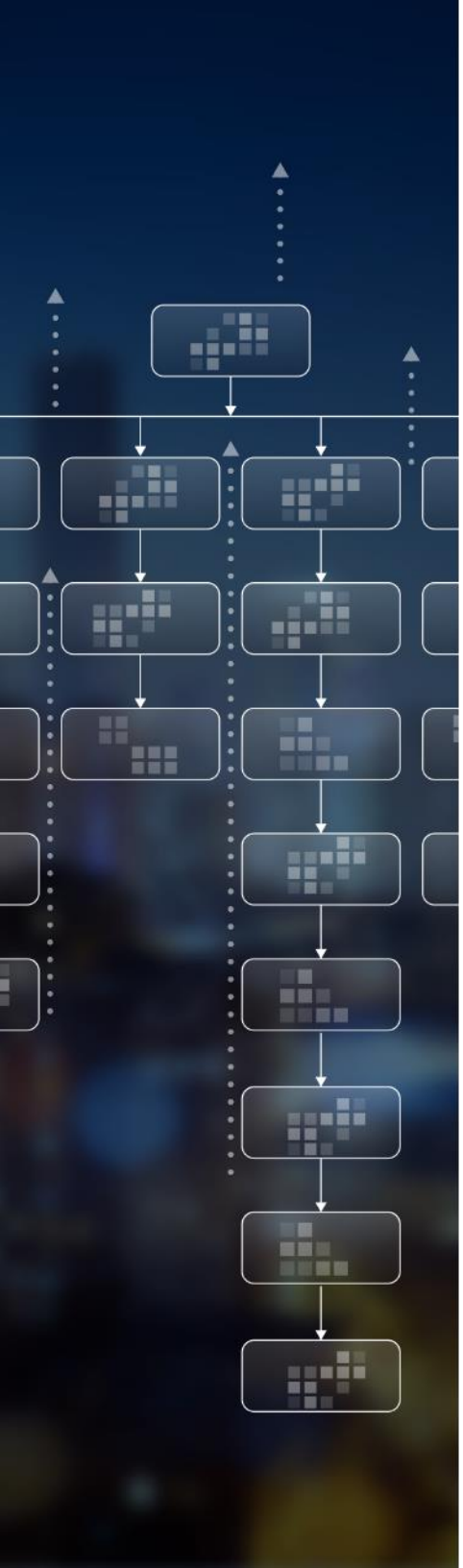


CI/CD pipelines explained: Everything you need to know

September 2024

TechTarget

In this guide:

[What is continuous integration?](#)

[What is continuous delivery?](#)

[What is continuous deployment?](#)

[Benefits and challenges of a CI/CD pipeline](#)

[Elements of a good CD/CD pipeline](#)

[What are the stages of a CI/CD pipeline?](#)

[What is an example of a CI/CD pipeline?](#)

[How do you implement a CI/CD pipeline?](#)

[CI/CD pipeline best practices](#)

CI/CD pipelines are formalized software development workflows and tool sets intended to provide a defined path for building, testing and delivering modern software. This guide takes a closer look at these continuous approaches, including how each works, benefits and challenges, stages of a CI/CD pipeline and best practices for implementation.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

CI/CD pipelines explained: Everything you need to know

STEPHEN BIGELOW, SENIOR TECHNOLOGY EDITOR

CI/CD pipelines are formalized software development workflows and tool sets intended to provide a defined path for building, testing and delivering modern software.

Speed is the key to modern software development. The monolithic all-or-nothing paradigm of traditional Waterfall software development has been replaced by rapid iterative techniques that support development and release. These techniques go by several names, including Agile, DevOps, continuous integration, continuous delivery and continuous deployment.

Although each technique offers slight differences, the common emphasis on continuous iteration has changed the nature and power of software development. Businesses can get software to market faster, test innovative new features or architectures while minimizing risk and cost, and effectively refine products over time.

Such iteration relies on well-planned and active pipelines designed to support multiple iterations in various stages of the development cycle simultaneously -- and keep entire development teams constantly busy. The key is that all of the stages that

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

compose a CI/CD pipeline are working simultaneously on different iterations of the software. For example, as one mature build is staged or deployed to production, a younger build is being tested and validated, while an even younger version is being coded and built -- and future builds are being designed to continue the software development lifecycle.

Let's take a closer look at these continuous approaches, see how each works, weigh the tradeoffs and consider best practices.

WHAT IS CONTINUOUS INTEGRATION?

Continuous integration (CI) focuses on the early stages of a software development pipeline where the code changes are built into artifacts that undergo automated tests. Multiple developers can work on the same codebase simultaneously and make frequent commits to the code repository. Build frequency can be daily or even several times per day at some points in the project's lifecycle. These small, frequent builds enable easy and low-risk experimentation, as well as the ability to easily roll back or abandon undesirable outcomes.

CI employs a variety of tools and automation techniques to create builds and shepherd them through initial testing, such as sniff or unit testing, along with more comprehensive integration and regression testing. CI is also noted for its rapid and

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

detailed feedback, letting developers and project managers see the results of the team's work in a timely manner. The limited nature of each iteration means that changes are intentionally small, so bugs can be identified, located, reported and corrected with relative ease.

CI ends when a build successfully completes initial testing and is ready to move to more comprehensive testing, such as user acceptance testing. If the build fails testing, the branch can receive further attention -- such as bug fixes or more additions -- and the build/test cycle can repeat until a build is successful.

Version control is a critical part of the CI process. Each time a developer pulls code from a code repository, it creates a branch that represents a unique version of the current codebase. When a branch is successfully built and tested, the resulting branch is typically merged or integrated back into the main codebase, which is given a new version number. This lets other developers continue working on the codebase while preventing so many branches and changes that they can't be properly integrated.

Final preparations might include packaging the build into a deployable image, such as a Docker or Apache Mesos container or traditional virtual machine (VM) image, before making it available to dedicated testers and moving the build to staging or deployment.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

WHAT IS CONTINUOUS DELIVERY?

Continuous delivery (CD) picks up where CI leaves off. It focuses on the later stages of a CI/CD pipeline, where a completed build is thoroughly tested, validated and *delivered* for deployment. Continuous delivery can -- but does not necessarily -- deploy a successfully tested and validated build.

In many cases, software delivery will result in a successful merge, where the successful branch is integrated back into the main codebase -- closing this abbreviated loop where developers can proceed with subsequent pulls to create new branches and updates.

In other cases, the successfully tested build can be packaged for deployment and delivered to a staging environment, such as a test server. Human managers can then decide whether to deploy the build, test the build in real-world conditions and report findings to developers, or forego deployment for the build in favor of continued development work.

CD likewise relies heavily on tools and automation to take a build through advanced testing, including functional, user acceptance, configuration and load testing. These validate that the build meets requirements and is ready for use in a production environment. Again, small incremental iterations ensure that any problems revealed

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

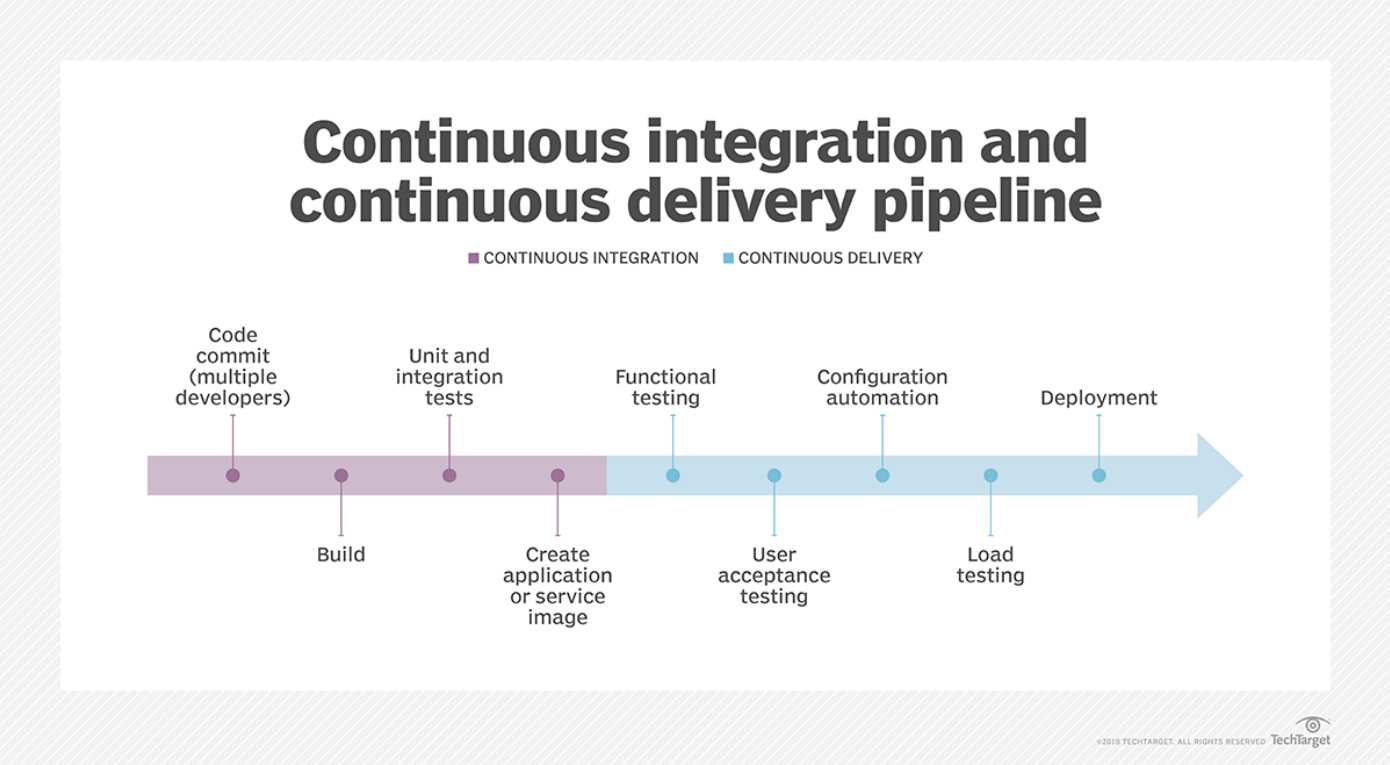
What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

in testing are identified and remediated quickly and less expensively than traditional software development approaches.



TechTarget

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

WHAT IS CONTINUOUS DEPLOYMENT?

Continuous deployment (also CD) follows the same basic steps as continuous delivery. The principal difference between delivery and deployment is that continuous deployment deliberately and automatically deploys each validated build to production. By comparison, continuous delivery typically just stages the validated build for manual deployment or other human authorization.

Continuous deployment further accelerates the iterative software development process by eliminating the lag between build validation and deployment. However, such a paradigm could also let undetected flaws or vulnerabilities slip through testing and wind up in production. For many organizations, automated deployment presents too many potential risks to enterprise security and compliance. These teams prefer the continuous delivery paradigm in which humans review a validated build -- often with further testing and analysis -- before it is released. Organizations that adopt continuous deployment typically implement a robust rollback process to ensure that a previous stable version can be redeployed in the event of unexpected issues with the newly deployed version.

The common theme through these three continuous paradigms is a heavy reliance on automation and testing to drive the pipeline workflow. Ideally, a developer needs only "press the button" to whisk a new build from the code repository through testing and on to delivery or even deployment. This tantalizing proposition depends on the

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

quality of the testing, the integrity of the automation behind it, and the careful attention of testers and software engineers.

BENEFITS AND CHALLENGES OF A CI/CD PIPELINE

In a CI/CD pipeline, everything is designed to happen simultaneously: Some software iterations are being coded, other iterations are being tested and others are heading for deployment. Still, there are important tradeoffs between CI/CD benefits and drawbacks.

The benefits of CI/CD pipelines include the following:

- **Efficient software development.** Smaller iterations -- keeping the changes to each branch small and light -- enable easier and more efficient testing. The limited scope of code in each new iteration, as well as the scope to test it, makes it easier to find and fix bugs. Features are more readily evaluated for usefulness and user acceptance, and less useful features are easily adjusted or even abandoned before further development is wasted.
- **Competitive software products.** Traditional software development approaches can take months or years to release, and formalized specifications and requirements aren't well suited to dynamically changing user needs and competitive environments. CI/CD development readily adapts to new and changing requirements, which enables developers to implement changes in

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

subsequent iterations. Products developed with CI/CD can reach market faster and with more success.

- **Freedom to fail.** CI/CD's rapid cyclicality enables developers to experiment with innovative coding styles and creative or unorthodox algorithms with far less risk than traditional software development paradigms. If an experiment doesn't work out, it won't ever see production and can be undone in the next rapid iteration. The potential for competitive innovation is a powerful driver for organizations to use CI/CD.
- **Better software maintenance.** Bugs can take weeks or months to fix in traditional software development, but the constant flow of a CI/CD pipeline makes it easier to address and fix bugs faster and with better confidence. The product is more stable and reliable over time, leading to higher levels of software quality.
- **Better operations support.** Regular software releases keep operations teams in tune with the software's requirements and monitoring needs. Administrators are better able to deploy software updates and handle rollbacks with fewer deployment errors and needless troubleshooting. Similarly, IT automation technologies can help speed deployments while reducing setup or configuration errors.
- **Availability of analytics.** The many tools involved in CI/CD pipelines are typically well integrated and can collect myriad metrics and KPIs related to development work and quality outcomes. One common metric is development velocity, indicating the amount of code changed or added over time. Another common metric is defect volume, related to the number of problems or tickets generated over time or for a build. Analytics helps ensure CI/CD pipeline

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

efficiency and can point to bottlenecks or problems that an organization can address.

Despite these compelling benefits, business leaders and development teams must consider some of the potential pitfalls of CI/CD pipelines:

- **Dedication to automation.** CI/CD relies on the consistency of an established tool set and strong automation framework to build, test and deploy each build. This demands a serious intellectual investment to implement and manage the automation, which can involve a steep learning curve and constant attention. Changes to the development process or tool set can profoundly impact the CI/CD pipeline, so CI/CD is often employed in mature and active development environments.
- **Staff discipline and planning.** A CI/CD process cannot bring full value to the business if it's not constantly generating new builds, testing release candidates and deploying selected candidates to production. This requires careful planning and expert project management skills. Developers must adhere to established development guidelines to ensure quality, style and architectural standards. Meanwhile, business leaders and project stakeholders can be extremely uncomfortable with automated deployments in continuous deployment paradigms, and meetings for manual go/no-go deployment decisions can be fraught with stress over unknown or unforeseen consequences. This can cause unnecessary delays -- all while new builds are coming through the pipeline.
- **Communication and collaboration.** No amount of automation and tooling is a substitute for effective communication and collaboration among developers, operations teams, project managers and project stakeholders. These vital team

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

interactions facilitate the rapid, efficient experimentation that makes CI/CD so powerful. Automation and tools are just the means to that end.

ELEMENTS OF A GOOD CD/CD PIPELINE

Here are some common characteristics of a good CI/CD process:

- **Speed.** A CI/CD pipeline can have numerous steps and parts, but a build should be able to move through the pipeline (integration, testing, delivery and even deployment) in short order -- just minutes to complete an integration and a few hours for testing cycles to finish. If it takes days to move a build through the pipeline, a great deal of valuable time is being wasted, and the process needs fine-tuning.
- **Consistency.** The processes and components used in one place or cycle are exactly the same in all places and cycles. For example, if a build results in a Docker container, that container is the object that's tested and moved through the pipeline to delivery or deployment. Developers can write scripts and create automation processes with confidence that such efforts will be successful and deliver the desired results every time. Processes that introduce variations or manual steps slow the pipeline and invite errors and inefficiency.
- **Tight version control.** Well-documented repositories or components and builds enable comprehensive branching and merging as well as rapid restoration of previous builds when new builds or deployments go awry. Good version control facilitates fast, accurate and confident rollbacks to the previous working version whenever the need arises.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

- **Automation.** Extensive automation moves new code through integration, testing and delivery or deployment with little, if any, manual interaction. Ideally, a human does little more than develop and commit code, then wait for an approved pull request and approval before a deployment, if needed. A pipeline that depends on manual steps and processes is slow and prone to errors.
- **Integrated feedback loops.** A CI/CD pipeline is a loop that yields countless iterative steps to a completed project, and each phase also offers a loop back to the beginning. A problem with the source code won't generate a build. A problem with the build won't move into testing. A problem in testing or after deployment will demand source fixes. The sooner a problem is identified, the faster, easier and cheaper it is to fix, keeping the overall pipeline in motion. Instrumentation can often be added to collect important metrics and provide comprehensive reporting for added feedback, which can help optimize the CI/CD process.
- **Security throughout.** Oversights and mistakes in programming and testing can create vulnerabilities and expose software to malicious activity. Thus, it is critical to infuse security best practices throughout the CI/CD pipeline. Tools such as vulnerability checkers can help spot potential security flaws in the code flowing through the pipeline, while additional security evaluations should take place during the testing phase.
- **Solid tool interoperability.** A CI/CD pipeline can involve many tools, and each tool across the pipeline should integrate well -- or support some level of interoperability so the output product of one tool can easily feed the input of the next tool in the pipeline. Without good integration or interoperability, the pipeline might require manual conversions or other human intervention that can lead to

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

errors and inefficiency. A central part of CI/CD tool selection is the consideration of interoperability and suitability to task.

WHAT ARE THE STAGES OF A CI/CD PIPELINE?

The CI/CD pipeline combines continuous integration, delivery and deployment into four major phases: source, build, test and deploy. Each phase uses highly detailed processes, standards, tools and automation. Just as physical products made in a factory can benefit from customized manufacturing machines, software pipelines are frequently tailored to suit the specific needs of the project and the business.

Source. The first phase in a CI/CD pipeline is the creation of source code, where developers translate software requirements into functional algorithms, behaviors and features. The tools employed for this depend on whether the development team is working in Java, .NET, C#, PHP or countless other development languages. Integrated development environments (IDEs) are often selected because they support a specific language in addition to various code-checking features, such as basic error detection, vulnerability scanning and adherence to established code quality standards. Other source code and pipeline support tools, including code repositories and version control systems such as Git, typically form the foundation for building and testing phases.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

There is no single pipeline for source creation. A business might have multiple source portions of the CI/CD pipeline depending on the various projects in development -- such as a server-side platform using C++, website applications using Java and mobile applications using Go. Similarly, coding features can vary between IDEs and projects due to different standards or vulnerabilities between projects, such as enterprise production systems versus a consumer app.

Build. The build process draws source code from a repository; establishes links to relevant libraries, modules and dependencies; and compiles, or builds, all these components into an executable (EXE) file or some other suitable construct. Tools used in this stage also generate logs of the process, denote errors to investigate and correct, and notify developers that the build is completed.

As with source code creation, build tools typically depend on the selected programming language. A development group might employ independent tools to generate a build; many IDEs incorporate such build capabilities, which means they effectively support both the source creation and building phases within a CI/CD pipeline.

A build phase might employ additional tooling, such as scripts, to translate the executable file into a packaged or deployable execution environment, such as a VM -- complete with an operating system and related components -- or a container, such

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

as Docker container with libraries and dependencies and using Kubernetes for orchestration and automation.

Test. Although source code has already completed some static testing, the completed build now enters the next CI/CD phase of comprehensive dynamic testing. This usually starts with basic functional or unit testing to verify that new features and functions work as intended, and regression testing to ensure that new changes or additions do not accidentally break any previously working features. The build also undergoes a battery of tests for integration -- ensuring that the changed component will continue to work properly with other components -- as well as user acceptance and performance. If errors occur during testing, the results are looped back to developers for analysis and remediation in subsequent builds.

Automation is particularly critical in the CI/CD test phase, where a build is subjected to an enormous array of tests and test cases to validate its operation. Human testing is typically too slow and subject to errors and oversights to ensure reliable or objective testing outcomes. Test specialists create comprehensive test cases and criteria but depend on test tools to implement testing and validation in a busy pipeline.

Deploy. The build passes the testing phase and is considered a candidate for deployment in a production environment. In a continuous delivery pipeline, it is sent

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

to human stakeholders, approved and then deployed, if appropriate. In a continuous deployment pipeline, the build automatically deploys as soon as it passes its test suite.

The deployment process typically involves creating a deployment environment -- for example, provisioning resources and services within the data center -- and moving the build to its deployment target, such as a server. These steps are typically automated with scripts or through workflows in automation tools. Deployments also usually connect to error reporting and ticketing tools to find unexpected errors after the build is deployed and alert developers. Users can also submit bug tickets to denote real or perceived errors with the release.

Even the most wildly optimistic deployment candidates are rarely committed to production without reservation. Deployments frequently involve additional precautions and live testing periods, including beta tests, A/B tests, blue/green tests and other cutover periods to curtail or roll back unforeseen software problems and minimize the business impact.

In this guide:

[What is continuous integration?]

[What is continuous delivery?]

[What is continuous deployment?]

[Benefits and challenges of a CI/CD pipeline]

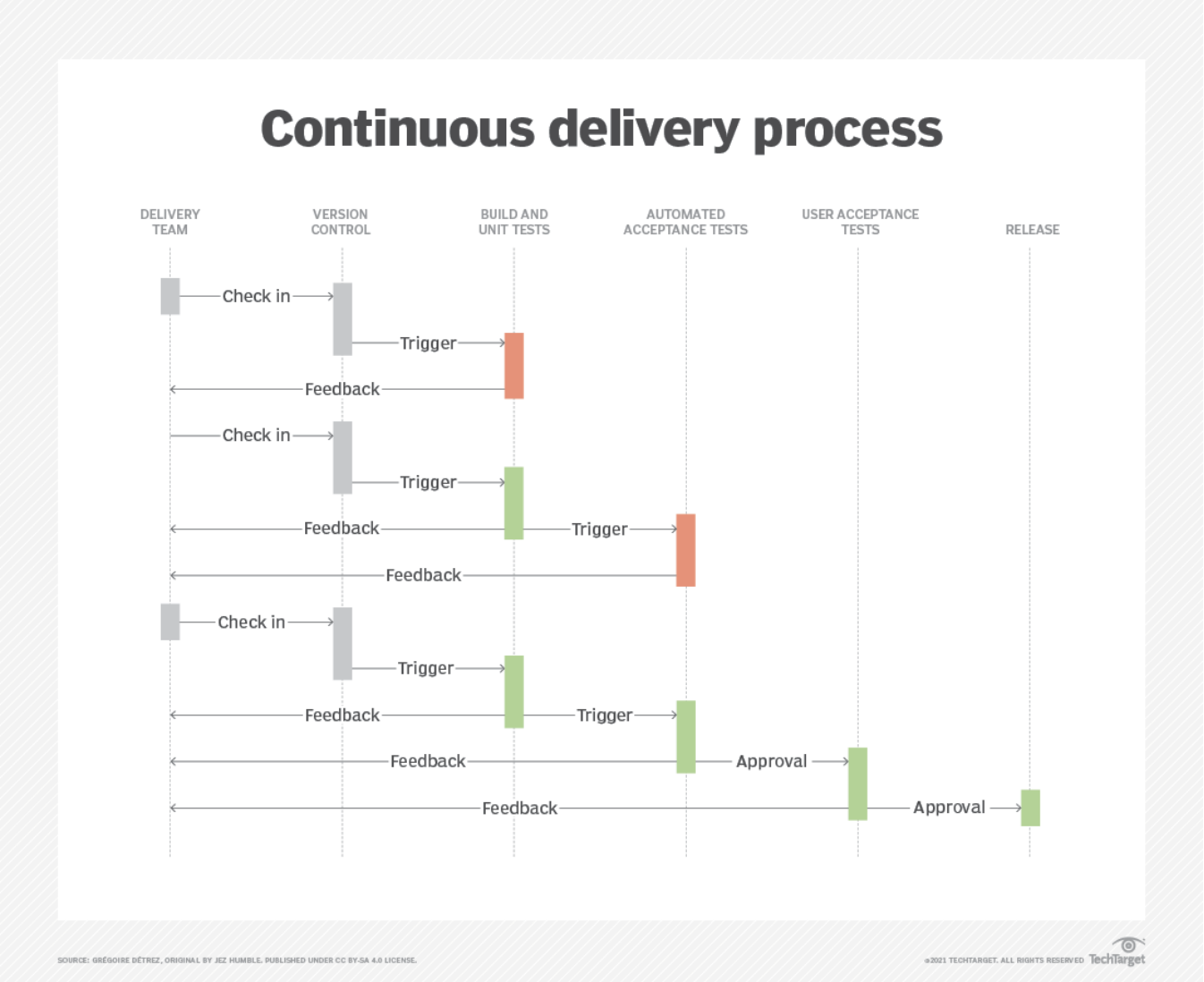
[Elements of a good CD/CD pipeline]

[What are the stages of a CI/CD pipeline?]

[What is an example of a CI/CD pipeline?]

[How do you implement a CI/CD pipeline?]

[CI/CD pipeline best practices]



TechTarget

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

WHAT IS AN EXAMPLE OF A CI/CD PIPELINE?

There is no single right way to build a CI/CD pipeline. Every pipeline can use different tools and include variations or alternate pathways to support project types with different scopes and sophistication. Regardless of the approach, an ideal CI/CD pipeline should meet three fundamental goals:

- Improve the software's quality.
- Make software development faster and more agile.
- Boost confidence in software deployment to production.

Let's examine a typical CI/CD pipeline, consider the activities within each stage and note several tools to tackle them.

SOURCE

Developers write code using editors or IDEs. A development team might employ several editors or IDEs to support multiple languages for different projects.

Tools: Examples of software IDEs include Atom, Cloud9 IDE, Microsoft Visual C++ or Visual Studio, PyCharm and Xcode.

The source code is typically stored in a common shared *repository*, or *repo*, where multiple developers can access and work on the codebase at the same time. Repos also generally hold other parts of the software development process, such as artifacts

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

(of compilation and linking), libraries, executables, modules, test scripts and suites. Repos provide a comprehensive version control system, which ensures developers work on the latest codebase and integrate the latest components in the build process.

Tools: GitHub, based on Git, is a popular repo; other examples of repositories include GitLab, Cloudsmith Package, Docker Hub for container projects, JFrog Artifactory, NuGet and SVN.

Once a developer commits changes to the codebase, those changes are saved to the version control system in the repository, which automatically triggers a new build.

BUILD

The build process typically involves multiple steps: Fetch the source code components from the repo, compile code, and link libraries and other modules. The result is a simple executable file or a more complex assembly, such as a deployable container or VM. All of the artifacts involved in the build process are typically retained in the repository. If there are problems or errors in the build, the process stops and issues are reported back to the developers for remediation. Typical problems include functional errors, such as a divide-by-zero math error, or missing components -- for example, a required library or module is not present in the build manifest.

Tools: Many IDEs include build tools natively tailored to the selected programming language. Alternatively, standalone build tools include Ant, Gradle, Make, Maven,

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

Meister, Phing and Rake. Jenkins is a popular CI engine, but there are Jenkins alternatives such as Atlassian Bamboo, AWS CodePipeline, CircleCI, GitHub Actions and TeamCity.

The build stage might also include some basic testing for vulnerabilities, such as software composition analysis (SCA) and static application security testing (SAST).

Tools: Examples of SAST tools include Arctic Wolf Vulnerability Assessment, Fortify Static Code Analyzer and Netsparker. Vendors with SCA tools include Checkmarx, Kiuwan, Snyk, Synopsys and Veracode.

If the build completes successfully and passes initial test scans, it moves to the CI/CD testing phase.

TEST

Although some testing is naturally part of the build process, such as syntax and code quality, most happens after the build is successfully completed. This complex phase involves numerous steps and goals, including the following:

- **Unit testing.** Validates new features and functions added to the build.
- **Dynamic application security testing (DAST).** Scans the build for security flaws, such as weak passwords or missing credentials.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

- **Interactive application security testing (IAST).** Analyzes traffic and execution flow to detect security issues, including those in third-party or open source components.
- **Regression testing.** Verifies that changes or additions do not harm previous features.
- **Integration testing.** Ensures the build operates with other applications or services.
- **User acceptance testing.** Assesses whether users are able to use new features and functions as intended.
- **Performance testing.** Ensures the build operates as required under load.

Developers and software testing specialists create test conditions that provide input to the build and compare the actual response or output to the expected response. If they match, the test is considered successful, and the build moves on to the next test. If they do not match, the deviation is noted, and error information is sent back to the development team for investigation and remediation.

Tools: There are hundreds of test tools applicable through these tests. Each supports some level of automation, and some can perform multiple types of tests. Popular testing tools include Appian, Katalon, Kobiton, Kualitee, OpenText UFT One, Qaprosoft, Sauce Labs, Selenium, Telerik Test Studio and TestArchitect.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

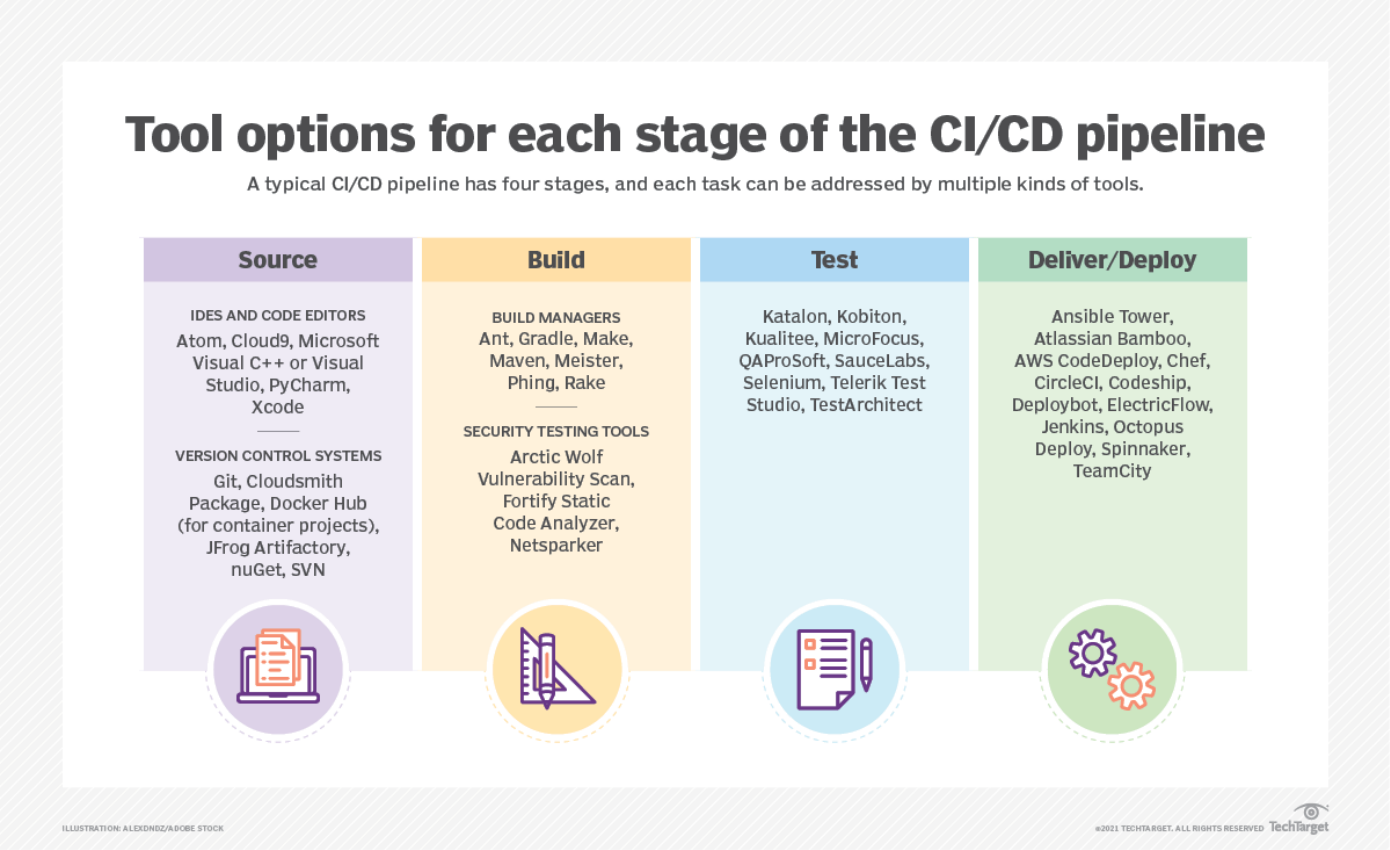
Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices



Not all builds that successfully complete the testing phase move into the deployment phase. Some builds might simply represent interim steps that need validation but are not yet ready for deployment. For example, developers might test an incomplete feature subset, flesh out the remaining feature subset in a subsequent build and then deploy it in its entirety.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

DEPLOY

A build that successfully passes testing might be initially deployed to a staging or test server; this is sometimes called a *test deployment* or preproduction deployment. A script copies a build artifact from the repo to a desired test server, then sets up dependencies and paths.

Once on a test server, the build can be configured to simulate a production environment; for instance, access to test databases and other applications can be enabled for "real-world" functional and performance evaluations. Much of this relies on automation but might involve human testing to shake down nuances of the build. This is sometimes called an *alpha* or *development release* and involves only a small base of well-informed testers and users. User acceptance testing is often performed here.

Preproduction deployment is typically the endpoint for continuous delivery pipelines. Once the build is completely validated and stakeholders have confidence in the build's stability and integrity, it can be deployed to an actual production environment. In a continuous deployment pipeline, once the build passes predeployment testing, it is automatically deployed to production.

To improve safety and guard against unforeseen consequences, a new build might be deployed in parallel to the current build in an A/B configuration, also called *beta testing*. This testing gradually expands to larger user groups until all users are on the

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

new build. At that point, the previous build is retired and its computing resources freed for other applications.

Tools: There are many dedicated deployment tools, including Ansible Tower, Chef, Codeship, ElectricFlow, Jenkins, Octopus Deploy, Spinnaker and TeamCity. Deployments also might involve application performance monitoring tools and other types of instrumentation to check and report on an application's health.

HOW DO YOU IMPLEMENT A CI/CD PIPELINE?

There is no one way to set up a CI/CD pipeline. Precise steps vary between tools and the process to implement -- and that's by design, to tailor an incredibly agile pipeline that meets the needs of the business and its projects.

Still, there are common steps and decisions at each stage of pipeline construction that apply to any CI/CD process:

1. Select a version control system to maintain code repositories. Determine if you need a hosted version, or a hosting provider. Major cloud providers also offer options here, such as Azure DevOps.
2. Create repositories to house application source code, artifacts, branches and builds.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

3. Determine what build, or CI, server to use. This can be self-hosted, such as Jenkins, or a third-party option such as GitHub Actions, CircleCI or Azure Pipelines.
4. Implement a task in the pipeline that compiles application source code into a build. In some setups, this will generate a Docker image or a VM.
5. Run basic tests on the code, such as static analysis and style checks, to ensure its quality and consistency with organizational guidelines.
6. The build should now generate an artifact, or container image, published to a store or registry.
7. Initiate further testing on the build, as listed previously (functional, security, user acceptance, etc.). If predetermined thresholds are not met, fail the stage. Publish results of tests and code coverage so they are easily available.
8. Once the software build passes tests, it is ready for final preparations to production deployment. This can include multiple staged environments, such as blue/green and canary deployments.
9. Consider deployment parameters such as provisioning resources, connecting services and migrating the successful release candidate into the production environment. It's common to connect instrumentation to collect data on the software's performance, errors and other factors. Rollback plans should also be in place.

Note that CI/CD based in the cloud functions the same but relies heavily on tools and services native to the cloud provider's platform, which might require specific steps.

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

CI/CD implementation encounters some specific challenges, too. Here are two major ones to watch out for:

- **Limited testing.** Allocating and coordinating resources and intellectual investment to configure test environments and construct test cases is a common problem for CI/CD pipelines. Continuous development involves multiple code commits and parallel testing demands that frequently result in configuration conflicts and limited/forgotten test cases. This can lead to errors slipping through the test phase and degrading the pipeline's efficiency. There's no substitute for skilled and knowledgeable software testers and well-documented requirements and goals.
- **Fixing bugs.** Pipelines are designed to provide feedback loops back to developers who can fix bugs in a new build. Finding a bug is easy enough, but it can be difficult to identify the specific developer responsible to fix that section of code. This makes it harder to hold developers responsible for their work and can obfuscate the need for more training. Logging, team communication and copious documentation can help determine the location of the bug and identify the developers to be involved in its resolution.
- **Ensure security.** Software projects represent a significant investment for any business, and that investment must embrace comprehensive security. From an external standpoint, the pipeline and its data -- from source code to test data to finished builds -- should be stored in a secure manner with attention to identity and access management and other features to ensure that only authorized developers can access that intellectual property. From an internal standpoint, the software should be designed and implemented using comprehensive

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

security techniques, such as APIs with authentication, to ensure that only appropriate users can utilize the software being developed.

CI/CD PIPELINE BEST PRACTICES

A business and its development teams can employ various methods to get the most from a CI/CD pipeline. These CI/CD best practices can help organizations derive even more value from them.

Start small. CI/CD brings speed and agility, so give the process time to evolve and enable developers to try different tools and steps. A business might start with a CI pipeline and add CD later. Small filler projects are ideal places to try new tools and techniques that can enhance a broader pipeline.

Work small. CI/CD is designed for limited and incremental code changes. This prevents one developer's pull from locking other developers out of that code for an extended period. Further, small changes can be tested and validated faster with fewer potential bugs to address. Small increments help keep the pipeline filled.

Define success. Understand the intended benefits, such as faster code building or lower error/rework rates, and then implement metrics to measure those criteria. Compare the metrics against pre-pipeline performance and track those metrics as the

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

pipeline evolves. This makes it easier to see the pipeline's value, spot problems over time and invest in ways to build and enhance the CI/CD pipeline.

Document processes. Often overlooked and underappreciated, documentation is an essential part of the development pipeline. It lays out the process and tools for all developers and business users and explains how everything is related and configured. Plus, it can help troubleshoot problems and alleviate unintended scope creep or configuration drift. Documentation also contributes to an organization's compliance and security posture, enabling leaders to audit activities.

Think about operations. Agile development paradigms, such as DevOps and continuous deployment, embrace both operations and development roles. Developers must understand both deployment and operations, and take greater ownership of the software's reliability, security and performance. Business and project leaders must foster and reinforce this attitude shift.

Focus on feedback. Feedback within the CI/CD pipeline is most effective when every step -- and every participant -- actively works to spot and address issues to save time and work efficiently. This starts with spotting errors in the source code and continues all the way through testing and deployment. For example, find and fix a syntax error in the source code at the build stage, rather than waste time and effort

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

during the testing phase. Categorizing and analyzing errors can also help businesses improve the development skills and processes.

Infuse security throughout. Security scanning tools at the code level (SAST and SCA tools) are handy for early vulnerability and error diagnostics but can produce a large number of false positives. Security scanning at the test level (DAST and IAST tools) requires the software to be built and running, which means errors are caught later in the pipeline where bug fixes are more time-consuming and costly. Select the best security scanning tools for the tasks at hand and use those tools to automatically update the bug tracking system and automatically generate tickets for fast examination and remediation.

Also, think about the security of the pipeline itself: Improper authorization and authentication can expose tools and repositories to malicious actions. Secure the tools and repositories, use logs to track user access to them and flag unusual activities -- such as downloads to IP addresses outside of the corporate LAN.

Embrace continuous testing. During the source and build stages, perform SCA, SAST and other basic code scans for style and security standards. When the build is complete, apply a battery of established test conditions using test tools. Start with simple functional validation and systematically expand testing to more complex and comprehensive integration, in-depth security (such as DAST) and performance. Staff

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

must carefully construct tests and test cases to validate the features and functionality of each new build as the builds, the project and even the project's requirements evolve. Frequently added new features require frequent tests and test cases.

Foster communication. Integration and delivery work together but are often implemented separately and handled by different teams, such as coders and testers. A CI/CD pipeline that functions smoothly requires timely and clear communication and collaboration between different teams across the pipeline; otherwise, it can easily break down with unnecessary delays.

Avoid waste. Creating and maintaining a CI/CD pipeline incurs various costs for tools, infrastructure and resources. Inefficient use of any of these -- underutilized or unused tools, overallocated IT infrastructure for testing and deployments, poor coordination, insufficient evaluation of processes or tools -- can slow development and impair developer productivity. CI/CD pipelines are dynamic entities that require frequent refinement and regular developer training to operate efficiently and reliably.

This also means avoiding less obvious areas of inefficiencies. Don't make 10 different builds in the same day if there is no practical way to test and deploy those 10 builds in the same day. Teams and project effort must reflect the most effective use of the pipeline.

Enhance the ecosystem. CI/CD and other agile pipelines are ecosystems composed of tools tied together with processes and automation, with myriad alternate

In this guide:

What is continuous integration?

What is continuous delivery?

What is continuous deployment?

Benefits and challenges of a CI/CD pipeline

Elements of a good CD/CD pipeline

What are the stages of a CI/CD pipeline?

What is an example of a CI/CD pipeline?

How do you implement a CI/CD pipeline?

CI/CD pipeline best practices

paths and steps for different products. Teams should always evaluate new tools and refine the processes to keep the overall pipeline as smooth and efficient as possible.

Recover resources. Busy CI/CD environments can consume significant storage, along with extensive commitments of compute resources for building, testing and deployments. As projects evolve, it's important for business and project leaders to consider how unused resources are tracked and recovered for reuse. For example, if a new build is deployed to supplant a previous build, the resources utilized by the obsolete build should eventually be recovered for reuse.

Stephen J. Bigelow, senior technology editor at TechTarget, has more than 30 years of technical writing experience in the PC and technology industry.

CONTINUE READING

- **How to weigh the benefits and challenges of CI/CD**
- **CI/CD best practices for DevOps teams**
- **Continuous delivery vs. continuous deployment: Which to choose?**