CI/CD PIPELINE

- A CI/CD pipeline is an automated software development workflow that builds, tests, and deploys code changes consistently and efficiently to deliver high-quality software faster.
- It combines Continuous Integration (CI), where developers frequently merge code changes to a central repository and run automated tests to detect issues early, with Continuous Delivery (CD), which automates the delivery of tested code to an environment for deployment to users.
- **Key Concepts**

  CI (Continuous Integration):

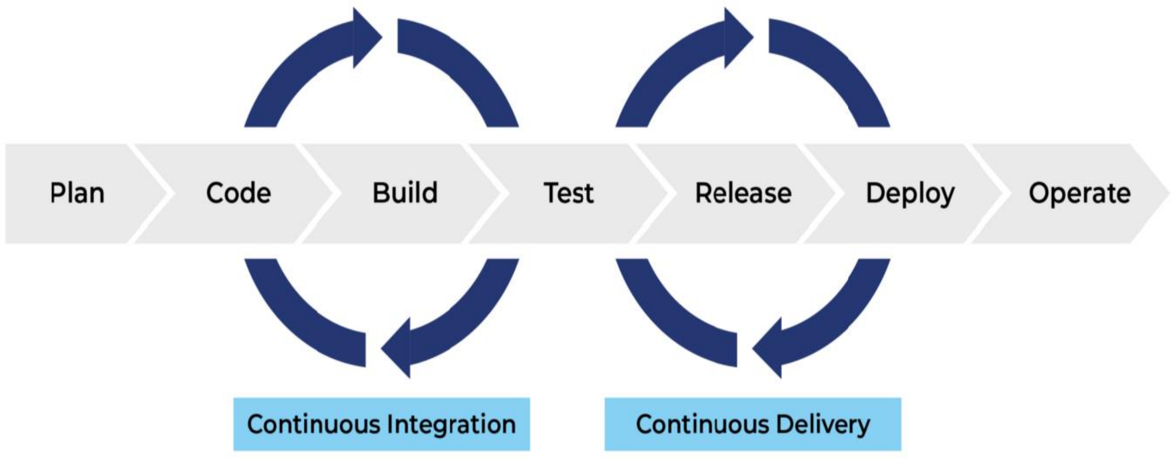
  Developers frequently merge code into a shared branch. Automated builds and tests run to catch issues early.

  CD (Continuous Delivery):

  Ensures code is always in a deployable state. Deployment up to staging can be automated, but production might need approval.

  CD (Continuous Deployment):

  Extends delivery — every passing build is automatically pushed to production with no manual step.



Continuous Integration (CI)

- Detect issues early by integrating code often.
- Developers commit changes frequently (daily or multiple times a day).
- A CI server (Jenkins, GitHub Actions, GitLab CI, etc.) automatically:
    1. Pulls the new code.
    2. Builds the application.
    3. Runs automated tests.
    4. Reports success/failure to the team.

Example:
If 5 developers work on different modules of an app, CI ensures their code integrates without breaking the build. If someone's code causes a test failure, it's detected immediately.

Continuous Delivery (CD – Delivery)

- Always keep the software in a ready-to-release state.
- After passing CI, the pipeline deploys to staging or test environments.
- Requires manual approval before production.
- Useful in industries where compliance or approvals are mandatory (e.g., banking, healthcare).

Example:
A new payment feature is deployed to staging for business validation. Once stakeholders approve, the team promotes it to production.

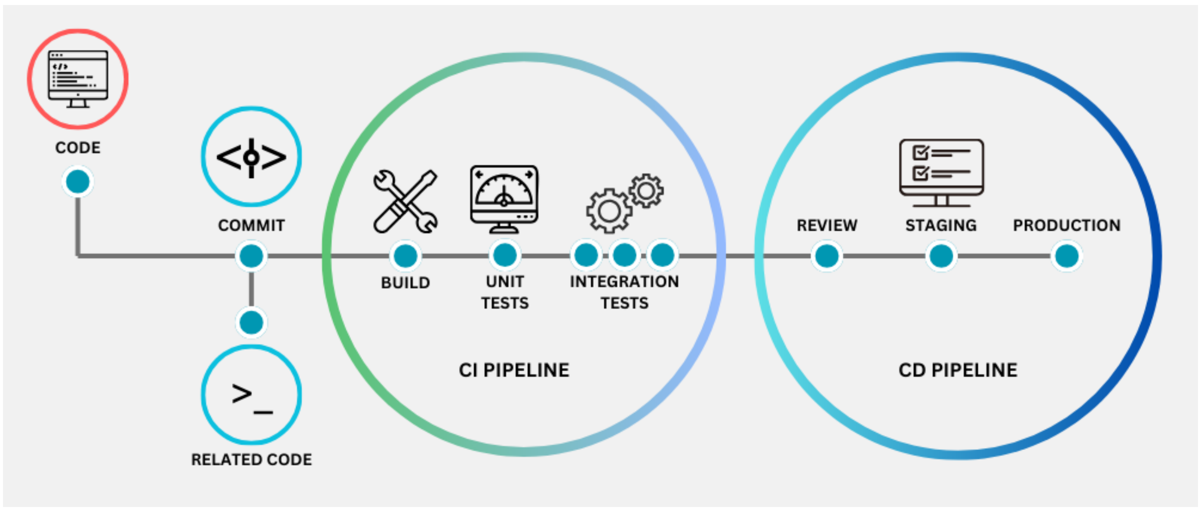
Continuous Deployment (CD – Deployment)

- Fully automate deployment to production as soon as code passes all tests.
- No manual approvals.
- Requires high confidence in test automation and monitoring.

Example:
Facebook or Amazon deploys hundreds of updates per day using continuous deployment users see new features without downtime.

CICD Pipeline Flow

1. Code Commit → Developer pushes code to Git.
2. Build → Code compiled, Docker image created.
3. Test → Unit, integration, functional, security tests run automatically.
4. Artifact Storage → Build stored in repository (Nexus, Artifactory, Docker Hub).
5. Deploy to Staging → Application deployed in staging/UAT.
6. Approval or Auto-Deploy → Depending on Delivery or Deployment.
7. Production Deployment → Live application updated.
8. Monitoring & Feedback → Metrics collected, bugs/alerts reported.



Benefits of CICD Pipelines

- Faster feature delivery.
- Reduced manual intervention.
- Immediate feedback for developers.
- Higher software quality.
- Easier collaboration between Dev, QA, and Ops teams.
- Enables DevOps culture.

Challenges in CICD

- Flaky Tests: Unreliable test cases can cause false failures.
- Security Risks: Automatic deployments may push vulnerabilities if testing is weak.
- Complexity: Setting up pipelines for microservices can be complicated.
- Cost: Running automated builds/tests for every commit can consume resources.