DevPipeline Artifact Management Guide

Document ID: PRD-DP-035 **Last Updated:** 2024-02-20 **Owner:** DevPipeline Product Team **Classification:** Public

---

Overview

DevPipeline provides built-in artifact storage for build outputs, test results, and deployment packages. This guide covers artifact creation, storage, retrieval, and best practices.

What Are Artifacts?

Artifacts are files produced during pipeline execution: - Build outputs (binaries, packages) - Test results and coverage reports - Docker images - Documentation - Logs and diagnostics

---

Creating Artifacts

Basic Artifact Upload

```
stages:
  - name: build
    steps:
      - name: Build application
        run: npm run build

      - name: Upload artifacts
        artifacts:
          - path: dist/
            name: build-output
```

Multiple Artifacts

```yaml
- name: Upload artifacts
  artifacts:
    - path: dist/
      name: build-output
    - path: coverage/
      name: test-coverage
    - path: docs/
      name: documentation
```

Conditional Artifacts

```yaml
- name: Upload on success
  if: ${{ pipeline.status == 'success' }}
  artifacts:
    - path: dist/
      name: release-build

- name: Upload on failure
  if: ${{ pipeline.status == 'failure' }}
  artifacts:
    - path: logs/
      name: debug-logs
```

Artifact Configuration

Options

```yaml
artifacts:
  - path: dist/              # Required: Path to artifact
    name: build-output       # Required: Artifact name
    retention: 30            # Optional: Days to retain (default: 14)
    compression: gzip        # Optional: gzip, zip, none
    if_no_files: warn        # Optional: error, warn, ignore
    exclude:                 # Optional: Patterns to exclude
      - "*.map"
      - "*.log"
```

Retention Periods

Tier	Default Retention	Max Retention
Free	7 days	14 days
Team	30 days	90 days
Enterprise	90 days	365 days

Size Limits

Tier	Per-Artifact Limit	Total Storage
Free	100 MB	1 GB
Team	1 GB	50 GB
Enterprise	5 GB	Unlimited

Downloading Artifacts

From Same Pipeline

```yaml
stages:
  - name: build
    steps:
      - name: Build
        run: npm run build
      - artifacts:
          - path: dist/
            name: build-output

  - name: deploy
    steps:
      - name: Download artifact
        uses: artifacts/download
        with:
          name: build-output
          path: ./dist

      - name: Deploy
        run: ./deploy.sh ./dist
```

From Previous Pipeline

```yaml
- name: Download from previous build
  uses: artifacts/download
```

```yaml
  with:
    name: build-output
    pipeline: ${{ repo.name }}/main  # Repository and branch
    run: latest                       # or specific run number
    path: ./dist
```

Via CLI

```bash
# List artifacts
devpipeline artifacts list --pipeline 12345

# Download artifact
devpipeline artifacts download --pipeline 12345 --name build-output --path ./dist

# Download all artifacts
devpipeline artifacts download --pipeline 12345 --all --path ./artifacts
```

Via API

```bash
# List artifacts
curl https://api.devpipeline.novatech.com/v1/pipelines/12345/artifacts \
  -H "Authorization: Bearer $TOKEN"

# Get download URL
curl https://api.devpipeline.novatech.com/v1/artifacts/art_abc123/download \
  -H "Authorization: Bearer $TOKEN"
```

Test Results

JUnit Format

```yaml
- name: Run tests
  run: npm test -- --coverage --reporters=default --reporters=jest-junit

- name: Upload test results
  uses: test-results
  with:
    format: junit
    path: junit.xml
    name: test-results
```

Supported Formats

Format	Extensions
JUnit	.xml
TAP	.tap
xUnit	.xml
TestNG	.xml
Mocha JSON	.json
Jest	.json

Test Result Features

- Test summary in pipeline UI
- Failure details and stack traces
- Test duration tracking
- Flaky test detection
- Historical comparison

Code Coverage

Coverage Upload

```yaml
- name: Run tests with coverage
  run: npm test -- --coverage

- name: Upload coverage
  uses: coverage
  with:
    format: lcov
    path: coverage/lcov.info
```

Supported Formats

Format	Extensions
LCOV	.info
Cobertura	.xml
JaCoCo	.xml
Clover	.xml

Format	Extensions
Go Coverage	.out

Coverage Features

- Coverage percentage tracking
- Diff coverage on PRs
- Coverage trends
- File-level breakdown
- Branch coverage

Coverage Requirements

```yaml
coverage:
  thresholds:
    lines: 80
    branches: 70
    functions: 75
  on_failure: warn  # error or warn
```

Docker Images

Building and Pushing

```yaml
stages:
  - name: build-image
    steps:
      - name: Build Docker image
        run: |
          docker build -t myapp:${{ git.short_sha }} .

      - name: Push to registry
        run: |
          docker push myapp:${{ git.short_sha }}

      - name: Save image reference
        artifacts:
          - path: image-ref.txt
            name: docker-image
            content: "myapp:${{ git.short_sha }}"
```

Container Registry Integration

```yaml
stages:
  - name: build-image
    steps:
      - name: Login to registry
        run: |
          echo ${{ secrets.REGISTRY_PASSWORD }} | docker login \
            -u ${{ secrets.REGISTRY_USERNAME }} \
            --password-stdin registry.example.com

      - name: Build and push
        uses: docker/build-push
        with:
          context: .
          push: true
          tags: registry.example.com/myapp:${{ git.short_sha }}
```

Supported Registries

- Docker Hub
- Amazon ECR
- Google Container Registry
- Azure Container Registry
- GitHub Container Registry
- Self-hosted registries

Artifact Caching

Build Cache

```yaml
stages:
  - name: build
    cache:
      - key: npm-${{ hash('package-lock.json') }}
        paths:
          - node_modules/
      - key: build-${{ hash('src/**/*') }}
        paths:
          - .next/cache/

    steps:
```

```
      - run: npm ci
      - run: npm run build
```

Cache Best Practices

1. **Use content-based keys:**

```
key: npm-${{ hash('package-lock.json') }}
```

2. **Scope appropriately:**

```
cache:
  scope: branch  # or 'all' for shared cache
```

3. **Cache large dependencies:**

```
paths:
  - node_modules/
  - ~/.m2/repository/
  - ~/.cache/pip/
```

Artifact Permissions

Access Control

```
artifacts:
  - path: dist/
    name: build-output
    access:
      - team: engineering
        permission: download
      - user: deploy-bot
        permission: download
```

Permission Levels

Permission	Description
none	No access

Permission	Description
download	Can download artifact
delete	Can delete artifact
admin	Full control

Artifact Cleanup

Automatic Cleanup

- Artifacts deleted after retention period
- Storage reclaimed automatically
- No action required

Manual Cleanup

```
# Delete specific artifact
devpipeline artifacts delete --id art_abc123

# Delete all artifacts from pipeline
devpipeline artifacts delete --pipeline 12345 --all

# Delete old artifacts
devpipeline artifacts cleanup --older-than 30d
```

Common Patterns

Release Artifacts

```
stages:
  - name: build
    steps:
      - run: npm run build
      - artifacts:
          - path: dist/
            name: release-${{ git.tag }}
            retention: 365  # Keep releases longer

  - name: publish
```

```
    when:
      tag: v*
    steps:
      - uses: artifacts/download
        with:
          name: release-${{ git.tag }}
      - run: npm publish
```

Debug Artifacts

```
stages:
  - name: test
    steps:
      - run: npm test

    always:  # Run even on failure
      - name: Upload debug info
        if: ${{ pipeline.status == 'failure' }}
        artifacts:
          - path: logs/
            name: test-logs
          - path: screenshots/
            name: test-screenshots
```

Cross-Platform Build

```
stages:
  - name: build-linux
    runner: linux
    steps:
      - run: make build-linux
      - artifacts:
          - path: build/linux/
            name: linux-build

  - name: build-macos
    runner: macos
    steps:
      - run: make build-macos
      - artifacts:
          - path: build/macos/
            name: macos-build

  - name: release
```

```yaml
needs: [build-linux, build-macos]
steps:
  - uses: artifacts/download
    with:
      name: linux-build
  - uses: artifacts/download
    with:
      name: macos-build
  - run: ./create-release.sh
```

Troubleshooting

Artifact Upload Failed

- Check file path exists
- Verify size within limits
- Check permissions

Artifact Not Found

- Verify artifact name matches
- Check retention hasn't expired
- Confirm pipeline run succeeded

Download Slow

- Large artifacts take time
- Consider compression
- Use caching where appropriate

Related Documents: Pipeline YAML Reference (PRD-DP-020), Environment Variables (PRD-DP-025), CI/CD Best Practices (PRD-DP-010)