DevPipeline Environment Variables Guide

Document ID: PRD-DP-025 **Last Updated:** 2024-02-20 **Owner:** DevPipeline Product Team **Classification:** Public

---

Overview

Environment variables in DevPipeline allow you to configure pipelines dynamically, manage secrets securely, and maintain different configurations for various environments.

Variable Types

Pipeline Variables

Variables defined in your pipeline configuration file.

```yaml
# .devpipeline.yaml
env:
  NODE_ENV: production
  API_URL: https://api.example.com
  LOG_LEVEL: info
```

Repository Variables

Variables set at the repository level, available to all pipelines.

Setting via UI: 1. Go to **Repository** → **Settings** → **Variables** 2. Click **Add Variable** 3. Enter name and value 4. Click **Save**

Setting via CLI:

```
devpipeline var set MY_VAR "my value" --repo my-org/my-repo
```

Organization Variables

Variables available to all repositories in an organization.

Setting via UI: 1. Go to **Organization** → **Settings** → **Variables** 2. Click **Add Variable** 3. Enter name and value 4. Select visibility (all repos or specific) 5. Click **Save**

Environment Variables

Variables scoped to specific deployment environments.

```yaml
# .devpipeline.yaml
environments:
  staging:
    env:
      API_URL: https://staging-api.example.com
      DEBUG: "true"

  production:
    env:
      API_URL: https://api.example.com
      DEBUG: "false"
```

Secrets

What Are Secrets?

Secrets are encrypted environment variables for sensitive data like: - API keys - Passwords - Tokens - Private keys

Creating Secrets

Via UI: 1. Go to **Settings** → **Secrets** 2. Click **Add Secret** 3. Enter name and value 4. Select scope (repository, environment, organization) 5. Click **Save**

Via CLI:

```
devpipeline secret set DATABASE_PASSWORD "mysecretpass" --repo my-org/my-repo
```

Using Secrets

```yaml
# .devpipeline.yaml
stages:
  - name: deploy
    steps:
      - name: Deploy
        env:
          DB_PASS: ${{ secrets.DATABASE_PASSWORD }}
          API_KEY: ${{ secrets.API_KEY }}
        run: ./deploy.sh
```

Secret Security

- Secrets are encrypted at rest (AES-256)
- Never printed in logs (masked)
- Not exposed in forked repositories
- Access controlled by permissions

Built-in Variables

Pipeline Context

Variable	Description	Example
`${{ pipeline.id }}`	Pipeline run ID	`12345`
`${{ pipeline.name }}`	Pipeline name	`main-pipeline`
`${{ pipeline.number }}`	Build number	`147`
`${{ pipeline.status }}`	Current status	`running`

Repository Context

Variable	Description	Example
`${{ repo.name }}`	Repository name	`my-app`
`${{ repo.full_name }}`	Full repo name	`my-org/my-app`
`${{ repo.owner }}`	Organization/owner	`my-org`
`${{ repo.default_branch }}`	Default branch	`main`
`${{ repo.url }}`	Repository URL	`https://github.com/...`

Git Context

Variable	Description	Example
`${{ git.branch }}`	Branch name	`feature/auth`
`${{ git.tag }}`	Tag name (if triggered by tag)	`v1.0.0`
`${{ git.commit_sha }}`	Full commit SHA	`abc123def...`
`${{ git.short_sha }}`	Short commit SHA	`abc123d`
`${{ git.commit_message }}`	Commit message	`Fix login bug`
`${{ git.author_name }}`	Author name	`John Doe`
`${{ git.author_email }}`	Author email	`john@example.com`

Pull Request Context

Variable	Description	Example
`${{ pr.number }}`	PR number	`42`
`${{ pr.title }}`	PR title	`Add new feature`
`${{ pr.source_branch }}`	Source branch	`feature/new`
`${{ pr.target_branch }}`	Target branch	`main`
`${{ pr.author }}`	PR author	`johndoe`

Environment Context

Variable	Description	Example
`${{ env.name }}`	Environment name	`production`
`${{ env.url }}`	Environment URL	`https://app.example.com`

Variable Syntax

Basic Usage

```yaml
steps:
  - run: echo "Branch is ${{ git.branch }}"
  - run: echo "Using API at $API_URL"
```

Default Values

```yaml
env:
  LOG_LEVEL: ${{ vars.LOG_LEVEL || 'info' }}
  TIMEOUT: ${{ vars.TIMEOUT || '30' }}
```

Conditional Values

```yaml
env:
  DEBUG: ${{ git.branch == 'main' && 'false' || 'true' }}
```

String Interpolation

```
steps:
  - run: |
      echo "Deploying ${{ repo.name }}:${{ git.short_sha }}"
      echo "Environment: ${{ env.name }}"
```

Variable Precedence

Variables are resolved in this order (later overrides earlier):

1. Built-in variables (lowest)
2. Organization variables
3. Repository variables
4. Pipeline file variables
5. Environment-specific variables
6. Step-level variables (highest)

Example

```
# Organization variable: API_URL=https://org-api.com

# Repository variable: API_URL=https://repo-api.com

# Pipeline file:
env:
  API_URL: https://pipeline-api.com  # This wins

stages:
  - name: test
    env:
      API_URL: https://stage-api.com  # This wins for this stage
```

Dynamic Variables

From Previous Steps

```
stages:
  - name: build
```

```
steps:
  - name: Get version
    id: version
    run: |
      VERSION=$(cat package.json | jq -r .version)
      echo "version=$VERSION" >> $DEVPIPELINE_OUTPUT

  - name: Build image
    run: docker build -t myapp:${{ steps.version.outputs.version }} .
```

From Files

```
steps:
  - name: Load config
    run: |
      while IFS='=' read -r key value; do
        echo "$key=$value" >> $DEVPIPELINE_ENV
      done < config.env
```

Setting for Subsequent Steps

```
# Set variable for subsequent steps
echo "MY_VAR=my_value" >> $DEVPIPELINE_ENV

# Set output for reference by other steps
echo "result=success" >> $DEVPIPELINE_OUTPUT
```

Masking Sensitive Data

Automatic Masking

Secrets are automatically masked in logs:

```
Deploying with key: ***
```

Manual Masking

Mask dynamic sensitive values:

```
steps:
  - name: Generate token
    run: |
      TOKEN=$(./generate-token.sh)
      echo "::mask::$TOKEN"
      echo "token=$TOKEN" >> $DEVPIPELINE_OUTPUT
```

Best Practices

Naming Conventions

```
# Good - descriptive, uppercase with underscores
DATABASE_HOST: db.example.com
API_SECRET_KEY: ${{ secrets.API_KEY }}
FEATURE_FLAG_NEW_UI: "true"

# Avoid - inconsistent, unclear
dbhost: db.example.com
apiKey: secret
flag1: "true"
```

Security

1. **Use secrets** for sensitive data
2. **Don't hardcode** credentials in pipeline files
3. **Limit scope** - use environment-specific where possible
4. **Rotate secrets** regularly
5. **Audit access** to secrets

Organization

1. **Group related variables** together
2. **Document** non-obvious variables
3. **Use defaults** where appropriate
4. **Consistent naming** across repositories

Debugging

```
steps:
  - name: Debug variables
```

```
    run: |
      echo "Branch: ${{ git.branch }}"
      echo "Commit: ${{ git.short_sha }}"
      # Don't echo secrets!
      # echo "API Key: $API_KEY"  # BAD
```

Common Patterns

Feature Flags

```
env:
  ENABLE_NEW_FEATURE: ${{ git.branch == 'main' && 'true' || 'false' }}

steps:
  - name: Conditional step
    if: ${{ env.ENABLE_NEW_FEATURE == 'true' }}
    run: ./enable-feature.sh
```

Environment-Specific Config

```
environments:
  development:
    env:
      DATABASE_URL: postgres://dev-db:5432/app
      REDIS_URL: redis://dev-redis:6379

  production:
    env:
      DATABASE_URL: ${{ secrets.PROD_DATABASE_URL }}
      REDIS_URL: ${{ secrets.PROD_REDIS_URL }}
```

Version Tagging

```
env:
  VERSION: ${{ git.tag || git.short_sha }}
  IMAGE_TAG: ${{ repo.name }}:${{ env.VERSION }}

steps:
  - run: docker build -t $IMAGE_TAG .
  - run: docker push $IMAGE_TAG
```

Troubleshooting

Variable Not Resolving

- Check spelling and case
- Verify scope (org vs repo vs environment)
- Check precedence order
- Ensure variable is set before use

Secret Not Available

- Verify secret exists in correct scope
- Check permissions
- Forked repos don't have access to secrets
- Secrets not available in PRs from forks

Masking Not Working

- Ensure using `secrets.` context
- Use `::mask::` for dynamic values
- Multiline secrets may not mask properly

Related Documents: Pipeline YAML Reference (PRD-DP-020), Security Scanning (PRD-DP-040), Secrets Management with SecureVault (PRD-SV-001)