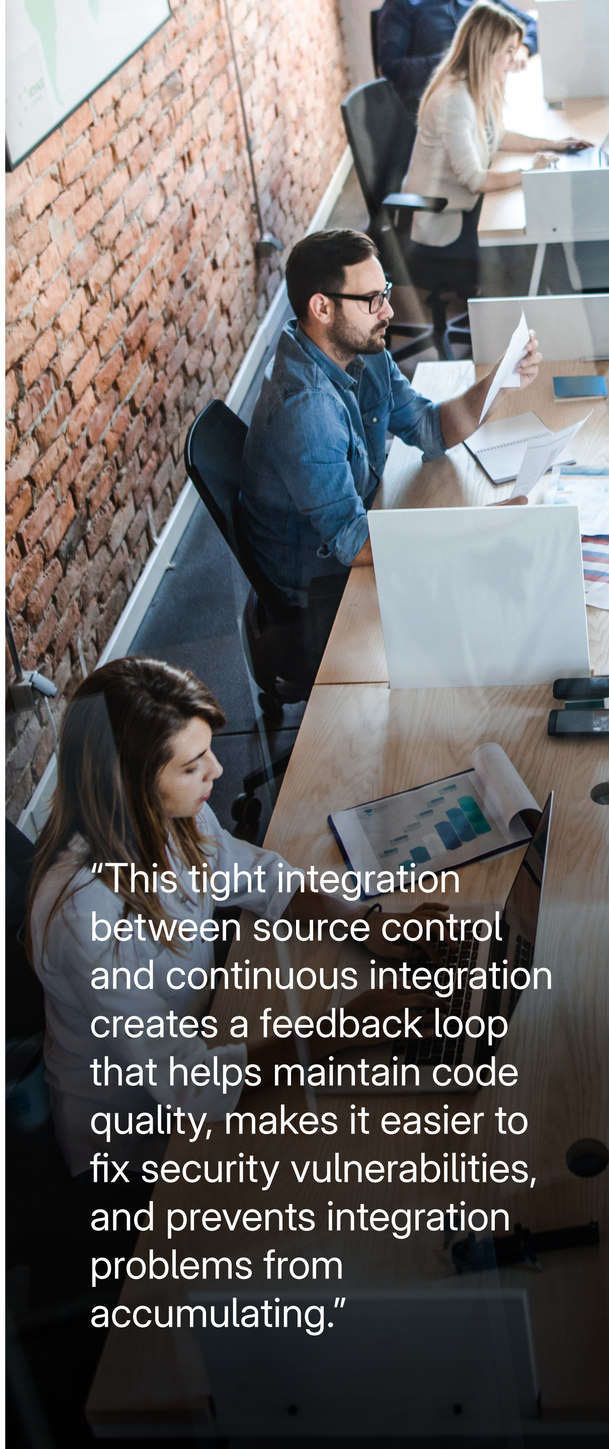
Integrating SCM and CI/CD

Source code management (SCM) and CI/CD form the foundation of modern software development practices. SCM systems like Git provide a centralized way to track changes, manage different versions of code, and facilitate collaboration among team members, all while maintaining a stable main branch that always contains production-ready code.

CI/CD takes the code managed by SCM systems and builds, tests, and validates it. If the SCM system and CI/CD tool are tightly integrated — or, ideally, part of the same DevSecOps platform — this happens automatically. When a developer submits their code changes, the CI/CD system automatically retrieves the latest code, combines it with the existing codebase, and runs through a series of automated checks. These typically include compiling the code, running unit tests, performing static code analysis, and checking code coverage. If any of these steps fail, the team is immediately notified, allowing them to address issues before they impact other developers or make their way to production.

This tight integration between source control and continuous integration creates a feedback loop that helps maintain code quality, makes it easier to fix security vulnerabilities, and prevents integration problems from accumulating. It also enhances productivity and speeds up the development process.

With separate SCM and CI/CD tools or ones that aren't tightly integrated, developers must constantly switch between them, manually triggering builds after commits and copying build information back into merge requests. This context switching increases the likelihood of human error and creates gaps in traceability between code changes and deployments. The lack of integration also makes it harder to implement automated policies that span the entire software development lifecycle, forcing teams to create custom scripts or webhooks to bridge the gap between systems. When troubleshooting issues, developers must piece together information from multiple disconnected tools, making it more time-consuming to understand what went wrong and where in the pipeline a failure occurred.



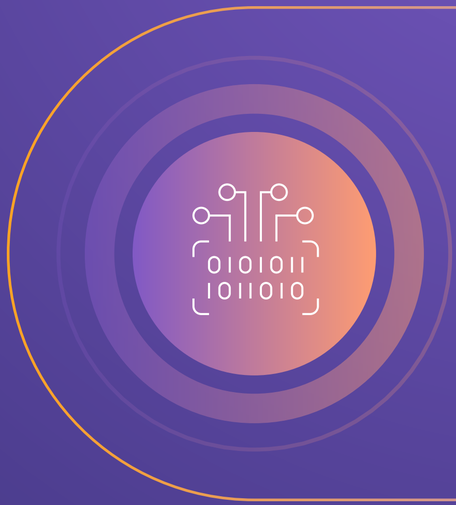
Enhancing CI/CD with AI

AI can go a step further and enhance CI/CD processes in several ways across the entire development lifecycle.

One key way organizations are using AI is to troubleshoot issues in their CI/CD pipelines. Instead of digging through job logs, error messages, and execution traces to try to determine why a CI/CD job failed, developers can use AI to determine the root cause and suggest a fix.

Developers can also use AI to help them understand, prioritize, and remediate security issues and vulnerabilities that are detected in their code. When a security scan in their CI/CD pipeline detects an issue, AI can summarize the vulnerability, give examples of how it could be exploited, and suggest a fix, complete with the necessary code to address the vulnerability.

On its own, CI/CD is a powerful way to ship better software faster. With AI, you can make your software delivery process even more efficient and secure.



How to get started with CI/CD

Getting started with CI/CD begins with identifying a simple but representative project to serve as your pilot. Choose a straightforward application with basic testing requirements, as this allows you to focus on learning the pipeline mechanics rather than dealing with complex deployment scenarios. Begin by ensuring your code is in version control and has some basic automated tests — even a few unit tests will suffice. The goal is to create a minimal pipeline that you can gradually enhance as your understanding grows.

For GitLab specifically, the process starts with creating a .gitlab-ci.yml file in your project's root directory. This YAML file defines your pipeline stages (basic ones like build, test, and deploy) and jobs. A simple pipeline might look like this: the build stage compiles your code and creates artifacts, the test stage runs your unit tests, and the deploy stage pushes your application to a staging environment. GitLab will automatically detect this file and start running your pipeline whenever changes are pushed to your repository. The platform provides built-in runners to execute your pipeline jobs, though you can also set up your own runners for more control.

As you become comfortable with the basics, gradually add more sophisticated elements to your pipeline. This might include adding code quality checks, security scanning, or automated deployment to production. GitLab's DevSecOps platform includes features like compliance management, deployment variables, and manual approval gates that you can incorporate as your pipeline matures. Pay attention to pipeline execution time and look for opportunities to run jobs in parallel where possible. Remember to add proper error handling and notifications so team members are promptly alerted of any pipeline failures. Start documenting common issues and solutions as you encounter them — this will become invaluable as your team grows.

Get started with GitLab CI/CD.

Sign up for GitLab Ultimate and try the AI-powered DevSecOps platform free for 60 days.

Learn more

About GitLab

GitLab is the most comprehensive AI-powered DevSecOps Platform for software innovation. GitLab provides one interface, one data store, one permissions model, one value stream, one set of reports, one spot to secure your code, one location to deploy to any cloud, and one place for everyone to contribute. The platform is the only true cloud-agnostic end-to-end DevSecOps platform that brings together all DevSecOps capabilities in one place.

With GitLab, organizations can create, deliver, and manage code quickly and continuously to translate business vision into reality. GitLab empowers customers and users to innovate faster, scale more easily, and serve and retain customers more effectively. Built on open source, GitLab works alongside its growing community, which is composed of thousands of developers and millions of users, to continuously deliver new innovations.