1. What is CI/CD?

Continuous Integration (CI)

- Developers frequently **merge code** into a shared repository (like GitHub or GitLab).
- Every push triggers **automated builds and tests**.
- Goal: Detect integration issues early.

✅ Example:
A developer pushes new Python code → Jenkins runs unit tests automatically.

Continuous Delivery (CD)

- Code that passes CI is automatically **deployed to staging or pre-production**.
- Manual approval is needed before going to production.

Continuous Deployment

- Every successful change **automatically goes to production** without manual approval.

✅ Summary:

Stage	Automation	Human Approval
CI	Build + Test	No
CD (Delivery)	Staging Deployment	Yes
Continuous Deployment	Production Deployment	No

🔧 2. Role of Git in CI/CD

Git is the **foundation** for CI/CD. It provides:

- Version control

- Collaboration
- Branching workflows
- Webhooks (to trigger pipelines)

Branching Strategies:

Branch Type	Purpose
main/master	Stable production-ready code
develop	Integration branch for next release
feature/	Used by developers for new features
release/	Used for preparing new releases
hotfix/	For urgent production fixes

🧩 Git Workflow Best Practices:

- Use **Pull Requests (PRs)** for code review.
- Keep **commits small** and meaningful.
- Use **semantic commit messages** (e.g., `feat: add login API`).

⚙️ 3. Build Tools & Automated Builds

Build tools compile and package your source code automatically.

Language	Build Tool	Example Command
Java	Maven / Gradle	`mvn clean install`
JavaScript	npm / yarn	`npm run build`

Python	setuptools / tox	`python setup.py build`

During the build:

- Dependencies are downloaded.
- Code is compiled.
- **Unit tests** run automatically.
- Artifacts (build outputs) are generated.

📦 4. Artifacts & Repositories

What are Artifacts?

Artifacts are the **build outputs** (JARs, Docker images, ZIPs, etc.) generated after compilation or packaging.

Repositories

They store and version artifacts:

- **Nexus / JFrog Artifactory:** for binary files like `.jar`, `.zip`, `.war`
- **Docker Registry:** for Docker images (e.g., `nginx:1.25`)
- **GitHub Packages:** for small-scale repos

Versioning & Tagging

Example:

- Application v1.2.3 → Git tag `v1.2.3`
- Docker image → `myapp:1.2.3`

🧪 5. Testing & Automation

Types of Tests:

Type	Scope	Example
Unit	Single function	pytest, JUnit
Integration	Multiple modules	Postman, pytest
End-to-End (E2E)	Full user flow	Selenium, Cypress

Frameworks:

- **Java:** JUnit, TestNG
- **Python:** pytest, unittest
- **Web/UI:** Selenium, Cypress

Parallel Testing:

Run tests simultaneously to reduce time (e.g., via Jenkins matrix build).

Reporting:

Use **Allure**, **JUnit XML**, or **HTML reports** for results visualization.

---

6. Deployment Strategies

Strategy	Description	Example
Rolling	Gradually update instances one by one	Update EC2s sequentially
Blue-Green	Two environments (Blue = live, Green = staging). Switch traffic when ready	AWS Elastic Beanstalk

Canary	Deploy to a small % of users first, then expand	Used by Netflix, Google

Infrastructure as Code (IaC)

Automates provisioning of servers and networks:

- **Terraform** → Cloud-agnostic (AWS, Azure, GCP)
- **AWS CloudFormation** → AWS-specific

Example (Terraform):

```
resource "aws_instance" "web" {
  ami = "ami-0abcd1234"
  instance_type = "t2.micro"
}
```

📊 7. Monitoring & Feedback

Monitoring ensures reliability and fast feedback.

Tools:

- **Prometheus:** metrics collection
- **Grafana:** visualization dashboards
- **New Relic / Datadog:** application performance monitoring
- **ELK Stack:** log aggregation (Elasticsearch + Logstash + Kibana)

✅ **Feedback loop:**
If an error occurs → alert sent → developer notified via Slack/Email → fix pushed → pipeline redeployed.

🧩 8. Popular CI/CD Tools

Tool	Description
Jenkins	Most popular, open-source CI server
GitLab CI/CD	Integrated with GitLab
CircleCI	Cloud-based CI/CD platform
GitHub Actions	Integrated directly with GitHub repos

Example: GitHub Actions YAML

```yaml
name: CI Pipeline
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install Dependencies
        run: npm install
      - name: Run Tests
        run: npm test
      - name: Build App
        run: npm run build
```

⚡ 9. Optimizing Pipelines & Security

🕒 Pipeline Performance:

- Use **caching** (dependencies, Docker layers)
- Split into **stages** (build → test → deploy)
- Use **parallel jobs**

⚠️ Avoid:

- Long-running pipelines
- Flaky (unstable) tests

🔒 Secrets Management:

Store sensitive data (tokens, passwords) securely:

- Jenkins Credentials Store
- GitHub Secrets
- Vault (HashiCorp)

🛡️ Security Scanning:

- **Snyk / Trivy:** scans for vulnerabilities in dependencies or Docker images.
- **SonarQube:** static code analysis.

🧭 10. Advanced CI/CD Use Cases

Multi-Branch Pipelines

Each Git branch has its own CI/CD workflow (useful for parallel development).

CI/CD for Microservices

Each microservice gets:

- Independent pipeline
- Separate Docker image
- Individual Kubernetes deployment

Docker + Kubernetes Integration

Example Jenkins stage:

```
stage('Deploy to Kubernetes') {
    sh 'kubectl apply -f k8s/deployment.yaml'
}
```

🎯 Final Recap

✅ **CI/CD ensures:**

- Faster, safer, repeatable deployments
- Early detection of bugs
- Continuous feedback loops

✅ **Core tools:**

Git + Jenkins/GitLab + Docker + Kubernetes + Terraform + Monitoring stack

✅ **Mindset:**

> *Automate everything — test, build, deploy, monitor.*