DEVOPS BASICS FOR WEB DEPLOYMENT

CI/CD Pipelines – Detailed Teaching Notes

Concepts • Workflow • Tools • Diagrams • Practical Example • Best Practices

Learning Outcomes

After studying this topic, learners should be able to:

- Explain DevOps and distinguish it from traditional software development and operations.
- Describe the stages of a web application's development and deployment lifecycle.
- Differentiate Continuous Integration (CI), Continuous Delivery, and Continuous Deployment.
- Explain the purpose of source control, automated builds, testing, artifacts, environments, and deployment strategies.
- Interpret a CI/CD pipeline diagram and outline a basic pipeline for a web application.
- Identify security, monitoring, reliability, and collaboration practices used in modern deployments.

1. Introduction to DevOps

DevOps is a set of practices, cultural principles, and automation techniques that brings software development (Dev) and IT operations (Ops) closer together. Its purpose is to help teams deliver software more frequently, reliably, and safely. Rather than treating development, testing, deployment, and operations as isolated activities, DevOps encourages shared responsibility across the entire software lifecycle.

In a traditional workflow, developers may complete an application and hand it to a separate operations team. Differences between development and production environments, incomplete deployment instructions, and late testing can cause delays or failures. DevOps reduces these problems through collaboration, repeatable processes, automation, continuous feedback, and infrastructure that can be configured consistently.

1.1 Main principles of DevOps

- **Collaboration and shared ownership:** Developers, testers, security specialists, and operations staff work toward common delivery goals.
- **Automation:** Builds, tests, infrastructure configuration, and deployments are automated where practical to reduce manual errors.
- **Continuous feedback:** Logs, metrics, alerts, user feedback, and test results are used to improve the application and process.
- **Small and frequent changes:** Smaller releases are easier to review, test, troubleshoot, and roll back than large releases.
- **Measurement and improvement:** Teams monitor delivery speed, reliability, quality, and service performance, then improve based on evidence.

1.2 DevOps lifecycle

The lifecycle is often represented as a loop because operational feedback informs the next planning and development cycle.

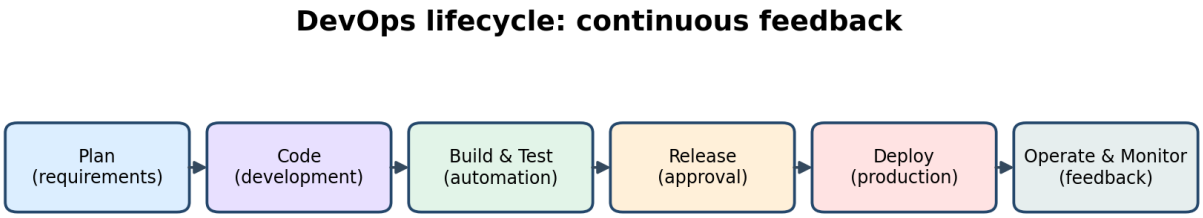


Figure 1. DevOps lifecycle with continuous feedback.

1. **Plan:** Define user needs, features, acceptance criteria, priorities, and release goals.
2. **Code:** Develop the application and store changes in a version-control repository.
3. **Build and test:** Compile or package the application and run automated checks.
4. **Release:** Prepare a version for deployment, including approvals or release metadata where required.
5. **Deploy:** Deliver the tested version to a staging or production environment.
6. **Operate and monitor:** Run the service, observe its health, respond to incidents, and collect feedback.

2. Web Application Deployment Basics

Web deployment is the process of making a web application available to users through a network, usually the internet or an organization's private network. A web application may contain a front end (the user interface), a back end (application logic and APIs), and data services (databases, object storage, or caches). Deployment places the required code, configuration, and supporting services into an environment where the application can run.

2.1 Common deployment components

- **Domain name and DNS:** DNS translates a human-readable domain name, such as example.com, into the network address of a service.
- **CDN (Content Delivery Network):** A CDN caches static resources such as images, stylesheets, and scripts closer to users to reduce latency.
- **Load balancer / reverse proxy:** Receives incoming requests and forwards them to suitable application servers; it may also terminate TLS and perform routing.
- **Web or application server:** Runs the application code and responds to browser or API requests.
- **Database and storage:** Persist user accounts, transactions, application records, files, and other data.
- **Monitoring and logging:** Collect operational information such as response time, errors, resource usage, and application events.

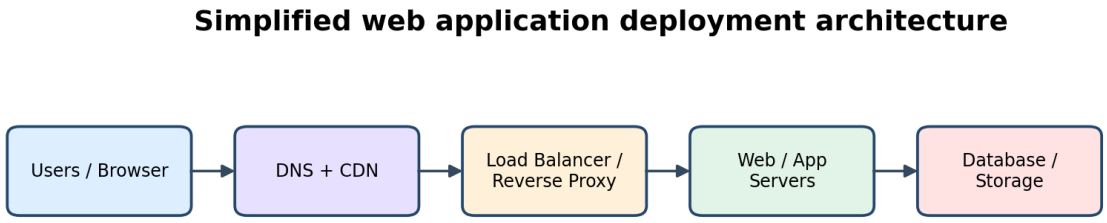


Figure 2. Simplified architecture of a deployed web application.

2.2 Deployment environments

- **Development:** Used by developers for coding and initial checks. It may contain frequently changing code and test data.
- **Testing / QA:** Used to verify functional, integration, performance, and security requirements.
- **Staging / pre-production:** Designed to resemble production and used for final validation before release.
- **Production:** The live environment accessed by real users. Availability, security, data protection, and change control are especially important.

A good practice is to keep environments as similar as possible and manage their differences through configuration, not by maintaining unrelated copies of the application.

3. What is a CI/CD Pipeline?

A CI/CD pipeline is an automated sequence of steps that takes a code change from a version-control repository through validation and packaging toward delivery or deployment. The pipeline provides a repeatable way to check whether a change is safe and suitable for release. A pipeline may stop when a test fails, a security policy is violated, or an approval is required.

3.1 Continuous Integration (CI)

Continuous Integration is the practice of frequently merging developers' code changes into a shared repository. Each change triggers automated activities such as compiling the code, running unit tests, checking code quality, and reporting results. CI helps detect integration problems early, while the change is still small and easier to fix.

3.2 Continuous Delivery

Continuous Delivery means that the software is kept in a deployable state. After CI checks pass, the pipeline can package and deploy the application to a staging or pre-production environment. A production release may require a human approval, a change window, or a business decision. The key idea is that the release process is ready and repeatable.

3.3 Continuous Deployment

Continuous Deployment goes one step further: every change that passes the required automated checks is automatically released to production, without a separate manual release approval. It requires strong automated testing, monitoring, rollback or recovery mechanisms, and confidence in the pipeline.

3.4 CI, Continuous Delivery, and Continuous Deployment compared

Aspect	Continuous Integration	Continuous Delivery	Continuous Deployment
Main focus	Integrate and validate code frequently	Keep software ready for release	Automatically release validated changes
Automated tests	Yes, central practice	Yes, plus release validation	Yes, extensive automated validation
Production release	Not implied	Usually requires approval/trigger	Automatic after pipeline gates pass
Human approval	May be used for code review	Common before production	Not required for each qualifying release

Note: Organizations use these terms in slightly different ways. The distinction above reflects their common industry meaning.

4. Typical Stages of a CI/CD Pipeline

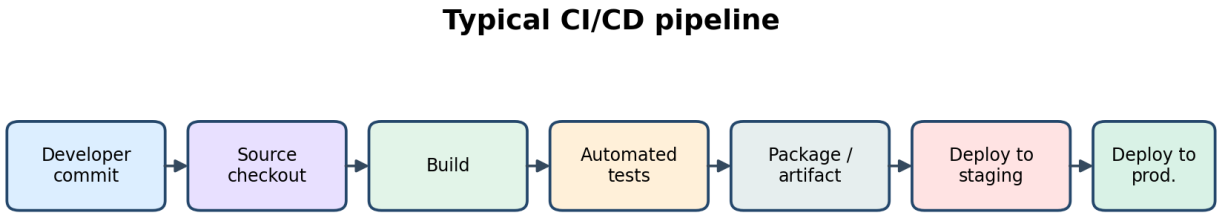


Figure 3. Typical CI/CD pipeline from a developer commit to production.

7. **Source / version control:** The source code and related files are stored in a repository. Branches and pull requests support parallel work and code review. Git is widely used for version control.
8. **Checkout and dependency installation:** The runner retrieves the required commit and installs dependencies using a lock file or controlled package versions.
9. **Build:** The application is compiled, bundled, or otherwise prepared. For a JavaScript web application, this may generate optimized HTML, CSS, and JavaScript assets.
10. **Automated testing:** Unit tests verify small units of code; integration tests check interactions between components; end-to-end tests simulate user journeys in a browser.
11. **Quality and security checks:** Static code analysis, dependency vulnerability scanning, secret detection, and software composition analysis can identify risks before release.
12. **Artifact creation:** The pipeline creates a versioned output, such as a JAR file, a container image, or a static-site bundle. An artifact should be traceable to the source commit that produced it.
13. **Staging deployment:** The same artifact is deployed to a staging environment for acceptance, smoke, and other pre-production tests.
14. **Production deployment:** The approved or automatically qualified artifact is deployed to production using a chosen deployment strategy.
15. **Monitoring and feedback:** The team observes application health and user-facing indicators, responds to alerts, and uses findings to improve future changes.

5. Common CI/CD Tools

Tools are selected according to the application's language, hosting platform, security needs, team skills, and budget. A pipeline does not require every tool listed below.

Purpose	Example tools	Typical use
Version control	Git, GitHub, GitLab, Bitbucket	Track changes, branches, reviews, and source history.
CI/CD automation	GitHub Actions, GitLab CI/CD, Jenkins, Azure Pipelines	Run build, test, scan, and deployment jobs.
Build and package	Maven, Gradle, npm, pnpm	Resolve dependencies and create application packages.
Containers	Docker, Podman	Package an application with its runtime dependencies.
Container orchestration	Kubernetes	Schedule, scale, and manage containerized workloads.
Configuration / infrastructure automation	Ansible, Terraform	Configure servers and provision infrastructure consistently.
Monitoring and logging	Prometheus, Grafana, OpenTelemetry, ELK / OpenSearch	Collect metrics, visualize service health, and investigate events.
Cloud / hosting	AWS, Microsoft Azure, Google Cloud, managed hosting	Provide compute, storage, networking, and deployment services.

6.1 Illustrative pipeline pseudocode

```
on: push to main

jobs:
  build-and-test:
    steps:
      - checkout source code
      - install locked dependencies
      - build application
      - run unit and integration tests
      - run code and dependency security checks
      - create versioned artifact

  deploy-staging:
    needs: build-and-test
    steps:
      - deploy the same artifact to staging
      - run smoke tests

  deploy-production:
    needs: deploy-staging
    steps:
      - wait for required approval (if configured)
      - deploy the same artifact to production
      - verify health checks
```

This is conceptual pseudocode, not a ready-to-run workflow file. Exact syntax depends on the CI/CD platform and the application's build system.

7. Deployment Strategies

A deployment strategy determines how a new application version is introduced to users. The right strategy depends on risk, traffic, infrastructure, and the ability to run multiple versions.

- **Recreate deployment:** The old version is stopped and the new version is started. It is simple, but may cause downtime during the transition.
- **Rolling deployment:** Instances are updated gradually. Some instances run the old version while others run the new version during the rollout. Health checks are important.
- **Blue-green deployment:** Two production-like environments are maintained. The new version is deployed to the idle environment, tested, and then traffic is switched to it. The previous environment can support a quick switch-back.
- **Canary deployment:** The new version is first exposed to a small portion of users or traffic. If metrics remain healthy, the rollout is expanded; otherwise, traffic can be returned to the stable version.

7.1 Rollback

Rollback is the process of returning to a previously working application version when a release causes serious problems. A rollback plan should identify the last known-good artifact, the method for switching traffic or replacing the release, and how database changes will be handled. Database migrations need special care because reverting application code does not automatically reverse data changes.

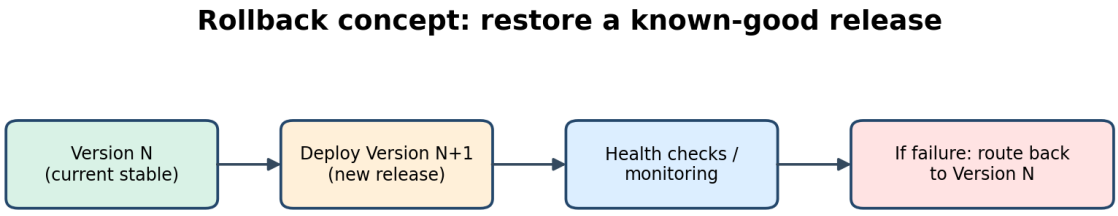


Figure 4. Conceptual rollback path after a failed release.

8. Security in CI/CD (DevSecOps)

DevSecOps integrates security into the development and delivery lifecycle instead of leaving it until the end. Security checks should be automated where possible, while risk decisions and exceptions remain documented and reviewed.

- **Protect secrets:** Do not store passwords, API keys, private keys, or tokens in source code. Use a secrets manager or protected pipeline variables.
- **Apply least privilege:** Give pipeline identities only the permissions required for their tasks. Separate permissions for development, staging, and production.
- **Scan dependencies:** Check third-party libraries and packages for known vulnerabilities and update them through a controlled process.
- **Scan source and configuration:** Use static analysis, secret detection, container image scanning, and infrastructure-as-code checks where appropriate.
- **Secure artifacts:** Use trusted registries, access controls, versioning, and artifact provenance. Avoid deploying unverified artifacts.

- **Protect deployment environments:** Use TLS, network restrictions, access logging, backups, and controlled administrative access.
- **Audit changes:** Keep records of source commits, approvals, pipeline runs, deployed versions, and security findings.

9. Monitoring, Logging, and Feedback

Deployment is not complete when files are copied to a server. The service must be observed after release to ensure that it works correctly for users. Monitoring uses measurements to show the health and performance of the system; logging records events that help explain what happened; tracing follows a request across multiple services.

- **Availability:** Whether users can reach the application and complete important actions.
- **Latency:** How long requests take, including slow requests at high percentiles such as p95 or p99.
- **Error rate:** The proportion or count of failed requests and application errors.
- **Throughput:** The number of requests or transactions handled over time.
- **Resource usage:** CPU, memory, disk, network, and database capacity.
- **Business and user indicators:** For example, successful sign-ins, completed orders, or failed form submissions.

A useful operational practice is to define service-level indicators (SLIs), service-level objectives (SLOs), and alert thresholds. Alerts should be actionable: they should identify a real condition that requires investigation rather than generate excessive noise.

10. Benefits and Challenges of CI/CD

10.1 Benefits

- Faster and more predictable delivery of application updates.
- Earlier detection of defects through automated tests and quality checks.
- Reduced manual deployment effort and fewer repetitive human errors.
- More consistent environments and repeatable release procedures.
- Improved traceability through versioned source code, artifacts, and pipeline logs.
- Quicker feedback from users and production monitoring.

10.2 Challenges

- Initial effort is required to design reliable tests and automation.
- Poorly designed pipelines can become slow, fragile, or difficult to maintain.
- Legacy applications may be difficult to package or deploy automatically.
- Secrets, permissions, dependencies, and third-party services introduce security risks.
- Teams need skills in source control, testing, infrastructure, monitoring, and incident response.
- Automated deployment without adequate checks can spread a defect quickly.

12. Key Terms

Term	Meaning
Repository	A managed location that stores source code and its change

	history.
Commit	A recorded set of changes in a version-control system.
Branch	A separate line of development used to work on changes independently.
Build	The process of transforming source code and dependencies into a runnable or distributable output.
Pipeline	An automated sequence of jobs that validates, packages, and may deploy software.
Runner / agent	A machine or execution environment that runs pipeline jobs.
Artifact	A versioned output produced by a build, such as a package or container image.
Container	A standardized unit that packages an application and its runtime dependencies.
Smoke test	A short, basic test that checks whether a deployed application is functioning at a fundamental level.
Rollback	Returning to a previous stable application release.
Infrastructure as Code	Managing infrastructure through version-controlled configuration files.
Observability	The ability to understand a system's internal state from outputs such as metrics, logs, and traces.

13. Summary

DevOps connects development and operations through collaboration, automation, and continuous feedback. In web deployment, CI/CD pipelines automate the path from a source-code change to a tested, packaged, and deployed application. Continuous Integration validates changes frequently; Continuous Delivery keeps software ready for release; and Continuous Deployment automatically releases changes that pass the required checks. Reliable pipelines combine testing, security, artifact management, controlled deployment strategies, monitoring, and recovery planning.

16. Create a small HTML/CSS/JavaScript application or use an existing classroom sample.
17. Create a Git repository and commit the source files.
18. Define a pipeline that checks out the code, validates files, runs available tests, and creates a deployable artifact.
19. Deploy the application to a staging host or local test environment.
20. Add a smoke test that checks whether the home page is reachable.
21. Document how production deployment would be approved and how a previous version would be restored.

Expected submission: a pipeline diagram, repository link or source files, screenshots of a successful pipeline run, and a short explanation of the deployment and rollback plan.