WHAT IS CI/CD? CONTINUOUS INTEGRATION & CONTINUOUS DELIVERY EXPLAINED

CI/CD stands for Continuous Integration and Continuous Delivery—a practice that introduces automation and continuous monitoring across the entire software development lifecycle (SDLC). CI handles code integration, building, and testing; CD automates deployment to production. Together, they form a unified pipeline that accelerates software delivery without sacrificing quality.

Flexibility, speed, and quality are the core pillars of modern software development. Increased customer demand and the evolving technological landscape have made software development more complex than ever, making traditional software development lifecycle (SDLC) methods unable to cope with the rapidly changing nature of developments.

Practices like Agile and DevOps have gained popularity in facilitating these changing requirements by bringing flexibility and speed to the development process without sacrificing the overall quality of the end product. CI/CD is the key practice that makes rapid, reliable delivery achievable within those frameworks.

(This article is part of our DevOps Guide. Use the right-hand menu to navigate.)

What is CI/CD?

CI/CD refers to Continuous Integration and Continuous Delivery. In its simplest form, CI/CD introduces automation and monitoring to the complete SDLC.

- Continuous Integration is the first part of a software delivery pipeline, where application code is integrated, built, and tested.

- Continuous Delivery is the second stage, where the application is deployed to its production environment to be utilized by end-users.

What is Continuous Integration?

Continuous Integration (CI) is the practice of automatically integrating, building, and testing code changes as developers push them to a shared version control system—producing a validated, deployable artifact at the end of each cycle.

Modern software development is a team effort with multiple developers working on different areas, features, or bug fixes of a product. All these code changes need to be combined to release a single end product. Manually integrating all these changes can be a near-impossible task, and there will inevitably be conflicting code changes with developers working across multiple streams simultaneously.

Continuous Integration offers the ideal solution by allowing developers to continuously push their code to the version control system (VCS). These changes are validated, and new builds are created from the new code that will undergo automated testing.

This testing typically includes unit and integration tests to ensure that changes do not cause any issues in the application. Continuous Integration also ensures that all code changes are properly validated, tested, and that immediate feedback is provided to the developer from the pipeline in the event of an issue, enabling them to fix problems quickly.

This not only increases the quality of the code but also provides a platform to quickly identify code errors with a shorter automated feedback cycle. Another benefit of Continuous Integration is that it ensures all developers have the latest codebase to work on, as code changes are quickly merged and merge conflicts are mitigated.

The end goal of the Continuous Integration process is to create a deployable artifact.

What is Continuous Delivery?

Continuous Delivery (CD) automates the deployment of a validated artifact from the CI stage through staging, pre-production, and production environments—including infrastructure, configuration, and monitoring responsibilities.

Once a deployable artifact is created, Continuous Delivery takes over by automating the entire delivery process. CD is responsible for application deployment as well as infrastructure and configuration changes, monitoring, and maintaining the application. CD can extend beyond application delivery to include infrastructure and data operations — which is where the Control-M integration with Jenkins CI/CD adds value, orchestrating the production data and application workflows that run downstream of every Jenkins deployment.

- Terraform
- Ansible
- Chef
- Puppet

Continuous Delivery also supports multi-stage deployments where artifacts are moved through different stages—staging, pre-production, and finally production—with additional testing and

verifications at each stage, further increasing the reliability and robustness of the application.

Why do organizations need CI/CD?

CI/CD is the backbone of modern software development, enabling organizations to develop and deploy software quickly and efficiently while adapting to changing requirements without slowing delivery.

A properly configured CI/CD pipeline offers a unified platform to integrate all aspects of the SDLC—from source control and testing tools to infrastructure modification and monitoring tools. This allows organizations to adapt to changing consumer needs and technological innovations with ease. In a traditional development strategy, fulfilling client change requests or adopting new technology is a long-winded process—and the consumer need may have shifted by the time the organization has responded.

DevOps with CI/CD solves this issue because CI/CD pipelines are far more flexible. A consumer requirement that is not currently addressed can be quickly identified, analyzed, developed, and deployed in a relatively short amount of time without disrupting the normal development flow of the application.

CI/CD also enables quick deployment of even small changes, providing end-users with direct visibility into the development process. End-users can see the product grow through frequent deployments tied to bug fixes or new features—a stark contrast with the waterfall model, where users only see the final product after the entire development cycle is complete.

How has CI/CD evolved?

CI/CD has evolved well beyond application delivery to encompass database changes, infrastructure as code, and integrated security testing across the full software pipeline.

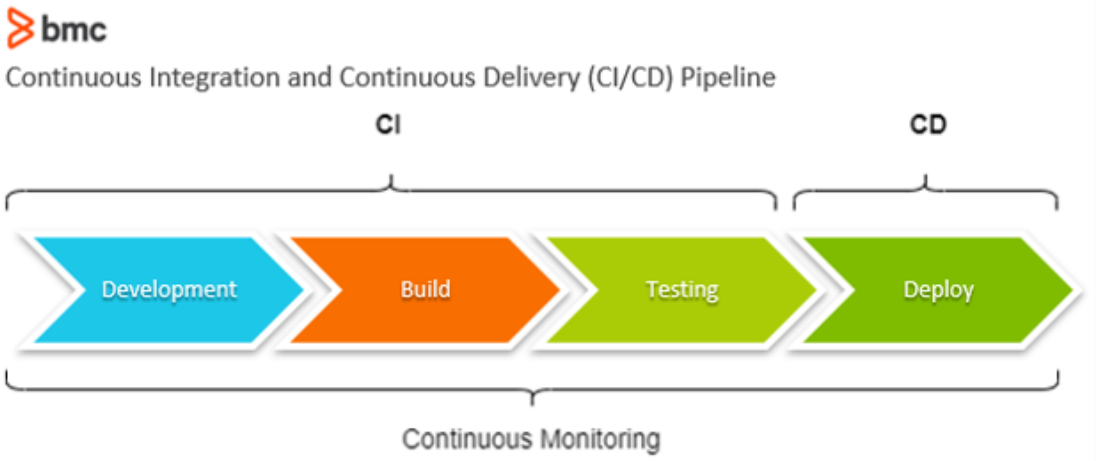
CI/CD began only as a platform to support application delivery. It has since expanded to support:

- Database DevOps, where database changes are continuously delivered alongside application changes.
- GitOps, where infrastructure is defined in a declarative, version-controlled manner and managed via CI/CD pipelines.

Users can now integrate almost all aspects of software delivery into Continuous Integration and Continuous Delivery. CI/CD can also extend into DevSecOps, where security testing—such as vulnerability scans, configuration policy enforcement, and network monitoring—is integrated directly into CI/CD pipelines rather than treated as a separate, post-development concern.

How does a CI/CD pipeline work?

A CI/CD pipeline is a software delivery process that automates the path from code commit to production deployment. The complexity and stages of a CI/CD pipeline vary depending on development requirements—one pipeline might deliver software as containers to a multi-cloud Kubernetes cluster, while another may simply build, test, and deploy an application as a serverless function.



A typical CI/CD pipeline breaks down into four stages:

- Development: Code is merged to a version control repository and validated.
- Build: The application is built using the validated code, producing the artifact used for testing.
- Testing: The built artifact is deployed to a test environment, where extensive tests are carried out to ensure application functionality.
- Deploy: The tested application is deployed to the production environment.

All stages are continuously monitored for errors, and relevant parties are notified quickly when issues arise.

What are the advantages of CI/CD?

CI/CD increases the speed and efficiency of the software development process while providing a top-down view of all delivery tasks. Key benefits include:

- Improved developer and QA productivity through automated validations, builds, and testing
- Time and resource savings by automating mundane and repeatable tasks
- Improved overall code quality
- Faster feedback cycles, with each stage and the process in the pipeline being continuously monitored
- Reduced bugs and defects in production systems
- Broader delivery support, including database and infrastructure changes directly through the pipeline
- Architecture flexibility, supporting everything from traditional server-based deployment to containers and serverless architectures
- Application reliability, through continuous monitoring in the production environment

What are the most popular CI/CD tools and platforms?

CI/CD tooling ranges from general-purpose platforms to architecture-specific tools, with options available natively through major source control systems.

Continuous Integration tools

- Jenkins
- TeamCity

- Travis CI
- Bamboo
- CircleCI

Continuous Delivery tools

- ArgoCD
- JenkinsX
- FluxCD
- GoCD
- Spinnaker
- Octopus Deploy

Cloud-based CI/CD platforms

- Azure DevOps
- Google Cloud Build
- AWS CodeBuild / CodeCommit / CodeDeploy
- GitHub Actions
- GitLab Pipelines
- Bitbucket Pipelines

Summing up CI/CD

Continuous Integration and Continuous Delivery have become an integral part of most software development lifecycles. With continuous development, testing, and deployment, CI/CD has enabled faster, more flexible development without increasing the workload on development, quality assurance, or operations teams.

Today, CI/CD has evolved to support all aspects of delivery pipelines, facilitating new paradigms such as GitOps, Database DevOps, and DevSecOps—and we can expect more to come.

BMC supports Enterprise DevOps

From legacy systems to cloud software, BMC supports DevOps across the entire enterprise. Learn more about Enterprise DevOps.

FAQ: CI/CD explained

What is the difference between Continuous Integration and Continuous Delivery?

Continuous Integration automates the process of merging, building, and testing code changes to produce a deployable artifact. Continuous Delivery takes that artifact and automates its deployment through staging environments to production. CI is the "build and test" phase; CD is the "deploy and release" phase.

What is the difference between Continuous Delivery and Continuous Deployment?

Continuous Delivery automates deployment up to production but typically requires a manual

approval step before releasing to end-users. Continuous Deployment goes one step further—every change that passes automated testing is deployed to production automatically, with no manual gate required.

What does a CI/CD pipeline include?

A CI/CD pipeline typically includes four stages: development (code commit and validation), build (artifact creation), testing (automated unit, integration, and other tests), and deploy (release to production). All stages are continuously monitored for errors.

Can CI/CD support security and compliance testing?

Yes. CI/CD pipelines can be extended to support DevSecOps by integrating security testing—such as vulnerability scanning, configuration policy enforcement, and network monitoring—directly into the pipeline rather than treating security as a separate post-development phase.

What is the relationship between CI/CD and DevOps?

CI/CD is a core technical practice within DevOps. DevOps describes the cultural and organizational approach to unifying development and operations; CI/CD provides the automation tooling and pipeline infrastructure that makes rapid, reliable delivery achievable in practice.

Related reading

- BMC DevOps Blog
- DevOps Branching Strategies Explained
- What's CD4ML? Continuous Delivery with Machine Learning Explained
- DevOps Job Titles, Roles, & Responsibilities
- The Complete DevOps Certifications Guide 2021-2022
- Control-M Python Client & Integrations

The views and opinions expressed in this post are those of the authors and do not necessarily reflect the official position of BMC.