DevPipeline Parallel Builds Guide

Document ID: PRD-DP-030 **Last Updated:** 2024-02-20 **Owner:** DevPipeline Product Team **Classification:** Public

---

Overview

DevPipeline's parallel build feature allows you to run multiple jobs simultaneously, dramatically reducing pipeline execution time. This guide covers parallel execution strategies, configuration options, and best practices.

Parallel Execution Concepts

Job Parallelism

Run independent jobs concurrently:

```yaml
# pipeline.yaml
stages:
  - build
  - test
  - deploy

jobs:
  build-api:
    stage: build
    script:
      - npm run build:api

  build-web:
    stage: build
    script:
      - npm run build:web

  build-mobile:
    stage: build
    script:
      - npm run build:mobile
```

All three build jobs run simultaneously during the build stage.

Stage Parallelism

Multiple stages can run in parallel with dependencies:

```yaml
stages:
  - build
  - test
  - security
  - deploy

workflow:
  parallel_stages:
    - [test, security]  # Run test and security in parallel
```

Matrix Builds

Run the same job across multiple configurations:

```yaml
jobs:
  test:
    matrix:
      node_version: [16, 18, 20]
      os: [ubuntu-latest, macos-latest]
    script:
      - node --version
      - npm test
```

This creates 6 parallel jobs (3 versions × 2 OS).

Configuration

Basic Parallel Configuration

```yaml
# pipeline.yaml
settings:
  parallel:
    max_jobs: 10          # Maximum concurrent jobs
    fail_fast: true       # Stop all jobs on first failure
    retry_failed: false   # Don't auto-retry failed jobs
```

Per-Job Parallelism

```yaml
jobs:
  test:
    parallel: 4           # Run 4 instances of this job
    script:
      - npm run test:shard
    env:
      CI_NODE_INDEX: $DEVPIPELINE_PARALLEL_INDEX
      CI_NODE_TOTAL: $DEVPIPELINE_PARALLEL_TOTAL
```

Parallel with Dependencies

```yaml
jobs:
  build:
    script:
      - npm run build

  test-unit:
    needs: [build]
    script:
      - npm run test:unit

  test-integration:
    needs: [build]
    script:
      - npm run test:integration

  test-e2e:
    needs: [build]
    script:
      - npm run test:e2e

  deploy:
    needs: [test-unit, test-integration, test-e2e]
    script:
      - npm run deploy
```

Visual representation:

```
build → test-unit
      → test-integration  → deploy
      → test-e2e
```

Matrix Builds

Simple Matrix

```yaml
jobs:
  test:
    matrix:
      version: [3.8, 3.9, 3.10, 3.11]
    image: python:${{ matrix.version }}
    script:
      - python --version
      - pytest
```

Multi-Dimensional Matrix

```yaml
jobs:
  test:
    matrix:
      node: [16, 18, 20]
      database: [postgres, mysql, sqlite]
      include:
        - node: 20
          database: postgres
          coverage: true
      exclude:
        - node: 16
          database: mysql
    script:
      - npm test
    env:
      DB_TYPE: ${{ matrix.database }}
```

Matrix with Artifacts

```yaml
jobs:
  build:
    matrix:
      platform: [linux, darwin, windows]
      arch: [amd64, arm64]
    script:
      - go build -o app-${{ matrix.platform }}-${{ matrix.arch }}
    artifacts:
      paths:
        - app-*
```

```
  release:
    needs: [build]
    script:
      - upload-release app-*
```

Test Parallelization

Test Splitting

Automatically split tests across parallel runners:

```
jobs:
  test:
    parallel: 4
    script:
      - devpipeline test-split --total=$CI_NODE_TOTAL --index=$CI_NODE_INDEX
      - npm test -- $(cat test_files.txt)
```

Jest Parallel

```
jobs:
  test:
    parallel: 4
    script:
      - npx jest --shard=$CI_NODE_INDEX/$CI_NODE_TOTAL
```

Pytest Parallel

```
jobs:
  test:
    parallel: 4
    script:
      - pip install pytest-split
      - pytest --splits=$CI_NODE_TOTAL --group=$CI_NODE_INDEX
```

RSpec Parallel

```
jobs:
  test:
```

```
    parallel: 4
    script:
      - gem install knapsack
      - bundle exec rake knapsack:rspec
    env:
      CI_NODE_INDEX: $DEVPIPELINE_PARALLEL_INDEX
      CI_NODE_TOTAL: $DEVPIPELINE_PARALLEL_TOTAL
```

Intelligent Test Distribution

DevPipeline can distribute tests based on historical timing:

```
jobs:
  test:
    parallel: 4
    test_splitting:
      mode: timing           # Split by execution time
      timing_file: .test-times.json
    script:
      - npm test -- --files=$(cat split_tests.txt)
    artifacts:
      reports:
        timing: .test-times.json
```

Resource Management

Concurrency Limits

```
# Organization-level limits
settings:
  concurrency:
    max_parallel_jobs: 50      # Across all pipelines
    max_parallel_pipelines: 10 # Concurrent pipeline runs

# Pipeline-level limits
settings:
  concurrency:
    group: $DEVPIPELINE_REF
    cancel_in_progress: true
```

Resource Tags

Route jobs to specific runners:

```yaml
jobs:
  heavy-test:
    tags: [high-memory, gpu]
    parallel: 2
    script:
      - npm run test:heavy

  standard-test:
    tags: [standard]
    parallel: 4
    script:
      - npm run test:standard
```

Resource Allocation

```yaml
jobs:
  test:
    parallel: 4
    resources:
      cpu: 2
      memory: 4Gi
    script:
      - npm test
```

Error Handling

Fail Fast

Stop all parallel jobs when one fails:

```yaml
settings:
  parallel:
    fail_fast: true

jobs:
  test:
    parallel: 4
    script:
      - npm test
```

Continue on Failure

Allow other jobs to complete:

```yaml
settings:
  parallel:
    fail_fast: false

jobs:
  test:
    parallel: 4
    allow_failure: false
    script:
      - npm test
```

Retry Failed Jobs

```yaml
jobs:
  test:
    parallel: 4
    retry:
      max: 2
      when:
        - runner_error
        - stuck_or_timeout
    script:
      - npm test
```

Partial Failures

Handle matrix partial failures:

```yaml
jobs:
  test:
    matrix:
      browser: [chrome, firefox, safari, edge]
    allow_failure:
      matrix:
        browser: edge    # Edge failures don't fail pipeline
    script:
      - npm run test:e2e -- --browser=${{ matrix.browser }}
```

Artifact Handling

Collecting Artifacts from Parallel Jobs

```yaml
jobs:
  test:
    parallel: 4
    script:
      - npm test -- --coverage
    artifacts:
      paths:
        - coverage/
      merge_strategy: combine  # Combine artifacts from all parallel jobs

  coverage-report:
    needs: [test]
    script:
      - npx nyc merge coverage/ merged-coverage.json
      - npx nyc report --reporter=html
```

Matrix Artifacts

```yaml
jobs:
  build:
    matrix:
      os: [linux, macos, windows]
    script:
      - build --target=${{ matrix.os }}
    artifacts:
      name: build-${{ matrix.os }}
      paths:
        - dist/

  package:
    needs: [build]
    script:
      # All matrix artifacts available
      - ls dist/
      - create-release dist/*
```

Caching with Parallel Jobs

Shared Cache

```yaml
cache:
  key: npm-${{ hashFiles('package-lock.json') }}
  paths:
    - node_modules/

jobs:
  test:
    parallel: 4
    script:
      - npm test
```

Per-Job Cache

```yaml
jobs:
  test:
    parallel: 4
    cache:
      key: test-cache-${{ env.CI_NODE_INDEX }}
      paths:
        - .test-cache/
    script:
      - npm test
```

Monitoring Parallel Execution

Pipeline Visualization

DevPipeline provides real-time visualization: - Parallel job execution graph - Stage dependencies - Job status and timing

Metrics

```yaml
# Enable detailed metrics
settings:
  metrics:
    parallel_execution: true
    timing_breakdown: true
```

Access metrics via API:

```
devpipeline metrics --pipeline-id=12345 --format=json
```

Optimization Insights

```
# Analyze parallel efficiency
devpipeline analyze --pipeline-id=12345

# Output:
# Parallel Efficiency: 78%
# Longest Job: test-e2e (12m 34s)
# Idle Time: 2m 15s
# Recommendations:
#   - Split test-e2e into 2 parallel jobs
#   - Move lint to parallel with build
```

Best Practices

1. Identify Independent Jobs

```
# Good: Independent jobs run in parallel
jobs:
  lint:
    script: npm run lint
  test-unit:
    script: npm run test:unit
  test-integration:
    script: npm run test:integration
  security-scan:
    script: npm run security:scan
```

2. Balance Job Duration

Aim for similar job durations:

```
# Good: Balanced parallel jobs (~5 min each)
jobs:
  test-1:
    script: npm test -- --shard=1/4
  test-2:
```

```
    script: npm test -- --shard=2/4
  test-3:
    script: npm test -- --shard=3/4
  test-4:
    script: npm test -- --shard=4/4
```

3. Minimize Dependencies

```
# Bad: Serial execution
build → test → lint → security → deploy

# Good: Parallel where possible
build → test
      → lint    → deploy
      → security
```

4. Use Matrix for Multi-Environment Testing

```
jobs:
  test:
    matrix:
      node: [18, 20]
      os: [ubuntu, macos]
    script:
      - npm test
```

5. Set Appropriate Limits

```
settings:
  parallel:
    max_jobs: 10  # Prevent resource exhaustion
    fail_fast: true  # Don't waste resources on doomed pipelines
```

Troubleshooting

Jobs Not Running in Parallel

1. Check dependencies:

```yaml
jobs:
  job-b:
    needs: [job-a]  # This creates a dependency
```

2. Verify concurrency limits:

```
devpipeline quota
```

3. Check runner availability:

```
devpipeline runners list
```

Parallel Jobs Conflicting

1. Use unique workspaces:

```yaml
jobs:
  test:
    parallel: 4
    workspace:
      unique: true
```

2. Avoid shared state:

```yaml
jobs:
  test:
    parallel: 4
    env:
      TEST_DB: test_db_$CI_NODE_INDEX
```

Inconsistent Results

1. Ensure test isolation
2. Check for race conditions
3. Use deterministic test ordering

Examples

Full-Stack Application

```
stages:
  - build
  - test
  - deploy

jobs:
  build-api:
    stage: build
    script:
      - cd api && npm run build
    artifacts:
      paths:
        - api/dist/

  build-web:
    stage: build
    script:
      - cd web && npm run build
    artifacts:
      paths:
        - web/dist/

  test-api:
    stage: test
    needs: [build-api]
    parallel: 2
    script:
      - cd api && npm test -- --shard=$CI_NODE_INDEX/$CI_NODE_TOTAL

  test-web:
    stage: test
    needs: [build-web]
    parallel: 2
    script:
      - cd web && npm test -- --shard=$CI_NODE_INDEX/$CI_NODE_TOTAL

  test-e2e:
    stage: test
    needs: [build-api, build-web]
    parallel: 4
    matrix:
      browser: [chrome, firefox]
```

```
    script:
      - npm run e2e -- --browser=${{ matrix.browser }} --shard=$CI_NODE_INDEX/$CI_NODE_TOTAL

  deploy:
    stage: deploy
    needs: [test-api, test-web, test-e2e]
    script:
      - npm run deploy
```

Multi-Platform Build

```
jobs:
  build:
    matrix:
      include:
        - os: linux
          arch: amd64
          runner: ubuntu-latest
        - os: linux
          arch: arm64
          runner: ubuntu-arm64
        - os: darwin
          arch: amd64
          runner: macos-latest
        - os: darwin
          arch: arm64
          runner: macos-m1
        - os: windows
          arch: amd64
          runner: windows-latest
    runs-on: ${{ matrix.runner }}
    script:
      - go build -o myapp-${{ matrix.os }}-${{ matrix.arch }}
    artifacts:
      paths:
        - myapp-*
```

Related Documents: Getting Started (PRD-DP-001), Pipeline Configuration (PRD-DP-005), Caching Guide (PRD-DP-025)