

Day 15: CI/CD Pipelines (Jenkins & GitHub Actions)

This module explains **why CI/CD exists**, **how pipelines are structured**, and how to implement **reliable, automated build-test-deploy workflows** used in real production systems. The focus is on **repeatability, safety, and rollback**, not just automation.

1. Background: What CI/CD Really Solves

Continuous Integration (CI)

CI ensures that: - Every code change is **automatically built and tested** - Integration issues are detected early - The main branch is always in a deployable state

Continuous Deployment / Delivery (CD)

CD ensures that: - Deployments are **automated and consistent** - Human error is minimized - Rollbacks are fast and predictable

CI/CD is not about speed alone — it is about **confidence**.

2. CI/CD Pipeline Stages (Standard Model)

Typical pipeline:

```
Commit → Build → Test → Security Scan → Deploy → Monitor
```

Each stage must be: - Deterministic - Idempotent - Observable (logs + status)

3. Jenkins Pipeline (Jenkinsfile)

3.1 Why Jenkins

- Mature and extensible
- Self-hosted (full control)
- Large plugin ecosystem

Trade-off: - Higher maintenance than SaaS CI

3.2 Jenkinsfile Structure

Jenkins pipelines are **defined as code** using a `Jenkinsfile`.

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/org/repo.git'
      }
    }

    stage('Build') {
      steps {
        sh 'pip install -r requirements.txt'
      }
    }

    stage('Test') {
      steps {
        sh 'pytest'
      }
    }

    stage('Deploy to Staging') {
      steps {
        sh './deploy.sh staging'
      }
    }
  }

  post {
    success {
      echo 'Build successful'
    }
    failure {
      echo 'Build failed'
    }
  }
}
```

✓ Checklist: Jenkins pipeline running successfully

4. GitHub Actions (Modern CI/CD)

4.1 Why GitHub Actions

- Native GitHub integration
 - Zero infrastructure management
 - Excellent for open-source and startups
-

4.2 Workflow Triggering

Common triggers: - `push` to main branch - `pull_request` - Manual (`workflow_dispatch`)

4.3 Example GitHub Actions Workflow

```
name: CI Pipeline

on:
  push:
    branches: [ main ]
  pull_request:

jobs:
  build-test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.12'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run tests
        run: pytest
```

✓ Checklist: GitHub Actions triggered on push/PR

5. Automated Testing in CI

Types of Tests in Pipeline

Test Type	Purpose	When Run
Unit tests	Logic correctness	Every commit
Integration tests	Component interaction	CI
E2E tests	User flows	Pre-release

Best Practices

- Tests must be **fast**
- Tests must be **deterministic**
- Fail fast on errors

✓ Checklist: All tests running in CI

6. Deployment Automation (Staging First)

Why Staging Environments

- Production-like testing
- Safe validation
- Catch config issues

Deployment flow:

```
CI → Deploy to Staging → Validate → Promote to Prod
```

Example Deploy Script

```
#!/bin/bash
ENV=$1

echo "Deploying to $ENV"
ssh server "cd app && git pull && systemctl restart app"
```

✓ Checklist: Automated deployment to staging

7. Rollback Strategies (Critical Topic)

Why Rollbacks Matter

Deployments **will fail**. Good systems recover fast.

Common Rollback Techniques

Strategy	Description
Git revert	Revert faulty commit
Blue-Green	Switch traffic to old version
Canary	Gradual rollout, stop on errors
Versioned artifacts	Redeploy previous build

Simple Rollback Example

```
git checkout previous_tag
systemctl restart app
```

✓ Checklist: Rollback strategy documented

8. Notifications & Observability

Why Notifications Matter

- Immediate feedback
- Faster incident response

Notification Channels

- Email
- Slack
- Microsoft Teams

GitHub Actions Example (Slack)

```
- name: Notify Slack
  uses: slackapi/slack-github-action@v1
```

```
with:
  payload: '{"text":"Build completed"}'
```

✓ Checklist: Build status notifications configured

9. Jenkins vs GitHub Actions (Comparison)

Feature	Jenkins	GitHub Actions
Hosting	Self-hosted	Managed
Setup	Complex	Simple
Flexibility	Very high	Medium
Maintenance	High	Low

10. Real Production Mindset

Strong CI/CD systems: - Treat pipelines as **code** - Enforce tests before deploy - Automate rollback - Never deploy manually

Day 15 Completion Checklist

- ☒ Jenkins pipeline running
 - ☒ GitHub Actions workflows active
 - ☒ Automated tests executed
 - ☒ Staging deployment automated
 - ☒ Notifications configured
 - ☒ Rollback strategy documented
-

Key Interview Insight

CI/CD is the backbone of modern engineering. Teams that deploy safely **outperform teams that deploy fast.**

Next strong steps: - CI/CD + Docker integration - Infrastructure as Code (Terraform) - Progressive delivery (canary releases) - End-to-end pipeline demo project