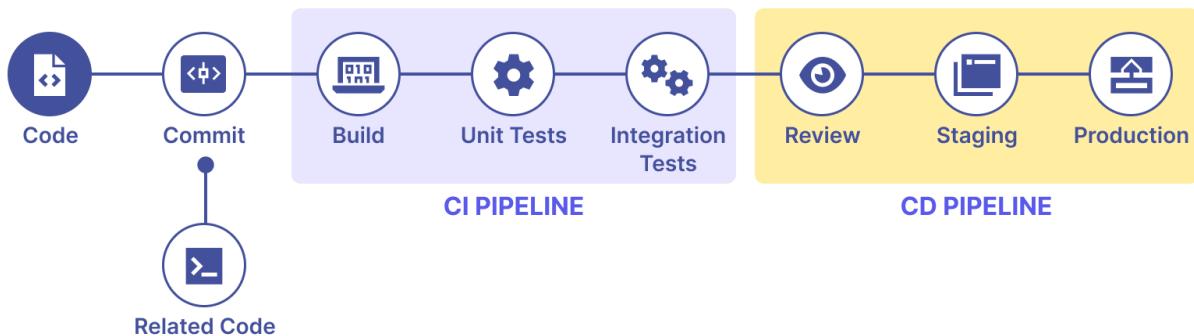


# Real-Time Corporate DevOps Workflow

## Understanding Real End-to-End CI/CD Workflows

A real-time CI/CD workflow ensures the seamless integration, testing, deployment, and monitoring of applications. Here's how it typically operates:



### 1. Code Commit:

- Developers push code to a version control system (e.g., GitHub, GitLab).
- Code is managed with branching strategies like GitFlow or trunk-based development.

### 2. Continuous Integration (CI):

- CI tools (e.g., Jenkins, GitHub Actions) fetch the latest code.
- Automated builds compile the code, run unit tests, and perform static code analysis.

### 3. Artifact Management:

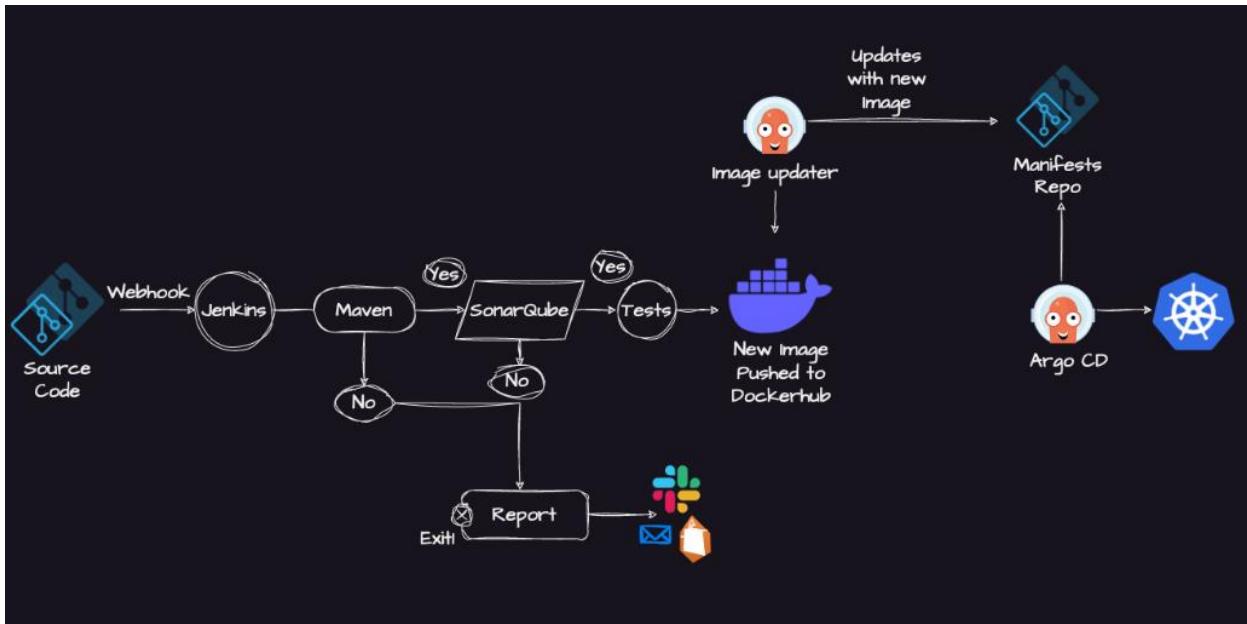
- Successfully built artifacts are stored in artifact repositories like Nexus, Artifactory, or Docker Hub.

### 4. Continuous Deployment (CD):

- Tested artifacts are deployed automatically to staging or production environments.
- Tools like Ansible, Terraform, or Kubernetes manage infrastructure and deployment processes.

### 5. Monitoring & Feedback:

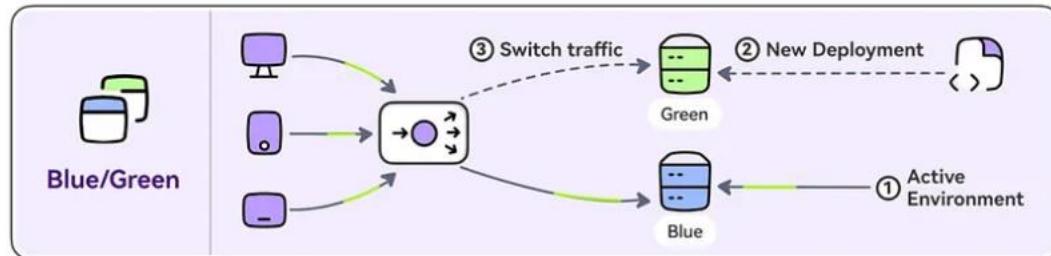
- Monitoring tools (e.g., Prometheus, Grafana) track application health.
- Alerts and logs are integrated with systems like Slack or PagerDuty for incident management.



## Deployment Strategies

To ensure smooth deployments with minimal risk, several strategies are employed:

### 1. Blue-Green Deployment:

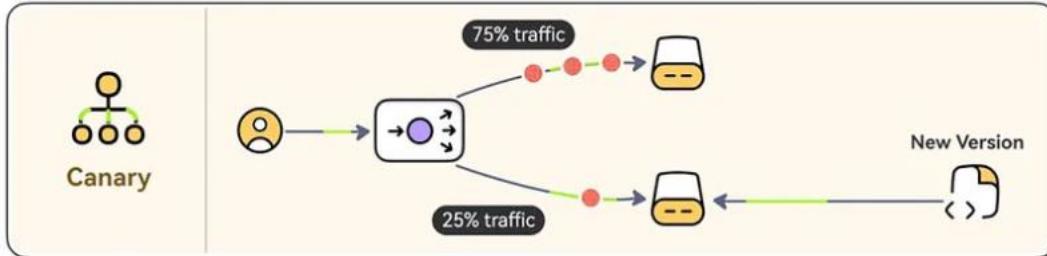


- Two environments ("Blue" for live traffic, "Green" for the new version) are maintained.
- Traffic is switched to the Green environment after successful testing.
- Rollbacks are simple: redirect traffic back to the Blue environment.

**Advantages:** Zero downtime, easy rollback.

**Use Case:** Critical production systems needing high availability.

## 2. Canary Deployment:

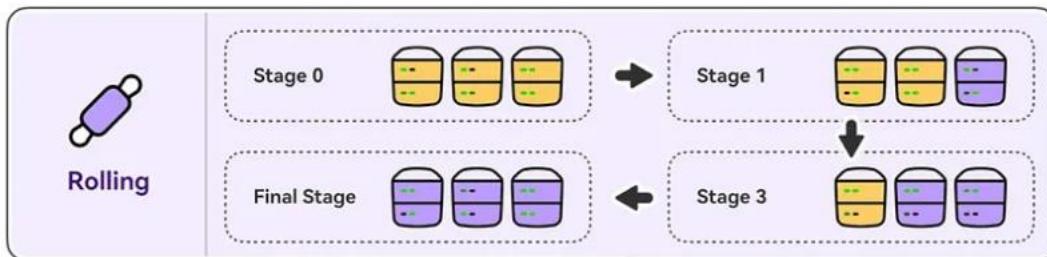


- The new version is deployed to a small subset of users initially.
- After monitoring and validation, the deployment is expanded incrementally.

**Advantages:** Controlled rollout, live user feedback.

**Use Case:** Gradual updates for complex systems or user behavior testing.

## 3. Rolling Deployment:



- Updates are deployed incrementally across servers or containers.
- At any point, some instances run the old version while others run the new one.

**Advantages:** No downtime, gradual transition.

**Use Case:** Dynamic environments like Kubernetes clusters.

## 4. Other Strategies:

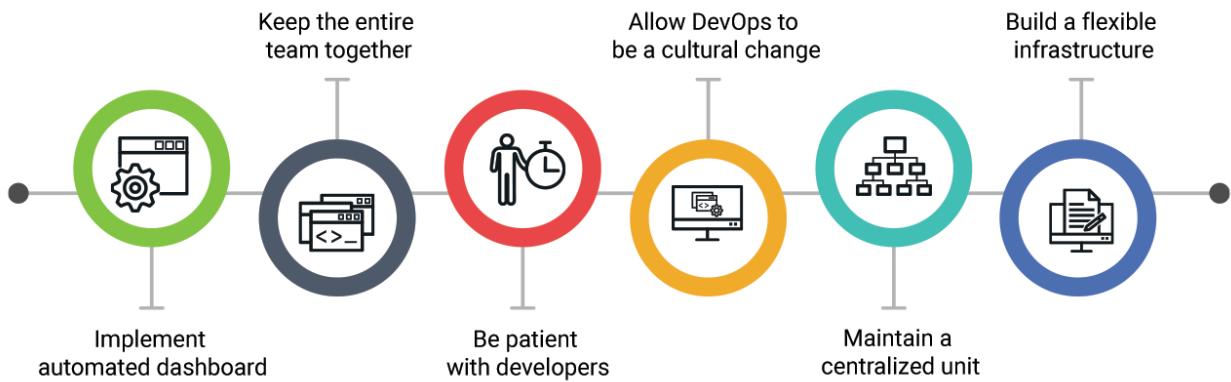
- **Recreate Deployment:**
  - Shut down the old version entirely before deploying the new one.
  - **Drawback:** Causes downtime.
- **A/B Testing:**
  - Split users into groups to test different application versions.
  - **Use Case:** Experimentation and feature validation.

- **Shadow Deployment:**
    - Mirror traffic to a new version without impacting live users.
    - **Use Case:** Load testing and performance benchmarking.
- 

## Best Practices for DevOps Workflows



# DEVOPS BEST PRACTICES TO FOLLOW



1. **Automation:**
  - Use CI/CD pipelines to automate builds, tests, and deployments.
  - Employ IaC tools like Terraform and Ansible for consistent environments.
2. **Monitoring & Observability:**
  - Centralize logs and metrics using tools like ELK Stack and Prometheus.
  - Set up real-time alerts for faster incident response.
3. **Security Integration:**
  - Implement DevSecOps practices: vulnerability scans, compliance checks, and secrets management.
4. **Testing at Every Stage:**
  - Include unit, integration, performance, and security tests in pipelines.

## 5. Collaboration:

- Foster collaboration between development, operations, and QA teams using tools like Slack and Jira.

This workflow ensures efficient, reliable, and secure application delivery, enabling organizations to respond quickly to business and customer needs.

---

 Document by Seshu (Seshadri Reddy Bhavanasi)

 Connect with me on [LinkedIn](#)

Stay tuned for the next topic, where we'll dive deeper into **DevOps tools** and **real-world practices** to enhance your understanding of this transformative approach!