Lab 1: Visualize the DevOps Lifecycle and CI/CD Flow

Lab overview

In this lab, you will explore the DevOps lifecycle and understand how Continuous Integration (CI) and Continuous Deployment (CD) fit within the broader DevOps framework. DevOps is a cultural and technical approach that bridges the gap between development and operations teams, emphasizing collaboration, automation, and continuous improvement. This lab will focus on mapping each DevOps stage to real-world tasks, identifying the specific roles of CI and CD within the lifecycle, and creating visual representations of the pipeline flow from code commit to deployment. By the end of this lab, you will have a comprehensive understanding of the DevOps lifecycle, practical knowledge of CI/CD processes, and hands-on experience creating pipeline diagrams.

In this lab, you will:

- Understand the core stages of the DevOps lifecycle and their purposes
- Map each DevOps stage to practical, real-world development tasks
- Identify where Continuous Integration and Continuous Deployment processes occur within the DevOps lifecycle
- Create visual pipeline diagrams that illustrate the complete flow from code commit to production deployment
- Analyze the benefits and challenges of implementing CI/CD practices

Estimated completion time

40 minutes

Task 1: Understanding the DevOps lifecycle stages

In this task, you will learn about the eight core stages of the DevOps lifecycle and understand their individual purposes and interconnections.

1. Create a folder named **Lab1** on the Desktop. To do this, right-click on the Desktop, and select **New Folder**. Name the folder **Lab1**.
2. Open a text editor (Text Editor, VS Code, Nano or any preferred editor) and create a new document named **DevOps_Lifecycle_Notes.txt**.
3. Add the following DevOps lifecycle stages to your notes document:

   DevOps Lifecycle - 8 Core Stages:

   1. Plan
   - Purpose: define project requirements, user stories, and roadmap
   - Key activities: sprint planning, requirement gathering, project timeline creation

   2. Code
   - Purpose: write and develop application source code
   - Key activities: programming, code reviews, version control management

   3. Build
   - Purpose: compile source code into executable artifacts
   - Key activities: compilation, dependency management, artifact creation

   4. Test
   - Purpose: validate code quality and functionality
   - Key activities: unit testing, integration testing, automated testing

   5. Release
   - Purpose: package and prepare applications for deployment
   - Key activities: release planning, change management, approval processes

   6. Deploy
   - Purpose: install applications into target environments
   - Key activities: environment provisioning, application installation, configuration

   7. Operate
   - Purpose: manage and maintain applications in production
   - Key activities: system monitoring, performance optimization, incident response

8. Monitor
   - Purpose: track application performance and user feedback
   - Key activities: logging, metrics collection, alerting, feedback analysis

4. Explanation of the DevOps lifecycle:
   - The DevOps lifecycle is **cyclical and continuous**, meaning teams constantly iterate through these stages.
   - Each stage **feeds into the next**, creating a continuous flow of development and improvement.
   - **Automation** is key to making this lifecycle efficient and reliable.
   - **Collaboration** between development and operations teams occurs throughout all stages.
5. Save your notes document in the Lab1 folder.

> **Challenge**
>
> Research and add one specific tool commonly used in each DevOps stage (e.g., Jira for PLAN, Git for CODE, etc.).

Task 2: Mapping real-world tasks to DevOps stages

In this task, you will create a practical mapping exercise that connects actual development scenarios to specific DevOps stages.

1. Create a new document named **RealWorld_Mapping.txt** in your Lab1 folder.
2. Consider the following real-world scenario:

   Developing a new user authentication feature for an e-commerce website
3. Map this scenario to each DevOps stage by adding the following content to your document:

   Real-world scenario: e-commerce user authentication feature
   - Plan stage:
     - Task: create user stories for login/registration functionality
     - Real-world activity: product managers define authentication requirements
     - Deliverable: sprint backlog with authentication user stories
     - Timeline: 2-3 days for planning and requirement analysis
   - Code stage:
     - Task: implement authentication APIs and frontend components
     - Real-world activity: developers write login/registration code
     - Deliverable: source code for authentication modules
     - Timeline: 1-2 weeks for development

- Build stage:
  - Task: compile authentication code into deployable packages
  - Real-world activity: automated build system creates application artifacts
  - Deliverable: compiled application with authentication features
  - Timeline: 10-15 minutes for automated build process
- Test stage:
  - Task: execute authentication functionality tests
  - Real-world activity: QA team runs automated and manual tests
  - Deliverable: test reports confirming authentication works correctly
  - Timeline: 2-3 days for comprehensive testing
- Release stage:
  - Task: prepare authentication feature for production deployment
  - Real-world activity: release manager creates deployment package
  - Deliverable: approved release candidate with authentication feature
  - Timeline: 1 day for release preparation and approval
- Deploy stage:
  - Task: install authentication feature to production environment
  - Real-world activity: DevOps team deploys to live e-commerce site
  - Deliverable: live authentication functionality for users
  - Timeline: 30 minutes to 2 hours for deployment
- Operate stage:
  - Task: ensure authentication system runs smoothly in production
  - Real-world activity: operations team monitors system performance
  - Deliverable: stable, functioning authentication for all users
  - Timeline: ongoing 24/7 operational support
- Monitor stage:
  - Task: track authentication usage and performance metrics
  - Real-world activity: analyze login success rates, response times
  - Deliverable: performance dashboards and user feedback reports
  - Timeline: continuous monitoring with weekly analysis reports

4. Key insights from real-world mapping:
   - Each stage has **specific deliverables** that feed into the next stage
   - **Timelines vary significantly** between stages (minutes for builds vs. weeks for development)
   - **Multiple teams** are involved across different stages
   - **Automation** can significantly reduce timeline for BUILD, TEST, and DEPLOY stages

Challenge

Create your own real-world mapping for **Adding a shopping cart feature to a mobile app** and identify which stages could benefit most from automation.

Task 3: Identifying CI and CD within the DevOps lifecycle

In this task, you will learn where Continuous Integration and Continuous Deployment fit within the DevOps lifecycle and understand their specific roles.

1. Create a new document named CI_CD_Integration.txt in your Lab1 folder.
2. Add the following content to understand CI and CD positioning:

   Continuous Integration (CI) and Continuous Deployment (CD) in DevOps

   CONTINUOUS INTEGRATION (CI):

   Primary stages: CODE → BUILD → TEST

   Purpose: automatically integrate code changes and validate quality

   CI process flow:

   1. Developer commits code to repository (CODE stage)
   2. Automated build system compiles code (BUILD stage)
   3. Automated tests run to validate changes (TEST stage)
   4. Results reported back to development team

   CI benefits:

   - Early detection of integration issues
   - Faster feedback for developers
   - Reduced manual effort in build and test processes
   - Higher code quality through automated validation

   CONTINUOUS DEPLOYMENT (CD):

   Primary stages: RELEASE → DEPLOY → OPERATE → MONITOR

   Purpose: automatically deploy validated changes to production

CD process flow:

1. Successful CI triggers release preparation (RELEASE stage)
2. Automated deployment to staging/production (DEPLOY stage)
3. Automated monitoring begins (OPERATE stage)
4. Performance metrics collected (MONITOR stage)

CD benefits:

- Faster time-to-market for new features
- Reduced deployment risks through automation
- Consistent deployment processes
- Rapid rollback capabilities when issues occur

CI/CD INTEGRATION POINTS:

- CI output becomes CD input (seamless pipeline)
- Automated quality gates between CI and CD
- Feedback loops from CD monitoring back to CI/planning
- Shared infrastructure and tooling across CI/CD processes

3. Now map CI/CD to your e-commerce authentication example:

E-commerce authentication feature - CI/CD flow:

CONTINUOUS INTEGRATION:

1. Developer commits authentication code to Git repository
2. Jenkins/GitHub Actions automatically builds the application
3. Automated tests run (unit tests, integration tests, security tests)
4. Code quality checks performed (SonarQube, ESLint)
5. Results sent to development team via Slack/email

CONTINUOUS DEPLOYMENT:

1. Successful CI triggers deployment pipeline
2. Application deployed to staging environment automatically
3. Smoke tests run in staging environment
4. Upon approval, automatic deployment to production
5. Monitoring dashboards track authentication performance

6. Alerts configured for authentication failures or performance issues

INTEGRATION BENEFITS FOR THIS FEATURE:

- Authentication bugs caught within minutes of code commit
- Security vulnerabilities detected before production
- Consistent deployment process reduces human error
- Real-time monitoring ensures authentication reliability

4. Understanding CI/CD relationship:
   - CI focuses on code quality and integration validation
   - CD focuses on deployment automation and production reliability
   - Together, they create a fully automated pipeline from code to production
   - Feedback loops ensure continuous improvement of both code and processes
5. Save the document.

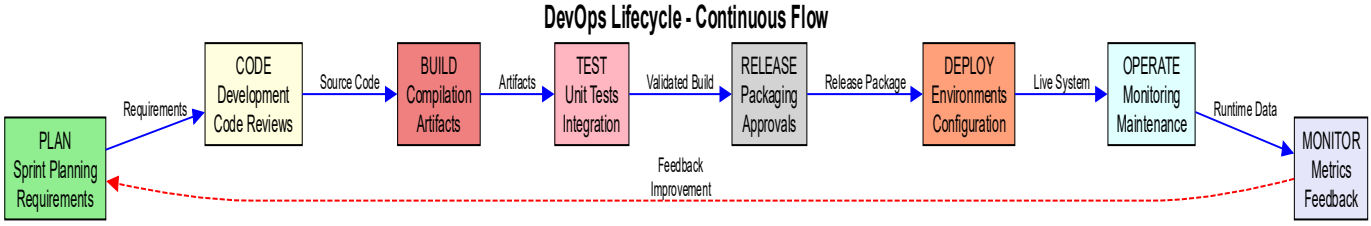
Challenge

Identify three specific tools commonly used for CI (e.g., Jenkins) and three for CD (e.g., Kubernetes), and explain how they integrate together.

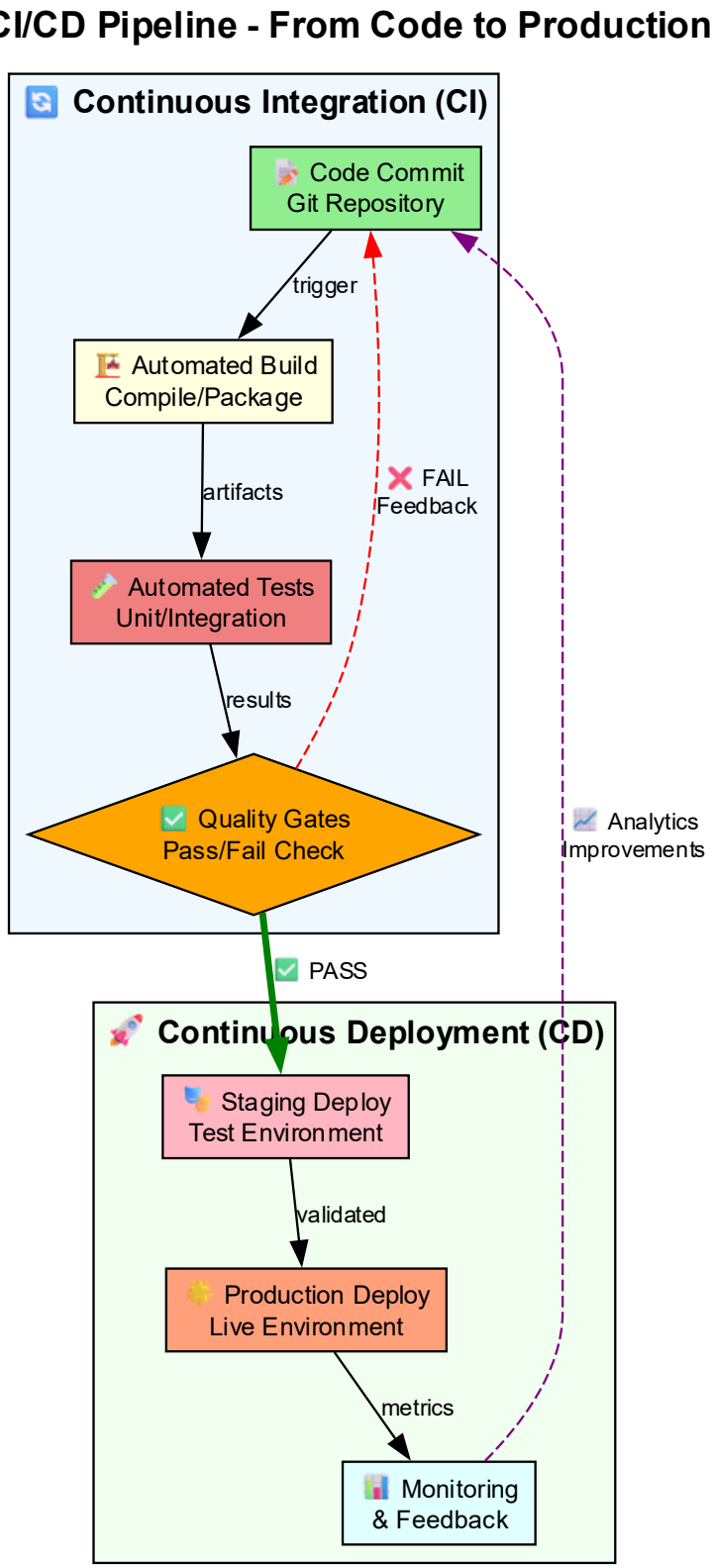
Task 4: Creating pipeline diagrams

In this task, you will create visual representations of CI/CD pipelines using simple diagramming techniques and understand the flow from code commit to deployment.

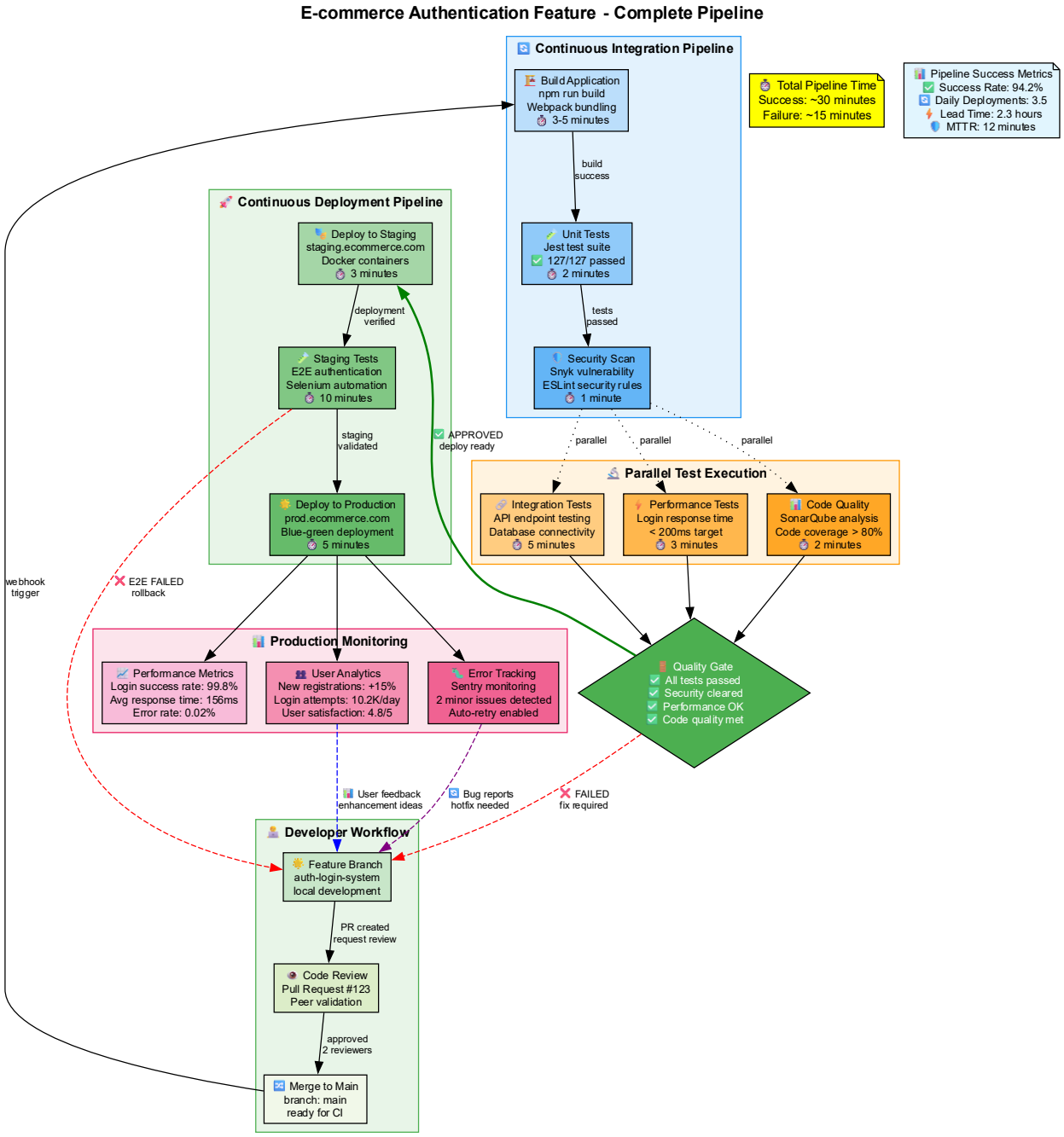
1. Open a drawing application on your computer (Draw.io, MS Paint, Lucidchart, or even PowerPoint - note: Draw.io bookmark is available via the Firefox browser).
2. Create your first diagram showing DevOps Lifecycle.



3. Create a second diagram showing CI/CD Pipeline Detail.



4. Create a third diagram for **Real-World Pipeline Example**.



5. Diagram analysis:
   - **Parallel processes** can occur simultaneously (testing phases)
   - **Quality gates** prevent faulty code from progressing
   - **Feedback loops** provide information for improvement
   - **Automation** reduces manual intervention and human error

6. Save your diagrams (as images or text files) in the Lab1 folder.

Challenge

Create a diagram showing what happens when a test fails in the CI pipeline, including the feedback loop back to developers.

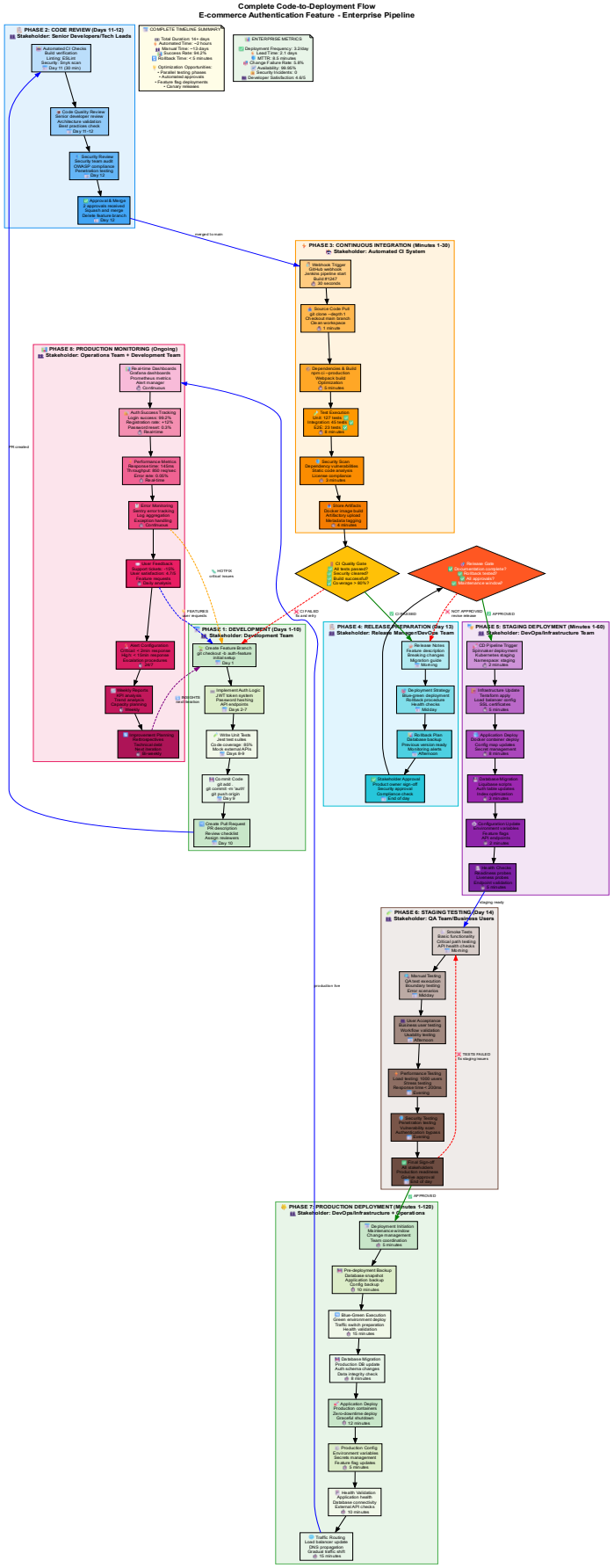
Task 5: Illustrating complete code-to-deployment flow

In this task, you will create a comprehensive flow diagram that shows the complete journey from a developer's code commit to production deployment, including all stakeholders and decision points.

1. Create a new diagram showing Complete Pipeline Flow.
2. Document the complete flow with stakeholders and timeframes.
3. Key insights from complete flow:
   - Total timeline: 14+ days from start to production
   - Automated portions: CI/CD pipeline (minutes to hours)
   - Manual portions: Code review, testing, approvals (days)
   - Critical success factors: Automation, clear gates, stakeholder communication
   - Risk mitigation: Multiple testing phases, gradual rollout, rollback capabilities
4. Save your document.

Challenge

Calculate the percentage of time spent in automated vs. manual processes and identify opportunities to reduce the overall timeline through additional automation.



Lab review

1. Which phase of the DevOps lifecycle does Continuous Integration (CI) primarily span across?
   A. PLAN → CODE → BUILD
   B. CODE → BUILD → TEST
   C. TEST → RELEASE → DEPLOY
   D. DEPLOY → OPERATE → MONITOR

STOP

You have successfully completed this lab.