# SMUS CI/CD Pipeline CLI Documentation

Complete Documentation with Diagrams

August 25, 2025

## Contents

*Source: README.md*

# 1 SMUS CI/CD CLI

A CLI tool for managing CI/CD pipelines in SageMaker Unified Studio (SMUS), enabling automated deployment of data science workflows and assets across multiple environments.

## 1.1 Pipeline Manifest Reference

See **Pipeline Manifest Reference** for complete guide to pipeline configuration.

## 1.2 CLI Commands Reference

See **CLI Commands Reference** for detailed command documentation and examples.

## 1.3 GitHub Actions Integration

See **GitHub Actions Integration** for automated CI/CD pipeline setup.

## 1.4 What is a CI/CD Pipeline?

**Continuous Integration/Continuous Deployment (CI/CD)** is a software development practice that automates the process of integrating code changes, testing them, and deploying them to different environments. A CI/CD pipeline consists of:

- **Source Control**: Code and configuration stored in version control
- **Build/Package**: Creating deployable artifacts from source code
- **Test Environments**: Staging areas for validation and testing
- **Production Deployment**: Automated deployment to live environments
- **Monitoring**: Tracking deployment success and application health

In the context of **SageMaker Unified Studio**, a CI/CD pipeline manages: - **Data Science Workflows**: Airflow DAGs, Jupyter notebooks, and ML pipelines - **Data Assets**: Datasets, models, and analytical outputs - **Environment Configuration**: Project settings, user permissions, and resource allocation - **Cross-Environment Promotion**: Moving validated work from dev -> test -> production

## 1.5 SMUS Pipeline Architecture

The SMUS CI/CD system consists of CLI operations that manage target environments. Each target represents a complete deployment environment with its own resources.

### 1.5.1 CLI Operations Flow

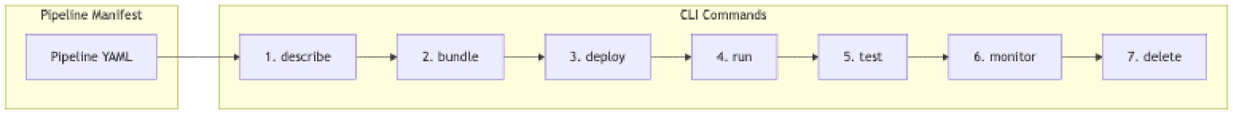
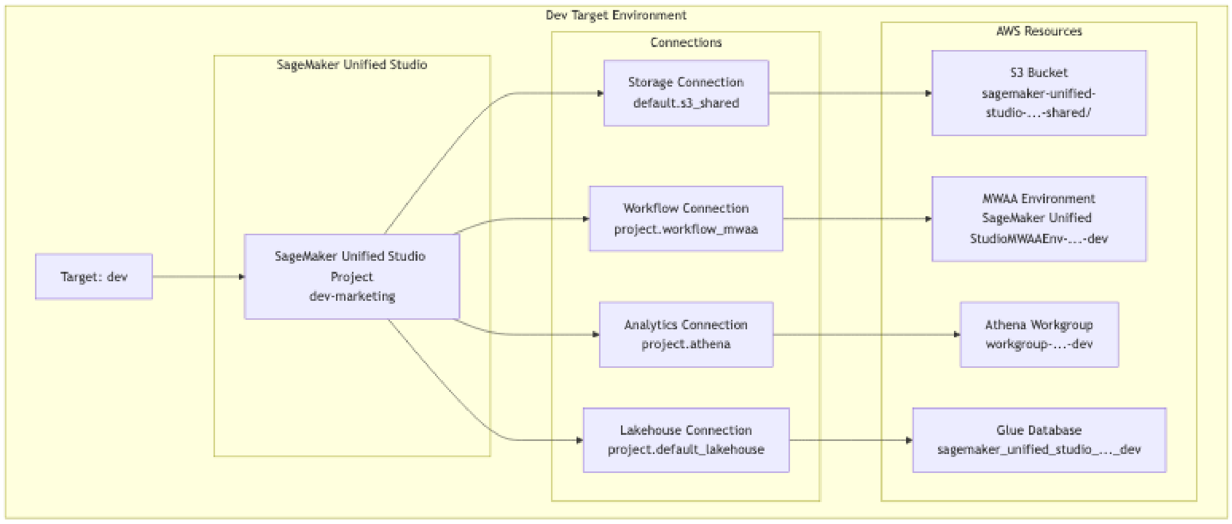


Figure 1: Diagram 1

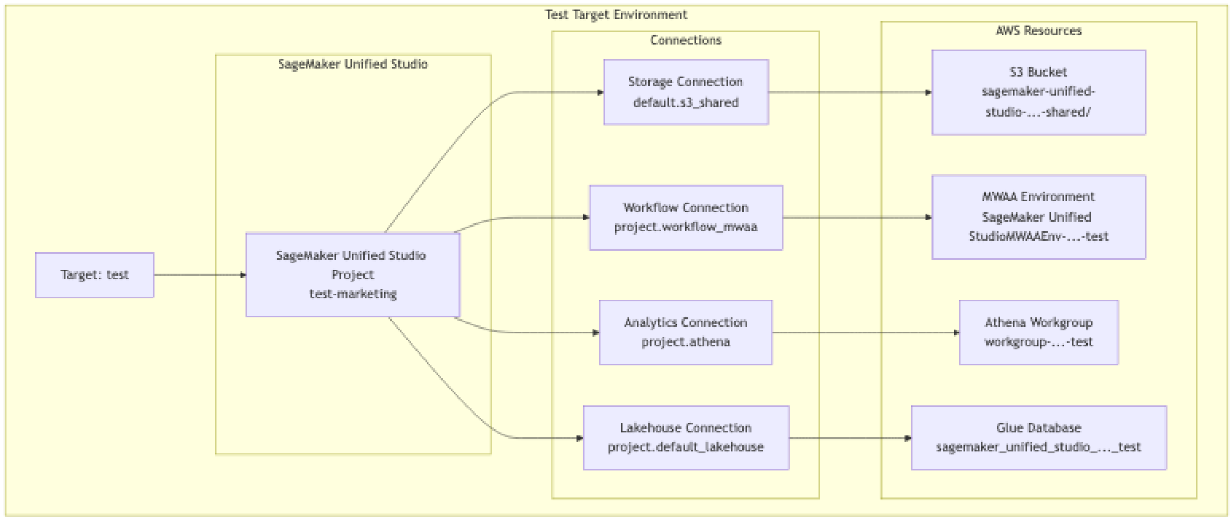
#### 1.5.2 Target Environment Composition



Dev Target Environment
SageMaker Unified Studio
Connections
AWS Resources
Target: dev
SageMaker Unified Studio Project dev-marketing
Storage Connection default.s3_shared
Workflow Connection project.workflow_mwaa
Analytics Connection project.athena
Lakehouse Connection project.default_lakehouse
S3 Bucket sagemaker-unified-studio-...-shared/
MWAA Environment SageMaker Unified StudioMWAAEnv-...-dev
Athena Workgroup workgroup-...-dev
Glue Database sagemaker_unified_studio_..._dev

Figure 2: Diagram 2

##### 1.5.2.1 Development Environment



Test Target Environment
SageMaker Unified Studio
Connections
AWS Resources
Target: test
SageMaker Unified Studio Project test-marketing
Storage Connection default.s3_shared
Workflow Connection project.workflow_mwaa
Analytics Connection project.athena
Lakehouse Connection project.default_lakehouse
S3 Bucket sagemaker-unified-studio-...-shared/
MWAA Environment SageMaker Unified StudioMWAAEnv-...-test
Athena Workgroup workgroup-...-test
Glue Database sagemaker_unified_studio_..._test

Figure 3: Diagram 3

##### 1.5.2.2 Test Environment

##### 1.5.2.3 Production Environment

### 1.6 Key Concepts

#### 1.6.1 Pipeline Stages -> SMUS Projects

Each **pipeline stage** (dev, test, prod) maps to a **SageMaker Unified Studio Project**:

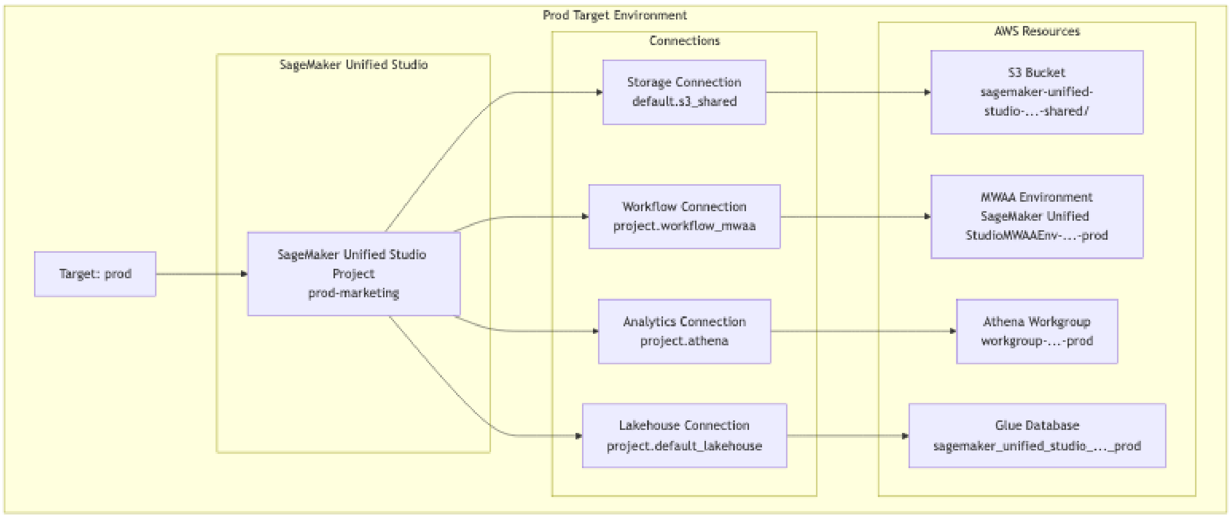

Prod Target Environment
SageMaker Unified Studio
Connections
AWS Resources
Target: prod
SageMaker Unified Studio Project prod-marketing
Storage Connection default.s3_shared
Workflow Connection project.workflow_mwaa
Analytics Connection project.athena
Lakehouse Connection project.default_lakehouse
S3 Bucket sagemaker-unified-studio-...-shared/
MWAA Environment SageMaker Unified StudioMWAAEnv-...-prod
Athena Workgroup workgroup-...-prod
Glue Database sagemaker_unified_studio_..._prod

Figure 4: Diagram 4

- **Dev Stage -> Dev Project** (`dev-marketing`)
  - Development and experimentation
  - Rapid iteration and testing
  - Individual developer workspaces
- **Test Stage -> Test Project** (`test-marketing`)
  - Integration testing and validation
  - Staging environment for QA
  - Pre-production verification
- **Prod Stage -> Prod Project** (`prod-marketing`)
  - Production deployment
  - Live data processing
  - Business-critical workflows

#### 1.6.2 Resource Mapping

Each project contains: - **S3 Storage Connections** - For data assets and notebooks - **Workflow Connections** - For Airflow DAGs and ML pipelines - **Environment Configurations** - Compute and runtime settings - **User Permissions** - Access control and collaboration

### 1.7 Installation

```
pip install -e .
```

### 1.8 Quick Start

For detailed command examples and outputs, see **CLI Commands Reference**.

#### 1.8.1 Basic Workflow

```
# 1. Validate pipeline configuration
smus-cli describe --pipeline pipelines/pipeline1.yaml --connect
```

```
# 2. Create deployment bundle from dev environment
smus-cli bundle --pipeline pipelines/pipeline1.yaml --targets dev

# 3. Deploy to marketing test stage
smus-cli deploy --targets marketing-test-stage --pipeline pipelines/pipeline1.yaml

# 4. Monitor workflow status
smus-cli monitor --pipeline pipelines/pipeline1.yaml

# 5. Trigger workflow execution
smus-cli run --pipeline pipelines/pipeline1.yaml --targets marketing-test-stage --workflow tes

# 6. Run tests to validate deployment
smus-cli test --pipeline pipelines/pipeline1.yaml --targets marketing-test-stage

# 7. Clean up resources (when needed)
smus-cli delete --targets marketing-test-stage --pipeline pipelines/pipeline1.yaml --force
```

## 1.9 Common Workflows

### 1.9.1 Development Workflow

1. **Update code** in dev environment S3 location
2. **Create bundle**: `smus-cli bundle` (downloads latest from dev)
3. **Deploy to test**: `smus-cli deploy --targets test` (deploys and triggers workflows)
4. **Verify execution**: Check workflow runs in SageMaker Unified Studio console
5. **Deploy to prod**: `smus-cli deploy --targets prod` (when ready)

### 1.9.2 Complete CI/CD Flow

```
# 1. Analyze pipeline configuration
smus-cli describe --pipeline pipelines/pipeline1.yaml --workflows --targets --connect

# 2. Create deployment bundle from current dev state
smus-cli bundle dev

# 3. Deploy to staging (auto-initializes if needed)
smus-cli deploy --targets staging

# 4. After validation, deploy to production (auto-initializes if needed)
smus-cli deploy --targets prod
```

## 1.10 Testing

- **Unit Tests**: `python run_tests.py --type unit`
- **Integration Tests**: `python run_integration_tests.py --type all`

See tests/README.md for detailed testing documentation.

## 2 Pipeline Manifest

*Source: docs/pipeline-manifest.md*

## 3 Pipeline Manifest Reference

<- Back to Main README

The pipeline manifest is a YAML file that defines your CI/CD pipeline configuration, including targets, workflows, and deployment settings.

### 3.1 Quick Links

- **Pipeline Manifest Schema Documentation** - Complete schema reference with validation rules and examples
- **CLI Commands Reference** - Detailed command documentation

### 3.2 Complete Example

```yaml
pipelineName: MarketingDataPipeline
bundlesDirectory: ./bundles

# Domain configuration
domain:
  name: my-studio-domain
  region: us-east-1

# What to include in deployment bundles
bundle:
  workflow:
    - connectionName: default.s3_shared
      append: true
      include: ['workflows/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc']
  storage:
    - connectionName: default.s3_shared
      append: false
      include: ['src/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc']
  git:
    repository: MyDataPipeline
    url: https://github.com/myorg/data-pipeline.git
    targetDir: git

# Target environments
targets:
  dev:
    default: true
    project:
```

```
    name: dev-marketing

test:
  project:
    name: test-marketing
  initialization:
    project:
      create: true
      profileName: 'All capabilities'
      owners: ['alice@company.com']
      contributors: ['bob@company.com', 'charlie@company.com']
    environments:
      - EnvironmentConfigurationName: 'OnDemand Workflows'
  tests:
    folder: tests/integration/
  bundle_target_configuration:
    storage:
      connectionName: default.s3_shared
      directory: 'src'
    workflows:
      connectionName: default.s3_shared
      directory: 'workflows'
  workflows:
    - workflowName: marketing_etl_dag
      parameters:
        environment: test
        debug_mode: true

prod:
  project:
    name: prod-marketing
  initialization:
    project:
      create: true
      profileName: 'All capabilities'
      owners: ['alice@company.com']
      contributors: []
    environments:
      - EnvironmentConfigurationName: 'OnDemand Workflows'
  bundle_target_configuration:
    storage:
      connectionName: default.s3_shared
      directory: 'src'
    workflows:
      connectionName: default.s3_shared
      directory: 'workflows'
  workflows:
    - workflowName: marketing_etl_dag
```

```
        parameters:
          environment: production
          debug_mode: false

# Global workflows (apply to all targets)
workflows:
  - workflowName: marketing_etl_dag
    connectionName: project.workflow_connection
    triggerPostDeployment: true
    logging: console
    engine: Workflows
    parameters:
      data_source: s3://marketing-data/
      output_bucket: s3://marketing-results/
```

## 3.3 Section Reference

### 3.3.1 Pipeline Metadata

```
pipelineName: MarketingDataPipeline
bundlesDirectory: ./bundles
```

- **`pipelineName`** (required): Name of your pipeline, used for bundle filenames and identification
- **`bundlesDirectory`** (optional): Directory where bundle zip files are created (default: `./bundles`)

### 3.3.2 Domain Configuration

```
domain:
  name: my-studio-domain
  region: us-east-1
```

- **`domain.name`** (required): SageMaker Unified Studio domain name
- **`domain.region`** (required): AWS region where the domain exists

### 3.3.3 Bundle Configuration

Defines what content to include in deployment packages:

#### 3.3.3.1 Workflow Bundles

```
bundle:
  workflow:
    - connectionName: default.s3_shared
      append: true
      include: ['workflows/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/']
```

- **`connectionName`** (required): S3 connection name in the source project
- **`append`** (optional): Whether to append to existing files (default: `true`)
- **`include`** (optional): List of paths/patterns to include

- **exclude** (optional): List of paths/patterns to exclude

#### 3.3.3.2 Storage Bundles

```yaml
bundle:
  storage:
    - connectionName: default.s3_shared
      append: false
      include: ['src/', 'data/']
      exclude: ['*.tmp', '.DS_Store']
```

- **append** (optional): Whether to append to existing files (default: `false` for storage)

#### 3.3.3.3 Git Repositories

```yaml
bundle:
  git:
    repository: MyDataPipeline
    url: https://github.com/myorg/data-pipeline.git
    targetDir: git
```

- **repository** (optional): Repository name for identification
- **url** (required): Git repository URL
- **targetDir** (optional): Directory name in bundle (default: `git`)

### 3.3.4 Target Configuration

Each target represents a deployment environment:

```yaml
targets:
  dev:
    default: true
    project:
      name: dev-marketing
```

- **default** (optional): Mark as default target for commands
- **project.name** (required): SageMaker Unified Studio project name

#### 3.3.4.1 Target Initialization

```yaml
targets:
  test:
    initialization:
      project:
        create: true
        profileName: 'All capabilities'
        owners: ['alice@company.com']
        contributors: ['bob@company.com']
        userParameters:
          - EnvironmentConfigurationName: 'Lakehouse Database'
            parameters:
```

```
          - name: glueDbName
            value: my_unique_db_name
      environments:
        - EnvironmentConfigurationName: 'OnDemand Workflows'
```

- **project.create** (optional): Whether to auto-create project (default: `false`)
- **project.profileName** (required if create=true): Project profile name
- **project.owners** (optional): List of project owner email addresses
- **project.contributors** (optional): List of project contributor email addresses
- **project.userParameters** (optional): Override project profile parameters during creation
  - **EnvironmentConfigurationName**: Name of environment configuration to override
  - **parameters**: Array of parameter name/value pairs to override
- **environments** (optional): List of environments to create post-project creation

#### 3.3.4.2 Bundle Target Configuration

```
targets:
  test:
    bundle_target_configuration:
      storage:
        connectionName: default.s3_shared
        directory: 'src'
      workflows:
        connectionName: default.s3_shared
        directory: 'workflows'
```

- **storage.connectionName** (required): Target S3 connection for storage files
- **storage.directory** (optional): Target directory path
- **workflows.connectionName** (required): Target S3 connection for workflow files
- **workflows.directory** (optional): Target directory path

#### 3.3.4.3 Target-Specific Workflows

```
targets:
  test:
    workflows:
      - workflowName: marketing_etl_dag
        parameters:
          environment: test
          debug_mode: true
```

- **workflowName** (required): Name of workflow to configure
- **parameters** (optional): Target-specific workflow parameters

#### 3.3.4.4 Target Tests

```
targets:
  test:
    tests:
      folder: tests/integration/
```

- **folder** (required): Relative path to folder containing Python test files

Test files receive environment variables: - SMUS_DOMAIN_ID: SageMaker Unified Studio domain ID - SMUS_PROJECT_ID: Project ID - SMUS_PROJECT_NAME: Project name - SMUS_TARGET_NAME: Target name - SMUS_REGION: AWS region - SMUS_DOMAIN_NAME: Domain name

#### 3.3.5 Global Workflows

```yaml
workflows:
  - workflowName: marketing_etl_dag
    connectionName: project.workflow_connection
    triggerPostDeployment: true
    logging: console
    engine: Workflows
    parameters:
      data_source: s3://marketing-data/
```

- **workflowName** (required): Workflow/DAG name in the workflow engine
- **connectionName** (required): Workflow connection name in projects
- **triggerPostDeployment** (optional): Whether to trigger after deployment (default: `false`)
- **logging** (optional): Logging level (`console`, `none`) (default: `none`)
- **engine** (optional): Workflow engine type (default: `Workflows`)
- **parameters** (optional): Global workflow parameters (merged with target-specific)

## 3.4 Validation Rules

### 3.4.1 Required Fields

- `pipelineName`
- `domain.name` and `domain.region`
- At least one target with `project.name`

### 3.4.2 Optional Sections

- `bundle` - If omitted, no bundling operations
- `workflows` - If omitted, no workflow operations
- `initialization` - If omitted, assumes projects exist

### 3.4.3 Connection Names

- Must match actual connection names in SageMaker Unified Studio projects
- Format: `{connection_name}` (e.g., `default.s3_shared`, `project.workflow_connection`)

### 3.4.4 File Patterns

- Support glob patterns: `*.py`, `**/*.yaml`
- Exclude patterns take precedence over include patterns
- Paths are relative to bundle source directories

## 3.5 Best Practices

### 3.5.1 Naming Conventions

- Use descriptive pipeline names: `MarketingDataPipeline`, `CustomerAnalytics`
- Use consistent target names: `dev`, `test`, `prod`
- Use clear project names: `{team}-{environment}` (e.g., `marketing-dev`)

### 3.5.2 Bundle Configuration

- Always exclude temporary files: `.ipynb_checkpoints/`, `__pycache__/`, `*.pyc`
- Use `append: true` for workflows (allows incremental updates)
- Use `append: false` for storage (ensures clean deployments)

### 3.5.3 Target Organization

- Mark one target as `default: true` for convenience
- Use initialization only for non-production environments
- Keep production targets minimal and explicit

### 3.5.4 Workflow Parameters

- Use global parameters for common settings
- Use target-specific parameters for environment differences
- Avoid hardcoded values - use parameters instead

# 4 Pipeline Manifest Schema

*Source: docs/pipeline-manifest-schema.md*

# 5 Pipeline Manifest Schema

<- Back to Main README

This directory contains the JSON schema and validation tools for SMUS CI/CD pipeline manifests.

## 5.1 Files

- **`pipeline-manifest-schema.yaml`** - YAML Schema definition for pipeline manifests
- **`pipeline-manifest-schema.json`** - JSON Schema definition (legacy)
- **`validate_manifests.py`** - Python script to validate manifests against the schema
- **`INCONSISTENCIES.md`** - Documentation of inconsistencies found during schema creation
- **`README.md`** - This documentation file

## 5.2 Schema Overview

The schema defines the structure for SMUS CI/CD pipeline manifests with the following main sections:

### 5.2.1 Required Fields

- **`pipelineName`** - Unique pipeline identifier
- **`domain`** - DataZone domain configuration (name, region)
- **`targets`** - Target environments (dev, test, prod, etc.)

### 5.2.2 Optional Fields

- **`bundle`** - Bundle creation configuration
- **`workflows`** - Global workflow definitions

### 5.2.3 Recent Schema Updates

**5.2.3.1 User Parameters Support** The schema now supports `userParameters` for overriding DataZone project profile parameters during creation:

```yaml
targets:
  test:
    initialization:
      project:
        create: true
        profileName: 'All capabilities'
        userParameters:
          - EnvironmentConfigurationName: 'Lakehouse Database'
            parameters:
              - name: glueDbName
                value: my_unique_db_name
```

This allows customization of project profile settings like database names, preventing conflicts during project creation.

## 5.3 Usage

### 5.3.1 Validate All Manifests

```
cd /path/to/smus_cicd
python schema/validate_manifests.py
```

### 5.3.2 Validate Single Manifest (Python)

```
import yaml
from jsonschema import validate

# Load schema
with open('schema/pipeline-manifest-schema.yaml', 'r') as f:
    schema = yaml.safe_load(f)

# Load manifest
with open('pipelines/pipeline1.yaml', 'r') as f:
    manifest = yaml.safe_load(f)

# Validate
validate(manifest, schema)
print("[OK] Valid!")
```

### 5.3.3 Integration with CLI

The schema can be integrated into the CLI commands for validation:

```
from smus_cicd.validation import validate_manifest_schema

# In describe command
if not validate_manifest_schema(manifest_path):
    typer.echo("[ERROR] Invalid manifest schema", err=True)
    raise typer.Exit(1)
```

## 5.4 Schema Structure

### 5.4.1 Domain Configuration

```
domain:
  name: cicd-test-domain     # Required: DataZone domain name
  region: us-east-1          # Required: AWS region
```

### 5.4.2 Bundle Configuration

```
bundle:
  bundlesDirectory: ./bundles  # Optional: Bundle output directory
  workflow:                    # Optional: Workflow bundle config
```

```
    - connectionName: default.s3_shared
      append: true              # Optional: Append vs replace
      include: ['workflows/']   # Optional: Include patterns
      exclude: ['*.pyc']        # Optional: Exclude patterns
  storage:                     # Optional: Storage bundle config
    - connectionName: default.s3_shared
      append: false
      include: ['src/']
  git:                         # Optional: Git repository
    repository: my-repo
    url: https://github.com/user/repo.git
    targetDir: ./src
```

### 5.4.3 Target Configuration

```
targets:
  dev:                         # Target name (required)
    stage: DEV                 # Optional: Stage identifier
    default: true              # Optional: Default target flag
    project:                   # Required: Project config
      name: dev-project        # Required: Project name
    initialization:            # Optional: Init config
      project:                 # Optional: Project creation
        create: true           # Optional: Auto-create project
        profileName: 'All capabilities'
        owners: [Eng1]         # Optional: Project owners
        contributors: []       # Optional: Project contributors
      environments:            # Optional: Environment configs
        - EnvironmentConfigurationName: 'OnDemand Workflows'
    bundle_target_configuration: # Optional: Target-specific bundle config
      storage:
        connectionName: default.s3_shared
        directory: 'src'
      workflows:
        connectionName: default.s3_shared
        directory: 'workflows'
    workflows:                 # Optional: Target-specific workflows
      - workflowName: prepareData
        parameters:
          stage_database: DevDB
```

### 5.4.4 Workflow Configuration

```
workflows:
  - workflowName: test_dag           # Required: Workflow name
    connectionName: project.workflow_mwaa  # Required: Connection
    triggerPostDeployment: true      # Optional: Auto-trigger
    engine: MWAA                     # Optional: Engine type
    parameters:                       # Optional: Workflow parameters
      default-sql-connection: project.athena
    logging: console                  # Optional: Logging config
```

## 5.5 Validation Rules

### 5.5.1 Naming Conventions

- **Pipeline names**: Must start with letter, contain only alphanumeric, underscore, hyphen
- **Target names**: Must start with letter, contain only alphanumeric, underscore, hyphen
- **Regions**: Must match AWS region pattern (e.g., `us-east-1`)

### 5.5.2 Constraints

- At least one target must be defined
- Only string, number, or boolean values allowed in parameters
- Connection names should follow DataZone naming conventions
- File patterns should use forward slashes

### 5.5.3 Optional vs Required

- Most fields are optional to accommodate different use cases
- Only core identification fields are required
- Schema allows for flexible manifest structures

## 5.6 Common Patterns

### 5.6.1 Dev-Only Pipeline

```
pipelineName: DevOnlyPipeline
domain:
  name: my-domain
  region: us-east-1
targets:
  dev:
    default: true
    project:
      name: dev-project
```

### 5.6.2 Multi-Target Pipeline

```
pipelineName: MultiTargetPipeline
domain:
  name: my-domain
  region: us-east-1
targets:
  dev:
    default: true
    project:
      name: dev-project
  test:
    project:
      name: test-project
    initialization:
      project:
        create: true
        owners: [Eng1]
  prod:
    project:
      name: prod-project
    initialization:
      project:
        create: true
        owners: [Eng1]
```

## 5.7 Error Handling

The validation script provides detailed error messages including: - **Path**: Location of the error in the manifest - **Message**: Description of the validation failure - **Expected**: What the schema expected (for enum/pattern violations)

Example error output:

```
[ERROR] INVALID - Found 2 schema violations:
  1. Path: domain -> region
     Error: 'invalid-region' does not match '^[a-z0-9-]+$'
     Expected: ^[a-z0-9-]+$

  2. Path: targets -> dev -> project
     Error: 'name' is a required property
```

## 5.8 Future Enhancements

1. **IDE Integration**: Add schema reference to YAML files for IDE validation
2. **CLI Integration**: Integrate validation into describe/bundle commands
3. **Schema Versioning**: Add version field and backward compatibility
4. **Custom Validators**: Add business logic validation beyond JSON Schema
5. **Documentation Generation**: Auto-generate docs from schema
6. **Template Generation**: Create manifest templates from schema

## 6 Cli Commands

*Source: docs/cli-commands.md*

## 7 CLI Commands Reference

<- Back to Main README

The SMUS CLI provides seven main commands for managing CI/CD pipelines in SageMaker Unified Studio.

### 7.1 Command Overview

| Command | Purpose | Example |
|---|---|---|
| `create` | Create new pipeline manifest | `smus-cli create --output pipeline.yaml` |
| `describe` | Validate and show pipeline configuration | `smus-cli describe --pipeline pipeline.yaml --connect` |
| `bundle` | Package files from source environment | `smus-cli bundle --targets dev` |
| `deploy` | Deploy bundle to target environment | `smus-cli deploy --targets test` |
| `run` | Execute workflow commands | `smus-cli run --workflow dag_name --command trigger` |
| `monitor` | Monitor workflow status | `smus-cli monitor --pipeline pipeline.yaml` |
| `test` | Run tests for pipeline targets | `smus-cli test --targets marketing-test-stage` |
| `delete` | Remove target environments | `smus-cli delete --targets marketing-test-stage --force` |

### 7.2 Detailed Command Examples

#### 7.2.1 1. Describe Pipeline Configuration

```
smus-cli describe --pipeline pipelines/pipeline1.yaml --connect
```

**Example Output:**

```
Pipeline: IntegrationTestMultiTarget
Domain: cicd-test-domain (us-east-1)

Targets:
  - dev: dev-marketing
    Project ID: your-dev-project-id
    Status: ACTIVE
    Owners: Admin, eng1
    Connections:
      default.s3_shared:
        connectionId: dqbxjn28zehzjb
        type: S3
        region: us-east-1
        awsAccountId: 123456789012
        description: This is the connection to interact with s3 shared storage location if enal
        s3Uri: s3://sagemaker-unified-studio-123456789012-us-east-1-your-domain-name/dzd_your-c
        status: READY
      project.athena:
        connectionId: amp1omxvjo3kiv
        type: ATHENA
        region: us-east-1
        awsAccountId: 123456789012
        description: This is a default ATHENA connection.
        workgroup: workgroup-your-dev-project-id-xyz123
      project.spark.compatibility:
        connectionId: 6236xbz8cowo4n
        type: SPARK
        region: us-east-1
        awsAccountId: 123456789012
        description: Glue-ETL compute with Permission Mode set to compatibility. (Auto-created
        glueVersion: 5.0
        workerType: G.1X
        numberOfWorkers: 10
      project.workflow_mwaa:
        connectionId: d5jq3vs4ol9s13
        type: WORKFLOWS_MWAA
        region: us-east-1
        awsAccountId: 123456789012
        description: Connection for MWAA environment
        environmentName: SageMaker Unified StudioMWAAEnv-dzd_your-domain-id-your-dev-project-i

Manifest Workflows:
  - test_dag (Connection: project.workflow_mwaa, Engine: MWAA)
  - runGettingStartedNotebook (Connection: project.workflow_mwaa, Engine: MWAA)
```

**What this shows:** The describe command validates your pipeline configuration and displays the structure of your CI/CD pipeline. It shows each target environment (dev, test, prod) with their associated SageMaker Unified Studio projects, available connections for data storage and workflow execution, and the workflows defined in your manifest. This is essential for understanding your pipeline setup and ensuring all resources are properly configured before deployment.

### 7.2.2 2. Create Bundle from Dev Environment

```
smus-cli bundle --pipeline pipelines/pipeline1.yaml --targets dev
```

**Example Output:**

```
Creating bundle for target: dev
Project: dev-marketing
Downloading workflows from S3: default.s3_shared (append: True)
  Downloaded: workflows/dags/test_dag.py
  Downloaded: workflows/.visual/runGettingStartedNotebook.wf
  Downloaded 17 workflow files from S3
Downloading storage from S3: default.s3_shared (append: False)
  Downloaded: src/test-notebook1.ipynb
  Downloaded 1 storage files from S3
Creating archive: IntegrationTestMultiTarget.zip
[OK] Bundle created: ./bundles/IntegrationTestMultiTarget.zip (279462 bytes)

[BUNDLE] Bundle Contents:
==================================================
|---- storage/
|   `---- src/
|       `---- test-notebook1.ipynb
`---- workflows/
    |---- .visual/
    |   `---- runGettingStartedNotebook.wf
    |---- dags/
    |   |---- test_dag.py
    |   `---- visual/
    |       `---- runGettingStartedNotebook.py
    `---- config/
        |---- requirements.txt
        `---- startup.sh
==================================================
[STATS] Total files: 18
Bundle creation complete for target: dev
```

**What this shows:** The bundle command downloads all workflows and storage files from your development environment and packages them into a deployment-ready ZIP file. This creates a snapshot of your current development state that can be deployed to other environments. The bundle contains both workflow files (DAGs, notebooks) and storage assets, ensuring consistent deployments across environments.

### 7.2.3 3. Deploy to Test Environment

```
smus-cli deploy --targets test --pipeline pipelines/pipeline1.yaml
```

**Example Output:**

```
Deploying to target: test
Project: integration-test-test
Domain: cicd-test-domain
Region: us-east-1
[CONFIG] Auto-initializing target infrastructure...
[OK] Target infrastructure ready
```

```
[OK] Project 'integration-test-test' exists
Bundle file: ./bundles/IntegrationTestMultiTarget.zip
Deploying storage to: default.s3_shared/src (append: False)
  S3 Location: s3://sagemaker-unified-studio-123456789012-us-east-1-your-domain-name/.../sharec
    Synced: test-notebook1.ipynb
  Storage files synced: 1
Deploying workflows to: default.s3_shared/workflows (append: True)
  S3 Location: s3://sagemaker-unified-studio-123456789012-us-east-1-your-domain-name/.../sharec
    Synced: test_dag.py
    Synced: runGettingStartedNotebook.py
  Workflow files synced: 17
[OK] Deployment complete! Total files synced: 18

[DEPLOY] Starting workflow validation...
[OK] MWAA environment is available
[NEW] New DAGs detected: runGettingStartedNotebook
```

**What this shows:** The deploy command uploads your bundled files to the target environment and synchronizes them with the SageMaker Unified Studio project's storage and workflow connections. It shows the deployment progress, file counts, and validates that the MWAA environment can access the new workflows. This ensures your code changes are properly deployed and ready for execution.

### 7.2.4 4. Monitor Workflow Status

```
smus-cli monitor --pipeline pipelines/pipeline1.yaml
```

**Example Output:**

```
Pipeline: IntegrationTestMultiTarget
Domain: cicd-test-domain (us-east-1)

[SEARCH] Monitoring Status:

[TARGET] Target: test
   Project: integration-test-test
   Project ID: your-test-project-id
   Status: ACTIVE
   Owners: Admin, eng1

   [STATS] Workflow Status:
      [CONFIG] project.workflow_mwaa (SageMaker Unified StudioMWAAEnv-dzd_your-domain-id-your-
          Airflow UI: https://your-mwaa-environment.airflow.us-east-1.on.aws
           test_dag
            Schedule: Manual | Status: ACTIVE | Recent: Unknown
           runGettingStartedNotebook
            Schedule: Manual | Status: ACTIVE | Recent: Unknown

 Manifest Workflows:
   - test_dag (Connection: project.workflow_mwaa)
   - runGettingStartedNotebook (Connection: project.workflow_mwaa)
```

**What this shows:** The monitor command provides real-time status of your pipeline's workflow environments. It displays project information, workflow connection details, and the current state of all DAGs in your MWAA environments. This is essential for tracking workflow health, identifying issues, and understanding the operational status of your data pipelines across different environments.

### 7.2.5 5. Trigger Workflow Execution

```
smus-cli run --pipeline pipelines/pipeline1.yaml --targets test --workflow test_dag --command t
```

**Example Output:**

```
[TARGET] Target: test
[CONFIG] Connection: project.workflow_mwaa (SageMaker Unified StudioMWAAEnv-dzd_your-domain-id-
 Command: trigger
[OK] Workflow triggered successfully
[OUTPUT] Run ID: manual__2025-08-25T11:45:00+00:00
```

**What this shows:** The run command executes Airflow CLI commands against your MWAA environments. In this example, it triggers a workflow execution and returns the run ID for tracking. This allows you to programmatically control workflow execution, check status, and manage your data pipelines from the command line.

### 7.2.6 6. Run Tests

```
smus-cli test --pipeline pipelines/pipeline1.yaml --targets marketing-test-stage
```

**Example Output:**

```
Pipeline: IntegrationTestMultiTarget
Domain: cicd-test-domain (us-east-1)

[TARGET] Target: test
   Test folder: tests/
  [CONFIG] Project: integration-test-test (your-project-id)
  [TEST] Running tests...
  [OK] Tests passed

[TARGET] Test Summary:
  [OK] Passed: 1
  [ERROR] Failed: 0
  [WARNING]  Skipped: 0
   Errors: 0
```

**What this shows:** The test command runs Python tests from the configured test folder against your deployed pipeline. Tests receive environment variables with domain ID, project ID, and other context information to validate the deployment. This ensures your pipeline is working correctly after deployment and provides automated validation of your data workflows.

#### 7.2.7 7. Clean Up Resources

```
smus-cli delete --targets test --pipeline pipelines/pipeline1.yaml --force
```

**Example Output:**

```
Pipeline: IntegrationTestMultiTarget
Domain: cicd-test-domain (us-east-1)

Targets to delete:
  - test: integration-test-test

[DELETE]  Deleting target: test
[OK] Successfully deleted project: integration-test-test

[TARGET] Deletion Summary
  [OK] test: Project deleted successfully
```

**What this shows:** The delete command removes SageMaker Unified Studio projects and their associated resources. It provides a summary of deletion operations, showing which projects were successfully removed. This is useful for cleaning up test environments and managing resource lifecycle in your CI/CD pipeline.

```
smus-cli --help
```

#### 7.2.8 Pipeline Commands

0. **`create`** - Create new pipeline manifest
1. **`describe`** - Describe and validate pipeline configuration
2. **`bundle`** - Create deployment packages from source
3. **`deploy`** - Deploy packages to targets (auto-initializes if needed)
4. **`monitor`** - Monitor workflow status
5. **`run`** - Run Airflow CLI commands
6. **`delete`** - Delete projects and environments

### 7.3 Command Details

#### 7.3.1 0. create - Create New Pipeline Manifest

Creates a new pipeline manifest file with basic structure.

```
smus-cli create [OPTIONS]
```

##### 7.3.1.1 Options

- **`-o, --output`**: Output file path for the new pipeline manifest (default: `pipeline.yaml`)
- **`-n, --name`**: Pipeline name (optional, will use placeholder if not provided)
- **`-t, --targets`**: Target name(s) - single target or comma-separated list (optional)
- **`--help`**: Show command help

##### 7.3.1.2 Examples

```
# Create basic pipeline manifest
smus-cli create

# Create with custom output file and name
smus-cli create -o my-pipeline.yaml -n MyPipeline

# Create with specific targets
smus-cli create -o pipeline.yaml -t dev,test,prod
```

### 7.3.2 1. describe - Describe Pipeline Configuration

Validates and displays information about your pipeline manifest.

```
smus-cli describe [OPTIONS]
```

#### 7.3.2.1 Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)
- **`-t, --targets`**: Target name(s) - single target or comma-separated list (optional, defaults to all targets)
- **`-o, --output`**: Output format: TEXT (default) or JSON
- **`-w, --workflows`**: Show workflow information
- **`-c, --connections`**: Show connection information
- **`--connect`**: Connect to AWS account and pull additional information
- **`--help`**: Show command help

#### 7.3.2.2 Examples

```
# Basic describe
smus-cli describe

# Describe specific targets with workflows
smus-cli describe -t dev,test -w

# Describe with AWS connection info in JSON format
smus-cli describe --connect -o JSON

# Describe specific pipeline file
smus-cli describe -p my-pipeline.yaml
```

### 7.3.3 2. bundle - Create Deployment Packages

Creates bundle zip files by downloading from S3.

```
smus-cli bundle [OPTIONS]
```

#### 7.3.3.1 Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)
- **`-t, --targets`**: Target name(s) - single target or comma-separated list (uses default target if not specified)

- **-d, --output-dir**: Output directory for bundle files (default: `./bundles`)
- **-o, --output**: Output format: TEXT (default) or JSON
- **--help**: Show command help

#### 7.3.3.2 Examples

```
# Bundle default target
smus-cli bundle

# Bundle specific targets
smus-cli bundle -t dev,test

# Bundle to custom directory
smus-cli bundle -d /path/to/bundles

# Bundle with JSON output
smus-cli bundle -o JSON
```

### 7.3.4 3. deploy - Deploy to Targets

Deploys bundle files to target environments (auto-initializes if needed).

```
smus-cli deploy [OPTIONS]
```

#### 7.3.4.1 Options

- **-p, --pipeline**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)
- **-t, --targets**: Target name(s) - single target or comma-separated list (uses default target if not specified)
- **--help**: Show command help

#### 7.3.4.2 Examples

```
# Deploy to default target
smus-cli deploy

# Deploy to specific targets
smus-cli deploy -t test,prod

# Deploy specific pipeline
smus-cli deploy -p my-pipeline.yaml -t prod
```

### 7.3.5 4. monitor - Monitor Workflow Status

Monitors workflow status across target environments.

```
smus-cli monitor [OPTIONS]
```

#### 7.3.5.1 Options

- **-p, --pipeline**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)

- **-t, --targets**: Target name(s) - single target or comma-separated list (shows all targets if not specified)
- **-o, --output**: Output format: TEXT (default) or JSON
- **--help**: Show command help

#### 7.3.5.2 Examples

```
# Monitor all targets
smus-cli monitor

# Monitor specific targets
smus-cli monitor -t dev,test

# Monitor with JSON output
smus-cli monitor -o JSON
```

### 7.3.6 5. run - Run Airflow CLI Commands

Executes Airflow CLI commands on target environments.

```
smus-cli run [OPTIONS]
```

#### 7.3.6.1 Options

- **-w, --workflow**: Workflow name to target (required)
- **-c, --command**: Airflow command to execute (required)
- **-t, --targets**: Target name(s) - single target or comma-separated list (optional, defaults to first available)
- **-p, --pipeline**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)
- **-o, --output**: Output format: TEXT (default) or JSON
- **--help**: Show command help

#### 7.3.6.2 Examples

```
# Run Airflow version command
smus-cli run -w my_dag -c version

# Run DAG list command on specific target
smus-cli run -w my_dag -c "dags list" -t prod

# Run with JSON output
smus-cli run -w my_dag -c version -o JSON
```

### 7.3.7 6. delete - Delete Target Environments

Deletes DataZone projects and associated resources for specified targets.

```
smus-cli delete [OPTIONS]
```

#### 7.3.7.1 Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipelines/pipeline1.yaml`)
- **`-t, --targets`**: Target name(s) - single target or comma-separated list (required)
- **`-f, --force`**: Skip confirmation prompt
- **`--async`**: Don't wait for deletion to complete
- **`-o, --output`**: Output format: TEXT (default) or JSON
- **`--help`**: Show command help

#### 7.3.7.2 Examples

```
# Delete single target with confirmation
smus-cli delete -t test

# Delete multiple targets without confirmation
smus-cli delete -t test,prod --force

# Delete asynchronously (don't wait for completion)
smus-cli delete -t test --force --async

# Delete with JSON output
smus-cli delete -t test --force -o JSON
```

#### 7.3.7.3 Behavior

- **Confirmation Required**: By default, prompts for confirmation before deletion
- **Force Mode**: `--force` skips confirmation and deletes immediately
- **Async Mode**: `--async` returns immediately without waiting for completion
- **Error Handling**: Properly handles AWS errors (e.g., projects with MetaDataForms)
- **Resource Cleanup**: Deletes DataZone projects and associated CloudFormation stacks

#### 7.3.7.4 Notes

- Some DataZone projects cannot be deleted if they contain MetaDataForms
- CloudFormation stacks are deleted automatically when projects are removed
- Use `--async` for faster execution when managing multiple targets

## 7.4 Global Options

All commands support: - **`--help`**: Show command help

## 7.5 Exit Codes

- **0**: Success
- **1**: Error (check error message for details)

## 7.6 Configuration Files

### 7.6.1 Pipeline Manifest

- Default location: `pipelines/pipeline1.yaml`

- Override with `--pipeline` option
- See Pipeline Manifest Reference for format

### 7.6.2 AWS Configuration

- Uses standard AWS credential chain
- Supports AWS profiles and environment variables
- Region can be specified in pipeline manifest or AWS config

## 7.7 Common Workflows

### 7.7.1 Development Workflow

```
# 1. Create new pipeline
smus-cli create -o my-pipeline.yaml

# 2. Validate configuration
smus-cli describe -p my-pipeline.yaml

# 3. Create bundle from dev
smus-cli bundle -p my-pipeline.yaml -t dev

# 4. Deploy to test
smus-cli deploy -p my-pipeline.yaml -t test

# 5. Monitor deployment
smus-cli monitor -p my-pipeline.yaml -t test

# 6. Run workflow commands
smus-cli run -w my_dag -c "dags list" -t test
```

### 7.7.2 Cleanup Workflow

```
# Delete test environment
smus-cli delete -t test --force

# Delete multiple environments
smus-cli delete -t test,staging --force --async
```

## 8 Github Actions Integration

*Source: docs/github-actions-integration.md*

## 9 GitHub Actions CI/CD Integration

The SMUS CLI can be integrated with GitHub Actions to create automated CI/CD pipelines that deploy your data science workflows across multiple environments.

### 9.1 Example GitHub Actions Workflow

Create `.github/workflows/smus-cicd.yml` in your repository:

```yaml
name: SMUS CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  AWS_REGION: us-east-1
  PIPELINE_FILE: pipeline.yaml

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install SMUS CLI
      run: |
        pip install smus-cicd-cli

    - name: Configure AWS Credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Validate Pipeline Configuration
```

```yaml
      run: |
        smus-cli describe --pipeline ${{ env.PIPELINE_FILE }} --connect

  bundle-from-dev:
    needs: validate
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'
    steps:
    - uses: actions/checkout@v4

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install SMUS CLI
      run: pip install smus-cicd-cli

    - name: Configure AWS Credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Describe Development Environment
      run: |
        smus-cli describe --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-dev-stage --

    - name: Create Bundle from Development
      run: |
        smus-cli bundle --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-dev-stage

    - name: Upload Bundle Artifacts
      uses: actions/upload-artifact@v4
      with:
        name: smus-bundle
        path: ./bundles/

  deploy-staging:
    needs: bundle-from-dev
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'
    steps:
    - uses: actions/checkout@v4

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install SMUS CLI
      run: pip install smus-cicd-cli

    - name: Configure AWS Credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Download Bundle Artifacts
      uses: actions/download-artifact@v4
      with:
        name: smus-bundle
        path: ./bundles/

    - name: Deploy to Staging
      run: |
        smus-cli deploy --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-test-stage

    - name: Run Staging Tests
      run: |
        smus-cli test --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-test-stage

    - name: Monitor Workflow Status
      run: |
        smus-cli monitor --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-test-stage

  deploy-production:
    needs: deploy-staging
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production
    steps:
    - uses: actions/checkout@v4

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install SMUS CLI
      run: pip install smus-cicd-cli

    - name: Configure AWS Credentials
```

```
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: Create Bundle from Development
        run: |
          smus-cli bundle --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-dev-stage

      - name: Deploy to Production
        run: |
          smus-cli deploy --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-prod-stage

      - name: Run Production Tests
        run: |
          smus-cli test --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-prod-stage

      - name: Monitor Production Deployment
        run: |
          smus-cli monitor --pipeline ${{ env.PIPELINE_FILE }} --targets marketing-prod-stage
```

## 9.2 Workflow Explanation

This GitHub Actions workflow implements a complete CI/CD pipeline for SMUS deployments:

### 9.2.1 Triggers

- **Push to `develop`**: Deploys to development and staging environments
- **Push to `main`**: Deploys to production (after staging validation)
- **Pull Requests**: Validates pipeline configuration only

### 9.2.2 Pipeline Stages

1. **Validate** (All branches)
   - Validates pipeline configuration
   - Connects to AWS to verify resources and permissions
   - Runs on every push and PR
2. **Bundle from Development** (develop branch only)
   - Describes the existing development environment
   - Creates bundle from `marketing-dev-stage` (where development work is done)
   - Uploads bundle as GitHub Actions artifact for reuse
   - No deployment - dev environment already exists with latest work
3. **Deploy Staging** (develop branch only)
   - Downloads bundle created from development
   - Deploys to `marketing-test-stage` target
   - Runs comprehensive tests
   - Monitors workflow execution
   - Pre-production validation

4. **Deploy Production** (main branch only)
    - Creates fresh bundle from development environment
    - Uses GitHub Environment protection rules
    - Deploys to `marketing-prod-stage` target
    - Runs production tests
    - Monitors deployment status

#### 9.2.3 Required GitHub Secrets

Configure these secrets in your GitHub repository settings:

- `AWS_ACCESS_KEY_ID`: AWS access key for SMUS CLI
- `AWS_SECRET_ACCESS_KEY`: AWS secret key for SMUS CLI

#### 9.2.4 Environment Protection

The production job uses GitHub's `environment: production` feature, which allows you to: - Require manual approval before production deployments - Restrict deployments to specific branches - Add deployment protection rules

#### 9.2.5 Benefits

- **Automated Testing**: Every deployment is automatically tested
- **Environment Progression**: Code flows through dev (source) -> staging -> production
- **Bundle Reuse**: Staging uses the same bundle created from dev environment
- **Development Isolation**: Dev environment is the source, not a deployment target
- **Rollback Safety**: Failed tests prevent promotion to next environment
- **Audit Trail**: Complete deployment history in GitHub Actions
- **Team Collaboration**: Pull request validation ensures code quality

This integration transforms your SMUS pipeline into a fully automated CI/CD system that scales with your team's development workflow.

## 10 Readme

*Source: tests/README.md*

## 11 SMUS CLI Tests

This directory contains unit tests and integration tests for the SMUS CI/CD CLI.

### 11.1 Prerequisites for Integration Tests

Before running integration tests, you must deploy the required AWS infrastructure:

#### 11.1.1 1. Deploy SageMaker Domain

```
cd tests/scripts
./deploy-domain.sh
```

#### 11.1.2 2. Deploy Environment Blueprints and Profiles

```
cd tests/scripts
./deploy-blueprints-profiles.sh
```

#### 11.1.3 3. Deploy Dev Project

```
cd tests/scripts
./deploy-projects.sh
```

**Important:** These scripts must be run in order as they have dependencies on each other. The integration tests require these AWS resources to be deployed and available.

### 11.2 Test Structure

```
tests/
|---- unit/                 # Unit tests (no AWS credentials required)
|   |---- test_describe.py     # Tests for describe command
|   |---- test_bundle.py    # Tests for bundle command
|   `---- test_monitor.py   # Tests for monitor command
|---- integration/          # Integration tests (require AWS credentials)
|   |---- config.yaml       # Default integration test configuration
|   |---- config.local.yaml # Local configuration (create from config.yaml)
|   |---- base.py           # Base class for integration tests
|   |---- basic_pipeline/   # Basic pipeline test suite
|   |   |---- basic_pipeline.yaml        # Basic pipeline configuration
|   |   |---- test_basic_pipeline.py     # Basic pipeline tests
|   |   `---- README.md                  # Test documentation
|   `---- multi_target_pipeline/         # Multi-target pipeline test suite
|       |---- multi_target_pipeline.yaml # Multi-target pipeline configuration
|       |---- test_multi_target_pipeline.py # Multi-target tests
|       `---- README.md                  # Test documentation
`---- requirements.txt      # Test dependencies
```

## 11.3 Running Tests (Python-Native)

### 11.3.1 Prerequisites

Install test dependencies:

```
pip install -r tests/requirements.txt
```

### 11.3.2 Quick Validation Commands

Use the Python validation script for all testing needs:

```
# Unit tests only (no AWS credentials required)
python scripts/validate.py --unit

# Integration tests (automatically refreshes AWS credentials)
python scripts/validate.py --integration

# README examples validation
python scripts/validate.py --readme

# AWS credentials refresh
python scripts/validate.py --aws-login

# Full validation pipeline
python scripts/validate.py --all

# Clean temporary files
python scripts/validate.py --clean
```

### 11.3.3 Unit Tests (No AWS Credentials Required)

Unit tests use mocks and don't require AWS credentials:

```
# Using validation script (recommended)
python scripts/validate.py --unit

# Or directly with pytest
pytest tests/unit/ -v

# Or legacy method
python run_tests.py --type unit
```

### 11.3.4 Integration Tests (Require AWS Credentials)

Integration tests require valid AWS credentials and may interact with real AWS resources.

#### 11.3.4.1 Setup Integration Tests

1. **Copy configuration file:**

```
cp tests/integration/config.yaml tests/integration/config.local.yaml
```

2. **Edit configuration:**

```yaml
# tests/integration/config.local.yaml
aws:
  profile: your-aws-profile  # or use access keys
  region: us-east-1

test_environment:
  domain_name: your-test-domain
  project_prefix: integration-test
```

3. **Run integration tests:**

```bash
# Using validation script (recommended - auto-refreshes AWS credentials)
python scripts/validate.py --integration

# Or check AWS setup first
python run_integration_tests.py --check-setup

# Or legacy methods
python run_integration_tests.py --type basic
python run_integration_tests.py --type all
```

### 11.3.5 All Tests

```bash
# Full validation (recommended)
python scripts/validate.py --all

# Or legacy method
python run_tests.py --type all
```

## 11.4 Python-Native Testing Approach

The project now uses modern Python tooling:

### 11.4.1 Configuration Files

- `pyproject.toml` - Modern Python project configuration
- `scripts/validate.py` - Python validation script (replaces Makefile)

### 11.4.2 Validation Script Features

- [OK] Automatic AWS credential refresh with `isenguardcli`
- [OK] Unit test execution
- [OK] Integration test execution

- [OK] README example validation
- [OK] Cleanup utilities
- [OK] Cross-platform compatibility

### 11.4.3 Alternative pytest Commands

```
# Direct pytest usage
pytest tests/unit/                          # Unit tests
pytest tests/integration/ -m "not slow"     # Integration tests
pytest tests/ -m "not slow" -v             # All tests excluding slow ones
```

## 11.5 Integration Test Features

### 11.5.1 Test Scenarios

Each integration test uses its own pipeline configuration:

- `basic_pipeline.yaml` - Single target pipeline
- `multi_target_pipeline.yaml` - Multiple targets (dev, test, prod)

### 11.5.2 Test Categories

- **Basic Tests** - Describe, bundle, monitor commands
- **Multi-Target Tests** - Operations across multiple targets
- **Validation Tests** - Error handling and edge cases
- **Slow Tests** - Full end-to-end workflows (marked with `@pytest.mark.slow`)

### 11.5.3 Test Workflow

Each integration test follows this pattern:

1. **Setup** - Configure AWS credentials and test environment
2. **Describe** - Validate pipeline configuration
3. **Bundle** - Create deployment packages (may fail if resources don't exist)
4. **Monitor** - Check pipeline status
5. **Cleanup** - Remove temporary resources
6. **Report** - Generate success/failure report

### 11.5.4 Expected Behavior

Integration tests are designed to be **informative** rather than strictly pass/fail:

- [OK] **Commands execute successfully** - CLI works correctly
- [WARNING] **Expected failures** - Missing AWS resources (marked as success)
- [ERROR] **Unexpected failures** - CLI bugs or configuration issues

## 11.6 Test Categories

### 11.6.1 Unit Tests

- **No AWS credentials required**
- Use mocks for AWS services
- Test CLI command logic and parsing
- Fast execution
- Safe to run in CI/CD

#### 11.6.2 Integration Tests

- **Require AWS credentials**
- May interact with real AWS resources
- Test end-to-end workflows
- Slower execution
- Should be run in dedicated test environments

## 11.7 Environment Variables

### 11.7.1 For Integration Tests

- `AWS_PROFILE` or `AWS_ACCESS_KEY_ID`/`AWS_SECRET_ACCESS_KEY` - AWS credentials
- `AWS_DEFAULT_REGION` - AWS region (defaults to us-east-1)

### 11.7.2 Test Markers

- `@pytest.mark.slow` - Slow running tests
- `@pytest.mark.integration` - Integration tests
- `@pytest.mark.unit` - Unit tests
- `@pytest.mark.aws` - Tests requiring AWS credentials

## 11.8 Example Usage

```
# Recommended Python-native approach
python scripts/validate.py --unit          # Fast unit tests
python scripts/validate.py --integration   # Integration tests with auto AWS login
python scripts/validate.py --readme        # Validate README examples
python scripts/validate.py --all           # Full validation pipeline

# Alternative pytest commands
pytest tests/unit/ -v                      # Unit tests only
pytest tests/integration/ -m "not slow" -v  # Integration tests excluding slow ones
pytest tests/integration/test_basic_pipeline.py -v  # Specific test file

# Legacy commands (still supported)
python run_tests.py --type unit
python run_integration_tests.py --type basic
```

## 11.9 Integration Test Configuration

The `config.yaml` file controls integration test behavior:

```
aws:
  profile: default
  region: us-east-1

test_environment:
  domain_name: integration-test-domain
  project_prefix: integration-test
```

```yaml
  cleanup_after_tests: true

test_scenarios:
  basic_pipeline:
    enabled: true
    pipeline_file: "basic_pipeline.yaml"
  multi_target_pipeline:
    enabled: true
    pipeline_file: "multi_target_pipeline.yaml"

timeouts:
  project_creation: 300
  bundle_creation: 120
```

## 11.10 AWS Credential Management

The validation script automatically handles AWS credentials:

```bash
# Manual credential refresh
python scripts/validate.py --aws-login

# Integration tests automatically refresh credentials
python scripts/validate.py --integration
```

This uses `isenguardcli` internally and verifies credentials with `aws sts get-caller-identity`.

## 12 Code Assist Script

*Source: code-assist-script.md*

## 13 Code Assist Script

### 13.1 Automated Workflow for Code Changes

When making any code changes to the SMUS CI/CD CLI, follow this automated workflow to ensure consistency and quality:

#### 13.1.1 0. AWS Credentials Setup (when needed)

```
# Refresh AWS credentials using validation script
python scripts/validate.py --aws-login

# Or manually:
isenguardcli
aws sts get-caller-identity
```

#### 13.1.2 1. Pre-Change Validation

```
# Verify current state is clean
python scripts/validate.py --unit
python scripts/validate.py --integration
git status
```

#### 13.1.3 2. Make Code Changes

- Implement the requested feature/fix
- Update relevant docstrings and comments

#### 13.1.4 3. Update Test Cases

```
# Run tests to identify failures
python scripts/validate.py --unit

# Fix any failing tests by:
# - Updating test expectations to match new behavior
# - Adding new test cases for new functionality
# - Ensuring mock objects match actual implementation
# - Verifying CLI parameter usage is correct
```

#### 13.1.5 4. Update README and Documentation

```
# Update README.md if:
# - CLI syntax changed
# - New commands added
# - Examples need updating
# - Diagrams need modification
# Verify examples work:
python scripts/validate.py --readme
```

#### 13.1.6 5. Integration Test Validation

```
# Run integration tests (automatically refreshes AWS credentials)
python scripts/validate.py --integration
```

#### 13.1.7 6. Final Validation and Commit

```
# Full validation pipeline
python scripts/validate.py --all

# Commit changes
git add .
git commit -m "Descriptive commit message

- List specific changes made
- Note test updates
- Note documentation updates"

# Verify clean state
git status
```

### 13.2 Python-Native Validation Commands

```
# Quick validation options
python scripts/validate.py --unit          # Unit tests only
python scripts/validate.py --integration   # Integration tests only
python scripts/validate.py --readme        # README examples only
python scripts/validate.py --aws-login     # AWS credentials only
python scripts/validate.py --clean         # Clean temp files
python scripts/validate.py --all           # Full validation (default)

# Alternative using pytest directly
pytest tests/unit/                         # Unit tests
pytest tests/integration/ -m "not slow"    # Integration tests
```

### 13.3 Checklist for Any Code Change

- ☐ AWS credentials refreshed (when needed)
- ☐ Unit tests pass (95/95)
- ☐ Integration tests pass (basic suite)
- ☐ README examples are accurate and tested
- ☐ CLI help text is updated if needed
- ☐ New functionality has corresponding tests
- ☐ Mock objects match real implementation
- ☐ CLI parameter usage is consistent

- ☐ Documentation reflects actual behavior
- ☐ Check that the code and markdown files don't contain aws account ids , nor web addresses, or host names. mask all of these before committing.
- ☐ All changes are committed

## 13.4 Common Test Patterns to Maintain

### 13.4.1 Unit Test Patterns

- Use `--targets` not `--target` in CLI tests
- Mock objects need proper attributes, not dictionaries
- Test expectations should match actual output format
- Use proper patch decorators for dependencies

### 13.4.2 Integration Test Patterns

- Use `["describe", "--pipeline", file]` not `["describe", file]`
- Use `--connect` not `--targets --connect`
- Expected exit codes should match test framework expectations
- Rename DAG files to avoid pytest collection (`.dag` extension)

### 13.4.3 README Patterns

- All CLI examples use correct parameter syntax
- Include realistic command outputs
- Keep examples concise but informative
- Verify examples actually work before documenting

## 13.5 Project Structure (Python-Native)

```
smus_cicd/
|---- pyproject.toml          # Modern Python project config
|---- scripts/
|   `---- validate.py         # Validation script (replaces Makefile)
|---- smus_cicd/            # Main package
|---- tests/                # Test suite
`---- README.md             # Documentation
```

## 13.6 AWS Credential Management

When you say "refresh your aws credentials", I will run:

```
python scripts/validate.py --aws-login
```

This ensures integration tests that require AWS access will work properly.

This script ensures that every code change maintains the quality and consistency of the codebase using Python-native tools.

# 14 Development

*Source: DEVELOPMENT.md*

# 15 SMUS-CICD-pipeline-cli

This is a BrazilPython 3 Python project.

## 15.1 Choosing your Python version

This is a change from BrazilPython 2; in BP3 you choose your Python version using branches in your versionset. By default the version is inherited from `live` (which as of this writing is CPython 3.4, but that is subject to change). The actual version can be chosen using the singleton interpreter process.

The short version of that is:

### 15.1.1 Using CPython3

Build the following package major version/branches into your versionset:

- `Python-default` : `live`
- `CPython3-default` : `live`

This will cause `bin/python` to run `python3.7` as of 03/2020 but over time this version will be kept up to date with the current best practice interpreters.

Your default interpreter is always enabled as a build target for python packages in your version set.

You should build the `no` branches for all interpreters into your versionset as well, since the runtime interpreter will always build:

- `CPython27-build` : `no`
- `CPython34-build` : `no`
- `CPython36-build` : `no`
- `CPython37-build` : `no`
- `CPython38-build` : `no`
- `Jython27-build` : `no`

(Note that many of these are off in `live` already)

### 15.1.2 Using a newer version of CPython3

If you need a special version of CPython (say you want to be on the cutting edge and use 3.9):

- `Python-default` : `live`
- `CPython3-default` : `CPython39`

This will cause `bin/python` to run `python3.9`

### 15.1.3 Using CPython2 2.7 or Jython

**Don't**

## 15.2 Building

Modifying the build logic of this package just requires overriding parts of the setuptools process. The entry point is either the `release`, `build`, `test`, or `doc` commands, all of which are implemented as setuptools commands in the BrazilPython-Base-3.0 package.

### 15.2.1 Restricting what interpreter your package will attempt to build for

If you want to restrict the set of Python versions a package builds for, first answer these questions

1. Do you need to build into version sets that may have more than the default interpreter enabled (such as live)?
2. Are there versions that are commonly enabled in those version sets that would be difficult to support?
3. Example: Python 3.6 is currently enabled in `live` but if you want to publish a package to live that is only valid for Python 3.7+ consumers, then you may want to filter on it
4. Counter example: while Jython is a valid build target, it has largely been deprecated from use and is not enabled in the vast majority of version sets, so filtering on it will add almost entirely unused cruft to your package when you may have no Jython-enabled consumers
5. Should the build fail if no valid interpreter is enabled?

If your answer to all of those is `yes`, then you may want to make a filter for your package.

Do so by creating an executable script named `build-tools/bin/python-builds` in this package, and having it exit non-zero when the undesirable versions build. By default, packages without this file package will build for every version of Python in your versionset.

The version strings that'll be passed in are:

- CPython##
- Jython##

Commands that only run for one version of Python will be run for the version in the `default_python` argument to `setup()` in `setup.py`. `doc` is one such command, and is configured by default to run the `doc_command` as defined in `setup.py`, which will build Sphinx documentation.

An example can be found here.

#### 15.2.1.1 Best practices for filtering

1. Use forwards-compatible filters (i.e. `$version -ge 37`). This will make it painless to test and update when you update your default
2. Don't tie to older versions. This is expensive technical debt that paying it down sooner is far better than chaining yourself (and your consumers) to older interpreters
3. If you want to specifically only build for the default interpreter, you can add the filter `[[ $1 == "$(default-python-package-name)" ]] || exit 1`
4. **Only do this if you intend to vend an executable that is specifically getting run with the default interpreter, for integration test packages, or for packages that only should be built for a single interpreter (such as a data-generation or activation-scripts package)**

## 15.3 Testing

`brazil-build test` will run the test command defined in `setup.py`, by default `brazilpython_pytest`, which is defined in the BrazilPython-Pytest-3.0 package. The arguments for this will be taken from `setup.cfg`'s `[tool:pytest]` section, but can be set in `pytest.ini` if that's your thing too. Coverage is enabled by default, provided by pytest-cov, which is part of the `PythonTestingDependencies` package group.

## 15.4 Running

(For details, check out the FAQ)

To run a script in your bin/ directory named `my-script` with its default shebang, you just do this:

```
brazil-runtime-exec *my-script*
```

To run the default interpreter for experimentation:

```
brazil-runtime-exec python
```

## 15.5 Deploying

If this is a library, nothing needs to be done; it'll deploy the versions it builds. If you intend to ship binaries, add a dependency on Python = default to `Config`, and then ensure that the right branch of `Python-default` is built into your versionset. You'll want either CPython2 or CPython3 for CPython.

# 16 Readme

*Source: test/README.md*

By default, this package is configured to run PyTest tests (http://pytest.org/).

## 16.1 Writing tests

Place test files in this directory, using file names that start with `test_`.

## 16.2 Running tests

```
$ brazil-build test
```

To configure pytest's behaviour in a single run, you can add options using the –addopts flag:

```
$ brazil-build test --addopts="[pytest options]"
```

For example, to run a single test, or a subset of tests, you can use pytest's options to select tests:

```
$ brazil-build test --addopts="-k TEST_PATTERN"
```

Code coverage is automatically reported for smus_cicd_pipeline_cli; to add other packages, modify setup.cfg in the package root directory.

To debug the failing tests:

```
$ brazil-build test --addopts=--pdb
```

This will drop you into the Python debugger on the failed test.

### 16.2.1 Importing tests/fixtures

The `test` module is generally not direcrtly importable and it's generally acceptable to use relative imports inside test cases.

### 16.2.2 Fixtures

Pytest provides `conftest.py` as a mechanism to store test fixtures. However, there may be times when it makes sense to include a `test/fixtures` module to locate complex or large fixtures.

### 16.2.3 Common Errors

**16.2.3.1 ModuleNotFoundError: No module named "test.fixtures"** The `test` and sometimes `test/fixtures` modules need to be importable. To allow these to be importable, create a `__init__.py` file in each directory. - `test/__init__.py` - `test/fixtures/__init__.py` (optional)

# 17 Readme

*Source: tests/integration/multi_target_pipeline/README.md*

# 18 Multi-Target Pipeline Integration Test

This test validates the multi-target pipeline workflow with dev, test, and prod environments.

## 18.1 Files

- `multi_target_pipeline.yaml` - Pipeline configuration with multiple targets
- `test_multi_target_pipeline.py` - Integration test implementation

## 18.2 Test Scenarios

### 18.2.1 1. Multi-Target Describe

- Describe pipeline with multiple targets
- Verify all targets (dev, test, prod) are present

### 18.2.2 2. Target Specific Operations

- Bundle creation for each target individually
- Monitor each target individually
- Validate target-specific output

### 18.2.3 3. Pipeline Workflow Comparison

- Compare basic vs multi-target pipeline behavior
- Validate target count differences

### 18.2.4 4. Sequential Target Deployment (Slow)

- Deploy to targets in sequence (dev -> test)
- Monitor all targets after deployment
- Skip prod for safety

## 18.3 Expected Behavior

- [OK] Describe should identify all 3 targets
- [WARNING] Bundle/Monitor may fail per target if resources don't exist
- [OK] Target-specific commands should reference correct target

## 18.4 Usage

```
# Run multi-target pipeline tests
python -m pytest tests/integration/multi_target_pipeline/ -v

# Run specific test
python -m pytest tests/integration/multi_target_pipeline/test_multi_target_pipeline.py::TestMul
```

## 19 Readme

*Source: tests/integration/basic_pipeline/README.md*

## 20 Basic Pipeline Integration Test

This test validates the basic single-target pipeline workflow.

### 20.1 Files

- `basic_pipeline.yaml` - Pipeline configuration with single dev target
- `test_basic_pipeline.py` - Integration test implementation

### 20.2 Test Scenarios

#### 20.2.1 1. Basic Pipeline Workflow

- Parse pipeline configuration
- Parse with workflows flag
- Parse with targets flag
- Attempt bundle creation
- Monitor pipeline status

#### 20.2.2 2. Pipeline Validation

- Valid pipeline file parsing
- Error handling for non-existent files
- Bundle with specific target
- Monitor specific target

#### 20.2.3 3. Full Pipeline with Resources (Slow)

- End-to-end workflow with actual AWS resources
- Requires configured DataZone domain and projects

### 20.3 Expected Behavior

- [OK] Parse commands should always succeed
- [WARNING] Bundle/Monitor may fail if AWS resources don't exist (expected)
- [ERROR] Unexpected CLI errors indicate bugs

### 20.4 Usage

```
# Run basic pipeline tests
python -m pytest tests/integration/basic_pipeline/ -v

# Run with markers
python -m pytest tests/integration/basic_pipeline/ -m "integration and not slow" -v
```