DevPipeline Runner Configuration Guide

Document ID: PRD-DP-020 **Last Updated:** 2024-02-10 **Owner:** DevPipeline Engineering **Classification:** Public

---

Overview

DevPipeline runners execute your CI/CD jobs. This guide covers runner types, configuration, and best practices for optimal pipeline performance.

Runner Types

Cloud Runners (Managed)

DevPipeline provides fully managed cloud runners:

Size	vCPUs	Memory	Disk	Use Case
Small	2	4 GB	20 GB	Simple builds, tests
Medium	4	8 GB	50 GB	Standard workloads
Large	8	16 GB	100 GB	Complex builds
XLarge	16	32 GB	200 GB	Large monorepos

GPU Runners (Preview): | Type | GPU | Memory | Use Case | |——|——|———|———-| | GPU-T4 | NVIDIA T4 | 16 GB | ML inference | | GPU-A10 | NVIDIA A10 | 24 GB | ML training |

Self-Hosted Runners

Install runners on your own infrastructure for: - Access to internal networks - Specific hardware requirements - Compliance/data residency - Cost optimization at scale

---

Cloud Runner Configuration

Specifying Runner Size

```
jobs:
  build:
    runs-on: devpipeline-medium  # small, medium, large, xlarge
    steps:
      - run: npm build
```

Runner Images

Available images: | Image | OS | Preinstalled | |——-|—–|—————| | ubuntu-22.04 | Ubuntu 22.04 | Docker, Git, common tools | | ubuntu-20.04 | Ubuntu 20.04 | Docker, Git, common tools | | macos-13 | macOS Ventura | Xcode, Homebrew | | windows-2022 | Windows Server | VS Build Tools, Docker |

```
jobs:
  build:
    runs-on: devpipeline-medium
    image: ubuntu-22.04
```

Preinstalled Software

Ubuntu runners include: - Docker 24.x - Git 2.x - Node.js 18, 20 (via nvm) - Python 3.9, 3.10, 3.11 - Go 1.21 - Java 11, 17 - Ruby 3.x - Common build tools

Custom Container Images

Run jobs in custom containers:

```
jobs:
  build:
    runs-on: devpipeline-medium
    container:
      image: node:18-alpine
      credentials:
        username: ${{ secrets.DOCKER_USER }}
        password: ${{ secrets.DOCKER_TOKEN }}
```

Self-Hosted Runner Setup

Requirements

Minimum: - 2 vCPUs - 4 GB RAM - 20 GB disk - Docker (for container jobs) - Network access to DevPipeline

Recommended: - 4+ vCPUs - 8+ GB RAM - SSD storage - Dedicated machine/VM

Installation

Linux:

```
# Download runner
curl -fsSL https://runners.devpipeline.novatech.com/install.sh | bash

# Configure
devpipeline-runner configure \
  --url https://devpipeline.novatech.com \
  --token YOUR_REGISTRATION_TOKEN \
  --name my-runner \
  --tags linux,docker

# Start
devpipeline-runner start
```

Docker:

```
docker run -d \
  --name devpipeline-runner \
  -e RUNNER_TOKEN=YOUR_TOKEN \
  -e RUNNER_NAME=my-runner \
  -e RUNNER_TAGS=docker,linux \
  -v /var/run/docker.sock:/var/run/docker.sock \
  novatech/devpipeline-runner:latest
```

Kubernetes:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devpipeline-runner
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: runner
          image: novatech/devpipeline-runner:latest
          env:
            - name: RUNNER_TOKEN
              valueFrom:
                secretKeyRef:
                  name: runner-token
                  key: token
            - name: RUNNER_TAGS
              value: "kubernetes,docker"
```

Registration Token

Get your registration token: 1. Go to Settings > Runners 2. Click "New Runner" 3. Copy the registration token 4. Token expires after 24 hours

---

Runner Configuration

Configuration File

`/etc/devpipeline-runner/config.yaml`:

```
name: my-runner
url: https://devpipeline.novatech.com
token: ${RUNNER_TOKEN}

tags:
  - linux
  - docker
  - gpu

concurrent: 4  # Jobs running simultaneously
check_interval: 3  # Seconds between job checks

executor:
  type: docker
  docker:
    image: ubuntu:22.04
```

```yaml
    privileged: false
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    network: bridge

cache:
  type: local
  path: /var/cache/devpipeline
  max_size: 10GB

logging:
  level: info
  format: json
```

Runner Tags

Use tags to route jobs to specific runners:

```yaml
# In pipeline
jobs:
  build-ios:
    runs-on: [self-hosted, macos, xcode]

  build-android:
    runs-on: [self-hosted, linux, docker]

  deploy-internal:
    runs-on: [self-hosted, internal-network]
```

Executor Types

Executor	Use Case	Isolation
docker	Container-based jobs	High
shell	Direct execution	Low
kubernetes	K8s pod per job	High
custom	Custom executor	Varies

Runner Groups

Creating Groups

Organize runners into groups for access control:

```
# Create group
devpipeline runner-group create \
  --name production-runners \
  --projects project-1,project-2

# Add runner to group
devpipeline runner update my-runner \
  --group production-runners
```

Access Control

Group Setting	Description
All projects	Any project can use runners
Selected projects	Only specified projects
Selected branches	Only specific branches

Autoscaling

Cloud Runner Autoscaling

Managed runners autoscale automatically based on queue depth.

Self-Hosted Autoscaling

AWS Auto Scaling:

```
# CloudFormation snippet
AutoScalingGroup:
  MinSize: 1
  MaxSize: 10
  TargetTrackingConfiguration:
    CustomizedMetricSpecification:
      MetricName: JobQueueDepth
      Namespace: DevPipeline
    TargetValue: 5
```

Kubernetes HPA:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: runner-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: devpipeline-runner
  minReplicas: 2
  maxReplicas: 20
  metrics:
    - type: External
      external:
        metric:
          name: devpipeline_job_queue_depth
        target:
          type: AverageValue
          averageValue: 3
```

Caching

Cache Configuration

```
cache:
  type: s3  # local, s3, gcs
  s3:
    bucket: my-runner-cache
    region: us-west-2
    prefix: cache/
  max_size: 50GB
  cleanup_interval: 24h
```

Cache in Pipelines

```
jobs:
  build:
    cache:
      key: ${{ runner.os }}-npm-${{ hashFiles('package-lock.json') }}
      paths:
        - node_modules/
        - ~/.npm/
      restore_keys:
        - ${{ runner.os }}-npm-
```

Security

Runner Security Best Practices

1. **Isolate runners:** Use dedicated VMs/containers
2. **Limit permissions:** Minimal cloud/network access
3. **Clean workspace:** Clear between jobs
4. **Update regularly:** Keep runner software current
5. **Monitor activity:** Log and audit job execution

Secure Configuration

```
executor:
  docker:
    privileged: false  # Never use privileged mode
    cap_drop:
      - ALL
    read_only: true
    security_opt:
      - no-new-privileges:true
```

Secret Handling

Runners receive secrets as environment variables: - Secrets are masked in logs - Secrets are cleared after job completion - Use secret masking for custom secrets

---

Monitoring

Runner Metrics

Metric	Description
job_duration	Time to complete jobs
job_queue_time	Wait time in queue
runner_busy	Percentage of time busy
job_success_rate	Successful job percentage

Health Checks

```
# Check runner status
devpipeline-runner status

# View recent jobs
devpipeline-runner jobs --limit 10

# Check connectivity
devpipeline-runner verify
```

Logs

```
# View runner logs
journalctl -u devpipeline-runner -f

# Docker logs
docker logs devpipeline-runner --tail 100
```

Troubleshooting

Runner Offline

1. Check network connectivity
2. Verify registration token
3. Check runner service status
4. Review runner logs

Jobs Stuck in Queue

1. Check runner availability
2. Verify tags match
3. Check runner group permissions
4. Review concurrent job limits

Job Failures

1. Check job logs in DevPipeline UI
2. SSH to runner for investigation
3. Check resource limits (memory, disk)
4. Verify Docker/executor health

Best Practices

Performance

1. **Right-size runners:** Match runner size to workload
2. **Use caching:** Cache dependencies aggressively
3. **Parallelize:** Run independent jobs concurrently
4. **Optimize images:** Use slim container images

Cost

1. **Autoscale:** Scale down during low activity
2. **Spot instances:** Use spot/preemptible for non-critical
3. **Shared runners:** Share runners across projects
4. **Cache effectively:** Reduce redundant downloads

Reliability

1. **Redundancy:** Multiple runners per workload
2. **Health monitoring:** Alert on runner issues
3. **Regular updates:** Keep runners current
4. **Test changes:** Test runner config changes

Related Documents: Quick Start Guide (PRD-DP-001), Pipeline YAML Reference (PRD-DP-010), CI/CD Best Practices (PRD-DP-005)