CI/CD

1. Introduction

CI/CD (Continuous Integration and Continuous Deployment/Delivery) forms the backbone of modern DevOps. It automates software delivery processes, enabling teams to build, test, and deploy applications efficiently and consistently. By integrating frequently and deploying continuously, teams can detect issues early, ensure reliability, and reduce manual intervention.

- **Continuous Integration (CI)** ensures code changes from multiple developers integrate seamlessly into a shared repository through automated builds and tests.

- **Continuous Deployment (CD)** takes it a step further by automatically deploying validated builds to production environments.

Together, these practices improve release velocity, reduce human errors, and promote software stability.

2. Continuous Integration (CI)

Objectives

- Detect integration issues early.
- Automate builds and tests.
- Maintain code quality through consistent validation.

Workflow

1. Developer commits changes to the main branch.
2. CI tool triggers an automated build.
3. Pipeline runs unit tests, integration tests, and code quality checks.
4. Reports and logs are generated, providing instant feedback.

Common CI Tools

- **Jenkins**: Open-source and highly customizable.
- **GitHub Actions**: Ideal for GitHub-based projects.
- **GitLab CI/CD**: Integrated within GitLab repositories.
- **CircleCI**: Scalable cloud-native CI solution.

3. Continuous Deployment (CD)

Objectives

- Automate deployment of tested builds.
- Maintain environment consistency.
- Reduce time between commit and production release.

Workflow

1. Code passes CI pipeline.
2. Artifacts (binaries, Docker images) are stored in a registry.
3. Deployment pipelines automatically promote builds through environments (dev → staging → prod).
4. Monitoring tools validate performance and stability.

Deployment Strategies

- **Blue-Green Deployment**: Maintain two identical environments; switch traffic once the new version is verified.
- **Canary Deployment**: Release incrementally to small user groups to minimize risk.
- **Rolling Deployment**: Gradually replace old instances with new ones.

4. CI/CD Pipeline Architecture

Stages

1. **Source**: Trigger via code commits or PR merges.
2. **Build**: Compile and package source code.
3. **Test**: Run automated tests to validate functionality.
4. **Artifact Management**: Store build outputs securely.
5. **Deploy**: Automate environment rollouts.
6. **Monitor**: Validate and observe deployed services.

Toolchain Overview

Function	Tools
Version Control	GitHub, GitLab, Bitbucket
Build & Test	Jenkins, GitHub Actions, GitLab CI
Artifact Storage	DockerHub, AWS ECR, JFrog Artifactory

Infrastructure as Code	Terraform, Ansible, CloudFormation
CD/GitOps	ArgoCD, Spinnaker, FluxCD
Monitoring	Prometheus, Grafana, Datadog

---

5. CI/CD Best Practices

1. **Automate Everything** – Builds, tests, security scans, and deployments.
2. **Fail Fast** – Catch and fix issues early.
3. **Keep Pipelines Modular** – Separate CI from CD for flexibility.
4. **Leverage IaC** – Manage infra declaratively for consistency.
5. **Enforce Security Checks** – Include SAST, DAST, and dependency scanning.
6. **Enable Observability** – Integrate logs, metrics, and alerts.
7. **Support Rollbacks** – Always plan automated rollback strategies.

6. Example: Modern Cloud-Native CI/CD Setup

Stack Example

- **CI**: GitHub Actions handles builds, tests, and image creation.
- **CD**: ArgoCD automates deployment to Kubernetes clusters.
- **IaC**: Terraform provisions environments.
- **Security**: Trivy and Snyk scan for vulnerabilities.
- **Monitoring**: Prometheus, Grafana, and Loki for observability.

End-to-End Flow

Developer Commit → GitHub Actions (Build/Test/Scan) → Push Docker Image →
ArgoCD detects manifest update → K8s deployment → Canary rollout → Monitoring & Alerts

7. Advantages of CI/CD

- **Speed**: Rapid feedback and deployment cycles.
- **Quality**: Automated validation improves code reliability.
- **Consistency**: Same pipeline runs across all environments.
- **Collaboration**: Streamlines developer-ops coordination.
- **Scalability**: Easily extendable to multiple microservices.