

# DevPipeline CI/CD Best Practices

**Document ID:** PRD-DP-005 **Last Updated:** 2024-02-10 **Owner:** DevPipeline Product Team **Classification:** Public

---

## Overview

This guide outlines best practices for implementing effective CI/CD pipelines using DevPipeline. Following these practices will help you achieve faster, more reliable deployments.

---

## Pipeline Design Principles

### 1. Keep Pipelines Fast

**Target:** < 10 minutes for CI, < 30 minutes for full CD

- Run tests in parallel
- Use caching aggressively
- Fail fast with ordered stages
- Skip unnecessary steps on certain branches

*# Example: Parallel test execution*

```
test:
  parallel:
    - unit-tests
    - integration-tests
    - lint
cache:
  key: $CI_COMMIT_REF_SLUG
  paths:
    - node_modules/
    - .cache/
```

### 2. Make Pipelines Reproducible

- Pin dependency versions
- Use specific image tags, not **latest**
- Avoid relying on external state
- Document environment requirements

### 3. Implement Proper Branching Strategy

#### Recommended: Trunk-Based Development

```
main (production)
  feature/xyz (short-lived, < 2 days)
```

#### Alternative: GitFlow (for longer release cycles)

```
main (production)
  develop (integration)
  release/1.0 (release preparation)
```

---

### Stage Best Practices

#### Build Stage

**Do:** - Build once, deploy many - Generate build artifacts - Tag builds with commit SHA - Store build metadata

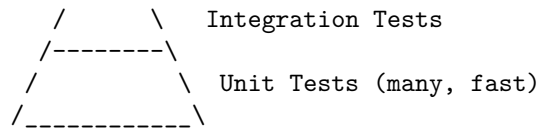
**Don't:** - Rebuild for each environment - Include secrets in build - Skip artifact versioning

```
build:
  stage: build
  script:
    - npm ci
    - npm run build
    - echo "$CI_COMMIT_SHA" > build/version.txt
  artifacts:
    paths:
      - build/
    expire_in: 1 week
```

#### Test Stage

#### Testing Pyramid:

```
  /\
 /  \
/----\ E2E Tests (few, slow)
```



**Unit Tests:** - Run on every commit - Target > 80% code coverage - Execute in < 2 minutes

**Integration Tests:** - Test component interactions - Use test databases/services  
- Run in parallel where possible

**E2E Tests:** - Test critical user paths only - Run on staging deployments - Consider running subset on PRs

```
test:
  stage: test
  script:
    - npm run test:unit -- --coverage
    - npm run test:integration
  coverage: '/Coverage: (\d+)\%/'
  artifacts:
    reports:
      coverage: coverage/lcov.info
```

## Security Scanning

Required Scans:	Scan Type	When	Tool	SAST
Every commit	DevPipeline	SAST	Dependency	Every commit
Depend- abot/Snyk	Container	Image build	Trivy	DAST
OWASP ZAP	Secret	Every commit	GitLeaks	

```
security:
  stage: test
  parallel:
    - sast
    - dependency-scan
    - secret-scan
  allow_failure: false # Block on security issues
```

## Deploy Stage

### Environment Progression:

Build	→	Dev	→	Staging	→	Production
		↓		↓		↓
		Auto		Auto		Manual/Auto

## Deployment Strategies:

Strategy	Use Case	Rollback Time
Rolling	Standard deployments	Minutes
Blue/Green	Zero-downtime required	Seconds
Canary	High-risk changes	Seconds
Feature Flag	Gradual rollout	Instant

```
deploy-production:
  stage: deploy
  environment:
    name: production
    url: https://app.example.com
  script:
    - devpipeline deploy --strategy canary --weight 10
    - sleep 300 # Monitor for 5 minutes
    - devpipeline deploy --strategy canary --weight 100
  when: manual
  only:
    - main
```

---

## Environment Management

### Environment Variables

**Hierarchy (highest to lowest priority):** 1. Pipeline variables 2. Project variables 3. Group variables 4. Instance variables

**Best Practices:** - Mark sensitive variables as protected - Use variable masking for secrets - Scope variables to environments - Never log secret values

```
variables:
  NODE_ENV: production
  # Reference secrets from vault
  DATABASE_URL: ${ secrets.DATABASE_URL }
```

### Secrets Management

**Do:** - Use DevPipeline Secrets or external vault - Rotate secrets regularly - Audit secret access - Use least-privilege access

**Don't:** - Hardcode secrets in pipelines - Log secret values - Share secrets across environments - Store secrets in repository

---

## Caching and Artifacts

### Caching

Cache dependencies to speed up pipelines:

```
cache:
  key:
    files:
      - package-lock.json
  paths:
    - node_modules/
  policy: pull-push # pull on start, push on success
```

**Cache Key Strategies:** | Strategy | Use Case | |-----|-----| | Fixed key  
| Shared across all branches | | Branch key | Isolated per branch | | File hash |  
Invalidate on dependency change |

### Artifacts

**Use artifacts for:** - Build outputs - Test reports - Coverage data - Deployment packages

```
artifacts:
  paths:
    - dist/
  reports:
    junit: test-results.xml
    coverage: coverage/cobertura.xml
  expire_in: 30 days
```

---

## Monitoring and Observability

### Pipeline Metrics

Track these metrics: - Pipeline duration (P50, P95) - Success rate - Failure reasons - Queue time - Stage duration

## Deployment Metrics

**DORA Metrics:** | Metric | Elite | High | Medium | Low | |——|——|——|  
|——|——| | Deploy Frequency | Multiple/day | Weekly | Monthly | Yearly | |  
Lead Time | < 1 hour | < 1 week | < 1 month | > 6 months | | MTTR | < 1  
hour | < 1 day | < 1 week | > 1 month | | Change Fail Rate | < 15% | 16-30%  
| 31-45% | > 45% |

## Notifications

```
notify:
  stage: notify
  script:
    - devpipeline notify slack "#deployments"
  when: always # Run on success or failure
```

---

## Troubleshooting

### Common Issues

**Pipeline stuck in pending:** - Check runner availability - Verify resource constraints - Review concurrent job limits

**Flaky tests:** - Add retry logic for external dependencies - Isolate test environments - Review test parallelism conflicts

**Slow pipelines:** - Analyze stage duration - Implement caching - Parallelize independent stages - Use smaller container images

### Debug Mode

```
variables:
  CI_DEBUG_TRACE: "true" # Verbose logging
```

---

## Compliance and Audit

### Required for Production Deployments

- All tests passing

- Security scans clean
- Code review approved
- Change request approved (if required)

## **Audit Trail**

DevPipeline maintains: - Complete pipeline history - Deployment records - Approval workflows - Variable change logs

---

## **Resources**

- DevPipeline Documentation
  - Pipeline Templates
  - Example Projects
  - Support: [devpipeline-support@novatech.com](mailto:devpipeline-support@novatech.com)
- 

*Related Documents: Quick Start Guide (PRD-DP-001), Pipeline YAML Reference (PRD-DP-010), Security Scanning Guide (PRD-DP-015)*