UnCodemy

CI/CD Pipeline Guide PDF

A CI/CD pipeline automatically builds, tests and delivers code after every change. A typical pipeline has stages for checkout, build, unit tests, code checks, packaging, deployment to a test environment, integration tests and a controlled release. This guide explains each stage, shows a sample configuration, and covers secrets, quality gates and failure handling.

Key takeaways

- Continuous integration merges and tests often; continuous delivery keeps software always releasable.
- Fast feedback matters: run quick checks first and slow checks later.
- Build once and promote the same artifact through environments.
- Keep secrets in a secret store, never in the repository.
- Add quality gates and an easy rollback before automating production releases.

What is CI/CD pipelines?

A CI/CD pipeline is an automated sequence of steps that integrates code changes, builds and tests them continuously, and delivers or deploys the result to environments such as testing and production.

What do CI and CD mean?

Continuous integration (CI) means developers merge small changes often, and each change triggers an automated build and tests. **Continuous delivery** means every change that passes the pipeline is ready to release, usually with a manual approval for production. **Continuous deployment** goes further and releases automatically when checks pass. The aim is fast, safe, repeatable delivery.

What stages does a pipeline have?

Stage	Purpose
Trigger	Start on push, pull request, schedule or manual run
Checkout	Get the code at the right commit
Build	Compile or package the app
Unit tests	Fast checks of small parts

Stage	Purpose
Code quality	Linting, formatting, static analysis, dependency scan
Package	Create an artifact or container image and version it
Deploy to test	Release to a test environment
Integration or end-to-end tests	Check the system as a whole
Approval	Manual or automatic gate
Deploy to production	Release with a safe strategy
Monitor	Check health and roll back if needed

Order the stages so that cheap, fast checks fail early.

What does a sample pipeline look like?

This example shows the idea in the style used by GitHub Actions; other tools use similar concepts.

```
name: ci
on:
  pull_request:
  push:
    branches: [main]
jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install dependencies
        run: npm ci
      - name: Lint
        run: npm run lint
      - name: Unit tests
        run: npm test
      - name: Build
        run: npm run build
```

A `deploy` job would depend on this job and run only on the main branch.

What are quality gates?

A quality gate is a rule that must pass before the pipeline continues: all tests pass, code coverage does not fall below an agreed level, no critical security findings, and at least one review is approved. Gates should be few, clear and fair; too many slow teams down.

How do I handle secrets?

Store passwords, tokens and keys in the pipeline's secret store or a vault, and expose them to jobs only as environment variables when needed. Never print secrets in logs. Give pipeline credentials the minimum permissions and rotate them regularly. Avoid secrets in the repository, even in old commits.

What deployment strategies exist?

Rolling: replace instances gradually. **Blue-green:** run two environments and switch traffic. **Canary:** send a small share of traffic to the new version first. **Feature flags:** deploy code switched off and enable it later. Pick a strategy based on risk and cost, and test the rollback path.

How do I keep pipelines fast?

Cache dependencies, run independent jobs in parallel, run only the tests affected by a change when your tooling allows it, use small images and split slow end-to-end tests into a nightly run. Track the pipeline duration and treat slowness as a defect.

How do I handle flaky or failing builds?

Fix or quarantine flaky tests quickly, read logs before re-running, keep builds reproducible by pinning versions and use the same container image locally and in CI. A red build should be fixed first, before new work.

What should I monitor after a release?

Error rates, latency, resource use and key business metrics. Set alerts and define a rollback trigger, for example an error rate above a set threshold for five minutes.

What are common mistakes?

Building separately for each environment, skipping tests to save time, storing secrets in code, no rollback plan, manual steps hidden outside the pipeline and pipelines nobody maintains.

How do I start a pipeline for a small project?

Begin with one job that installs dependencies, runs linting and unit tests on every pull request. Add a build step, then an automatic deploy to a test site from the main branch, then a manual approval for production. Add one stage at a time and keep each stage under a few minutes where possible. A simple pipeline that runs reliably is worth more than a complex one that nobody trusts.

How do I write a good pipeline README?

List what triggers each pipeline, the stages, how to run the same checks locally, where secrets live, who to contact when it fails and how to roll back. Link to the dashboard of recent runs. New team members can then fix a failed build without asking around.

Which metrics show a healthy pipeline?

Track lead time from commit to production, deployment frequency, change failure rate and time to restore service after a failure. These four measures, used widely in DevOps research, show whether delivery is getting faster and safer. Use them to find bottlenecks, not to rank people.

Frequently asked questions

What is a CI/CD pipeline?

It is an automated sequence that integrates code changes, builds and tests them and delivers the result to test or production environments.

What is the difference between continuous delivery and continuous deployment?

Delivery keeps every passing change ready to release, usually with manual approval, while deployment releases automatically.

Why build once and promote the artifact?

Using the same artifact in every environment proves that what you tested is what you release.

Where should pipeline secrets be stored?

In the pipeline secret store or a vault, never in the repository or logs.

How do I make a pipeline faster?

Cache dependencies, run jobs in parallel, fail fast with quick checks and move slow tests to a scheduled run.

Sources and further reading

- GitHub Actions documentation
- Jenkins documentation
- Docker documentation

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550