DEPLOYMENT PIPELINES (CI/CD) IN SOFTWARE ENGINEERING

A deployment pipeline is the automated process that takes code from version control and delivers it to production—without manual steps. On any software engineering team, a well-designed deployment pipeline enables developers and DevOps professionals to reliably compile, build, test, and deploy their code at speed and scale.

The most common pipeline components are build automation/continuous integration, test automation, and deploy automation—supported by tools across five categories: source control, build tools, containerisation, configuration management, and monitoring.

What is a deployment pipeline?

A deployment pipeline is the process of taking code from version control and making it available to end users in an automated, repeatable fashion. Before deployment pipelines, releasing software was a manual process prone to communication overhead and human error.

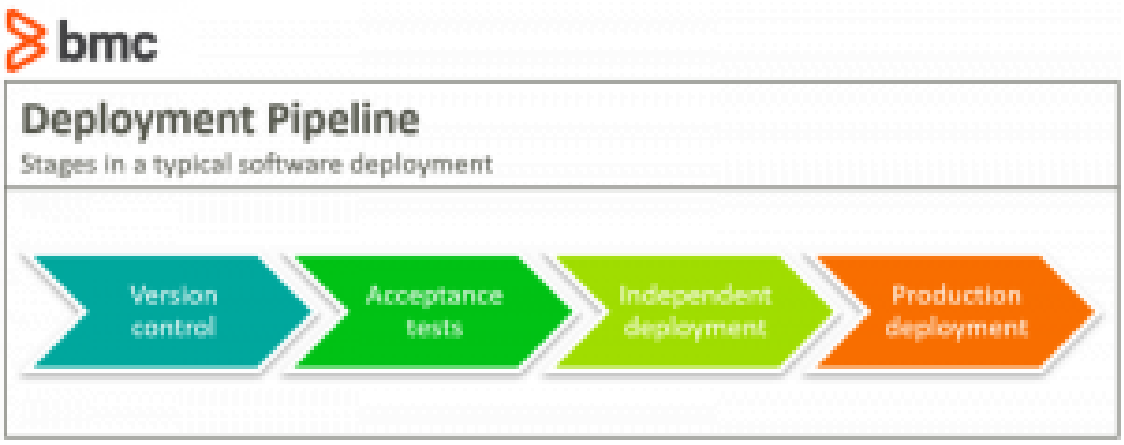
The key objective is full automation—no manual steps or changes required between any stages of the pipeline. Manual execution of repetitive tasks introduces human error that can affect deliverables and, in some cases, breach SLAs due to failed or botched deployments.

What are the stages of a deployment pipeline?

A typical deployment pipeline moves through four stages: version control, acceptance tests,

independent deployment, and production deployment.

The stages of a typical deployment pipeline are as follows.



Version Control

Software developers commit code changes into source control (e.g., GitHub). Each commit triggers the first stage of the deployment pipeline, which runs code compilation, unit tests, code analysis, and installer creation. If all steps complete successfully, executables are assembled into binaries and stored in an artefact repository for later use.

Acceptance tests

Acceptance testing is the process of running a series of automated tests against compiled code to verify it meets the predefined acceptance criteria set by the business.

Independent deployment

Independent deployment is the process of deploying compiled and tested artefacts onto development environments. Those environments should ideally mirror production as closely as possible, allowing the software to be functionally tested on production-like infrastructure before any further automated or manual testing.

Production deployment

Production deployment is typically managed by the Operations or DevOps team. It follows a similar process to independent deployment but targets live production servers. To minimize risk, this stage commonly uses blue/green deployments or canary releases—both of which enable zero-downtime releases and straightforward version rollbacks if unexpected issues arise. Where zero-downtime deployment isn't available, release windows are agreed with the business in advance.

What is CI/CD and how does it relate to deployment pipelines?

CI/CD—continuous integration and continuous delivery—is the practice that makes deployment pipelines fast, reliable, and repeatable.

Continuous integration (CI) is the practice of developers committing code to a version-controlled repository multiple times per day. Automated build pipelines triggered by each check-in enable fast,

precise error detection. The key benefits of CI include: smaller changes integrate more easily into large codebases; bugs are caught early and are cheaper to fix; code quality is validated consistently; and fewer integration issues allow for rapid delivery.

Continuous delivery (CD) is the process that allows development and operations engineers to deliver bug fixes, features, and configuration changes to production reliably, quickly, and sustainably. Key benefits include lower-risk releases enabled by blue/green and canary deployments, faster feature and bug-fix delivery, and cost savings through smaller iterative release batches that incorporate user feedback sooner.

What are blue/green deployments?

A blue/green deployment reduces risk and downtime by maintaining two mirrored production environments—one designated "blue," one designated "green"—with only one serving live traffic at any given time.

During a deployment, software is released to the non-live environment while production traffic continues uninterrupted. Once all tests pass, traffic routing switches to the newly updated environment. In the next deployment cycle, the roles reverse: the previously live environment becomes the staging target.

What are canary deployments?

A canary deployment differs from a blue/green deployment in that it does not require two full duplicate environments running in parallel. Instead, a canary release rolls out a new version to a specific number or percentage of users or servers, enabling live production testing before a full rollout.

The primary benefit of canary releases is early failure detection: if issues arise, changes can be rolled back quickly, limiting the number of affected users or services.

What tools can you use to build a CI/CD pipeline?

Several widely used tools support building reliable, robust CI/CD pipelines—all of which offer free tiers to get started:

- Jenkins
- Azure DevOps (formerly VSTS)
- CodeShip

In summary, CI is the automated process that enables software development teams to check in and verify the quality of their code. CD allows development and operations teams to reliably and efficiently deliver new features and bug fixes to end users in an automated fashion.

Frequently asked questions

What is the difference between a deployment pipeline and a CI/CD pipeline?

The terms are often used interchangeably, but they differ in scope. A CI/CD pipeline specifically covers continuous integration (automated build and test on code commit) and continuous delivery

(automated release process). A deployment pipeline is the broader end-to-end system that moves code from source control through testing environments to live production—CI/CD is one approach to implementing it.

What happens if a stage in the deployment pipeline fails?

If any stage fails—such as a unit test, code analysis check, or acceptance test—the pipeline halts and the change is not promoted to the next stage. The developer is notified so the issue can be fixed and the pipeline re-triggered. This gate-based approach prevents broken code from ever reaching production.

How does a deployment pipeline reduce deployment risk?

By automating every stage of the release process and enforcing quality gates between stages, a deployment pipeline eliminates the manual errors that cause most deployment failures. Blue/green deployments and canary releases further reduce risk by enabling zero-downtime releases and instant rollbacks when unexpected issues occur.

What is an artefact repository and why does a pipeline need one?

An artefact repository is a storage location for compiled, tested build outputs—binaries, packages, or container images. The deployment pipeline stores verified artefacts there after the build stage so that downstream stages use the exact same validated build rather than recompiling from source, ensuring consistency across environments.

Can a deployment pipeline work without containerisation?

Yes. Containerisation is a common tool category in deployment pipelines but is not a requirement. Pipelines predate containerisation. That said, containers make environment parity between development, staging, and production significantly easier to achieve, reducing "works on my machine" deployment failures.

Related reading

- BMC DevOps Blog
- DevOps Guide, a series of 30+ articles on DevOps
- Continuous Delivery vs Deployment vs Integration: What's the Difference?
- Deploying vs Releasing Software: What's The Difference?
- Testing Automation Explained: Why & How To Automate Testing
- Shift Left Testing: What, Why & How To Shift Left

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