CI/CD

Continuous Integration / Continuous Delivery or Deployment

What is CI/CD?

CI/CD angimais a method or frequently delivering applications to customers by automating the stages of app development.

When to Use CI/CD

For projects requiring rapid and reliable software delivery involving agile methodologies, or having need for manual tasks

Why Use CI/CD

- Faster release new features
- Improved code quality
- Early detection of defects
- Enhanced collaboration among teams

How CI/CD Works

Continuous integration (CI) automates the process of building and testing code changes

Continuous delivery or Depeloyment (CD) automates the release of code to production or other environments

CI

Focus on building and testing code

CD

Focus on deploying code changes

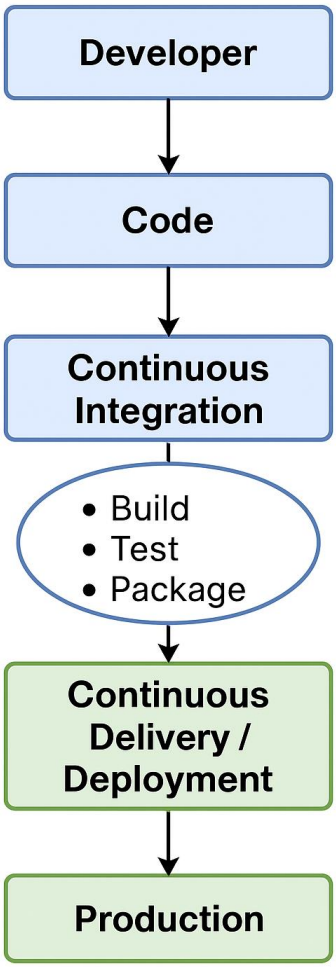

Fig : CI/CD Workflow

What is CI/CD?

CI/CD stands for:

CI: Continuous Integration

CD: Continuous Delivery or Continuous Deployment

Together, CI/CD is a set of practices and tools that enable development teams to deliver code changes more **frequently, reliably, and automatically**.

What is Continuous Integration (CI)?

Continuous Integration is a software development practice where developers **frequently commit** (push) code changes to a shared repository. Each change is automatically tested and validated through a **CI pipeline**.

Goals of CI:

Detect bugs early

Improve code quality

Automate testing

Avoid integration problems

How CI works:

1. Developer pushes code to a repository (like GitHub/GitLab/Bitbucket).
2. A CI tool (like Jenkins, GitHub Actions, GitLab CI, CircleCI) gets triggered.
3. The tool:

    Pulls the new code

    Installs dependencies

    Runs unit tests / lint checks

    Optionally builds the application (e.g., .jar, .zip, Docker image)

4. If all checks pass → the code is ready for delivery.

What is Continuous Delivery (CD)?

Continuous Delivery ensures that code is **automatically prepared for release** to a production-like environment **after CI succeeds.** The code is not deployed automatically, but it's always ready to be deployed with a **manual approval.**

Goals of CD:

Ensure deployment-ready code at all times

Enable faster, safer releases

Reduce manual error

What is Continuous Deployment (CD)?

Continuous Deployment goes one step further than Continuous Delivery. It **automatically deploys** every change that passes CI **to production** without manual intervention.

Goals of Continuous Deployment:

Fully automate the software delivery process

Reduce time to market

Allow real-time feature releases

Key Differences Between CI and CD

Feature	Continuous Integration (CI)	Continuous Delivery (CD)	Continuous Deployment (CD)
Purpose	Automate build & test process	Automate release process (till staging)	Automate full deployment (till production)
Manual Approval	No	Yes (for production)	No
Main Focus	Code quality & test validation	Deployment readiness	Full automation to production
Tools	Jenkins, GitHub Actions, CircleCI	Spinnaker, ArgoCD, Octopus	AWS CodeDeploy, Harness, GitLab CD

When is CI/CD Used?

In **modern DevOps practices**

For **agile teams** with frequent commits

For **microservices**, **cloud-native apps**, or **containerized environments (like Docker + Kubernetes)**

When teams want **fast, safe, and automated delivery**

Why CI/CD is Important?

Faster Time to Market: Deliver features quicker

Higher Code Quality: Early bug detection through automated tests

Reduced Risk: Small, frequent updates are safer

Improved Developer Productivity: Less manual testing and deployments

Scalability: Works well for growing teams and applications

Customer Satisfaction: Users get faster updates and fixes

Common CI/CD Tools

Category	Tools
CI Tools	Jenkins, GitHub Actions, GitLab CI, CircleCI, Travis CI
CD Tools	ArgoCD, Spinnaker, Octopus Deploy, Harness
Container & Orchestration	Docker, Kubernetes
Testing	Selenium, JUnit, PyTest
Monitoring	Prometheus, Grafana, ELK Stack

Example CI/CD Workflow

Let's say a developer commits code:

1. **CI Pipeline starts:**
    1. Pull latest code
    2. Run unit tests
    3. Lint/format checks
    4. Build code
    5. Package/Dockerize

If all CI steps succeed:

2. **CD Pipeline starts:**
    1. Deploy to staging server
    2. Run integration tests
    3. Wait for manual approval (if Continuous Delivery)
    4. Auto-deploy to production (if Continuous Deployment)

Best Practices for CI/CD

Commit code frequently in small batches

Write automated tests (unit, integration, E2E)

Use feature flags for safe deployments

Ensure rollback strategies (blue/green, canary)

Secure CI/CD pipelines with secrets management

Monitor performance and errors post-deployment