SMUS CI/CD Pipeline CLI

Automate deployment of data science workflows across SageMaker Unified Studio environments

Deploy Airflow DAGs, Jupyter notebooks, and ML pipelines from development to production with confidence. Built for data engineers, ML engineers, and platform teams managing multi-environment data science workflows.

Why SMUS CI/CD CLI?

✅ **Deploy with Confidence** - Automated testing and validation before production

✅ **Multi-Environment Management** - Dev → Test → Prod with environment-specific configuration

✅ **DataZone Integration** - Automatic catalog asset subscription and approval workflows

✅ **Infrastructure as Code** - Version-controlled pipeline definitions and reproducible deployments

✅ **GitHub Actions Ready** - Native CI/CD integration for automated deployments

---

Quick Start

Install from source:

```
git clone https://github.com/aws/Unified-Studio-for-Amazon-
        Sagemaker.git
cd Unified-Studio-for-Amazon-Sagemaker/experimental/SMUS-CICD-
        pipeline-cli
pip install -e .
```

Deploy your first pipeline:

```
# Validate configuration
smus-cli describe --pipeline pipeline.yaml --connect

# Create deployment bundle
smus-cli bundle --pipeline pipeline.yaml --targets dev

# Deploy to test environment
smus-cli deploy --targets test --pipeline pipeline.yaml

# Run validation tests
smus-cli test --pipeline pipeline.yaml --targets test
```

See it in action: Live GitHub Actions Example

Who Is This For?

👷 DevOps Teams

Build and deploy CI/CD pipelines for data engineering, ML, and GenAI workflows.
→ **Quick Start Guide** - Deploy your first pipeline in 10 minutes

Includes examples for: - Data Engineering (Glue, Notebooks, Athena) - ML Workflows (SageMaker, Notebooks) - GenAI Applications (Bedrock, Notebooks)

🔧 Platform Administrators

Set up and manage multi-environment infrastructure for DevOps teams.
→ **Admin Guide** - Configure infrastructure in 15 minutes

Key Features

🚀 Automated Deployment

- **Bundle Creation** - Package workflows, notebooks, and data assets into deployable artifacts
- **Multi-Target Deployment** - Deploy to dev, test, and prod with a single command
- **Environment Variables** - Dynamic configuration using `${VAR}` substitution
- **Rollback Support** - Revert to previous versions when needed

🔍 Testing & Validation

- **Automated Tests** - Run validation tests before promoting to production
- **Quality Gates** - Block deployments if tests fail
- **Workflow Monitoring** - Track execution status and logs
- **Health Checks** - Verify deployment correctness

📊 DataZone Catalog Integration

- **Asset Discovery** - Automatically find required catalog assets
- **Subscription Management** - Request access to tables and datasets
- **Approval Workflows** - Handle cross-project data access
- **Asset Tracking** - Monitor catalog dependencies

🔄 CI/CD Integration

- **GitHub Actions** - Pre-built workflows for automated deployment
- **GitLab CI** - Native support for GitLab pipelines
- **Environment Variables** - Flexible configuration for any CI/CD platform
- **Webhook Support** - Trigger deployments from external events

🏗️ Infrastructure Management

- **Project Creation** - Automatically provision SageMaker Unified Studio projects
- **Connection Setup** - Configure S3, Airflow, Athena, and Lakehouse connections
- **Resource Mapping** - Link AWS resources to project connections
- **Permission Management** - Control access and collaboration

---

Supported AWS Services

Deploy workflows using these AWS services through Airflow YAML syntax:

🎯 Analytics & Data

Amazon Athena • AWS Glue • Amazon EMR • Amazon Redshift • Lake Formation

🤖 Machine Learning

SageMaker Training • SageMaker Pipelines • Feature Store • Model Registry • Batch Transform

🧠 Generative AI

Amazon Bedrock • Bedrock Agents • Bedrock Knowledge Bases • Guardrails

📊 Additional Services

S3 • Lambda • Step Functions • DynamoDB • RDS • SNS/SQS • Batch

See complete list: Airflow AWS Operators Reference

Core Concepts

Pipeline

A YAML manifest defining your complete CI/CD workflow: - Target environments (dev, test, prod) - Bundle configuration (what to deploy) - Workflow definitions and parameters - Environment-specific settings

Bundle

A deployable package containing: - Airflow DAGs and Python scripts - Jupyter notebooks and data files - ML models and configuration - Git repository dependencies

Target

A deployment environment mapping to a SageMaker Unified Studio project: - Environment configuration (domain, region, project) - Resource definitions (S3, Airflow, Athena, Glue) - Deployment settings and parameters - Access control and permissions

How it works: Bundles code → Deploys to projects → Runs workflows → Monitors execution

---

Documentation

Getting Started

- **Quick Start Guide** - Deploy your first pipeline (10 min)
- **Admin Guide** - Set up infrastructure (15 min)

Guides

- **Pipeline Manifest Reference** - Complete YAML configuration guide
- **CLI Commands** - Detailed command documentation
- **Substitutions & Variables** - Dynamic configuration
- **GitHub Actions Integration** - Automated CI/CD workflows

Reference

- **Pipeline Manifest Schema** - YAML schema reference
- **Airflow AWS Operators** - Custom operators

Examples

- **ETL Pipeline** - Glue jobs with Airflow orchestration
- **ML Pipeline** - SageMaker training with MLflow tracking
- **Serverless Example** - Airflow Serverless workflows
- **MWAA Example** - Managed Airflow workflows

Development

- **Development Guide** - Contributing and testing

Example Pipeline Flow

```
graph LR
    subgraph "Development"
        A[Write Code] --> B[Test Locally]
    end

    subgraph "CI/CD Pipeline"
        B --> C[Bundle]
        C --> D[Deploy to Test]
        D --> E[Run Tests]
        E --> F{Tests Pass?}
        F -->|Yes| G[Deploy to Prod]
        F -->|No| H[Block Deployment]
    end

    subgraph "Production"
        G --> I[Monitor]
    end
```

Example Pipelines

ETL Pipeline (`examples/analytic-workflow/etl/`)

What it deploys: - **2 AWS Glue jobs** running on Glue 4.0 - `data_discovery_task` - Lists and discovers S3 data files - `data_summary_task` - Processes COVID-19 data from Athena tables - **Airflow Serverless workflow** orchestrating job dependencies - **Python scripts** bundled and uploaded to S3 shared storage

Pipeline manifest (`etl_pipeline.yaml`): - Bundles `etl/` directory to S3 connection `default.s3_shared` - Deploys to `dev` (DEV stage) and `test` (TEST stage with auto-created project) - Injects environment variables (`S3_PREFIX`, `AWS_REGION`) - Runs

integration tests from `pipeline_tests/` folder

Workflow (`workflow_combined.yaml`):

```yaml
workflow_combined:
  dag_id: 'covid_etl_workflow'
  tasks:
    data_discovery_task:
      operator:
        airflow.providers.amazon.aws.operators.glue.GlueJobOperator
      script_location: 's3://.../etl/bundle/glue_s3_list_job.py'
      script_args:
        '--BUCKET_NAME': 'amazon-sagemaker-...'

    data_summary_task:
      operator:
        airflow.providers.amazon.aws.operators.glue.GlueJobOperator
      script_location: 's3://.../etl/bundle/glue_covid_summary_job.py'
      script_args:
        '--DATABASE_NAME': 'covid19_db'
        '--TABLE_NAME': 'us_simplified'
```

Deploy:

```
cd examples/analytic-workflow/etl
smus-cli bundle --pipeline etl_pipeline.yaml --targets dev
smus-cli deploy --targets test --pipeline etl_pipeline.yaml
```

ML Training Pipeline (`examples/analytic-workflow/ml/`)

What it deploys: - **SageMaker Notebook Operator** executing ML orchestration notebook - **Training code** bundled with compression to S3 (`job-code/` directory) - **Workflow definition** with MLflow connection injection - **MLflow tracking server** connection for experiment tracking

Pipeline manifest (`ml_pipeline.yaml`): - Bundles 2 storage locations: - `training-code` → compressed tarball with training script + inference code - `ml-workflows` → notebook and workflow definitions - Auto-creates test project with MLflow connection - Injects MLflow tracking server ARN via connection parameter substitution

Workflow (`ml_dev_workflow_v3.yaml`):

```yaml
ml_dev_workflow:
  dag_id: "ml_dev_workflow_v3"
  tasks:
    ml_orchestrator_notebook:
      operator:
        airflow.providers.amazon.aws.operators.sagemaker_unified_studio.SageMakerNotebook
      input_config:
        input_path:
        "ml/bundle/workflows/ml_orchestrator_notebook.ipynb"
        input_params:
          mlflow_tracking_server_arn: "{proj.connection.mlflow-
        server.trackingServerArn}"
```

ML Orchestrator Notebook does: 1. Generates synthetic training data (1000 samples, 20 features, 3 classes) 2. Uploads training/inference data to S3 3. Launches SageMaker training job with SKLearn estimator 4. Logs metrics and model to MLflow tracking server 5. Runs batch transform inference on test data 6. Outputs predictions to S3

Training script (`sagemaker_training_script.py`): - Trains RandomForest classifier on synthetic data - Logs hyperparameters and metrics to MLflow - Saves model artifacts (model.joblib, scaler.joblib) - **Copies inference.py to model tarball** for batch transform

Deploy:

```
cd examples/analytic-workflow/ml
smus-cli bundle --pipeline ml_pipeline.yaml --targets dev
smus-cli deploy --targets test --pipeline ml_pipeline.yaml
```

Key features: - ✅ Dynamic parameter injection from project connections - ✅ MLflow experiment tracking and model registry integration - ✅ SageMaker training with custom dependencies (requirements.txt) - ✅ Batch transform inference with custom inference handler - ✅ Compressed bundle storage for efficient deployment

---

Common Use Cases

Deploy Airflow DAGs

```
# Bundle and deploy workflows (YAML syntax) to test environment
smus-cli bundle --targets dev
smus-cli deploy --targets test
smus-cli run --targets test --workflow my_dag
```

Promote to Production

```
# Run tests in staging
smus-cli test --targets test

# Deploy to production if tests pass
smus-cli deploy --targets prod
```

Manage Catalog Assets

```
# Request access to catalog tables
smus-cli deploy --targets test  # Automatically requests subscriptions
smus-cli monitor --targets test  # Track approval status
```

CI/CD Automation

```
# .github/workflows/deploy.yml
- name: Deploy to Test
  run: smus-cli deploy --targets test --pipeline pipeline.yaml

- name: Run Tests
  run: smus-cli test --targets test --pipeline pipeline.yaml

- name: Deploy to Prod
  if: success()
  run: smus-cli deploy --targets prod --pipeline pipeline.yaml
```

Security Notice

⚠️ **DO NOT** install from PyPI - always install from official AWS source code.

```
# ✅ Correct - Install from official AWS repository
git clone https://github.com/aws/Unified-Studio-for-Amazon-
        Sagemaker.git
cd Unified-Studio-for-Amazon-Sagemaker/experimental/SMUS-CICD-
        pipeline-cli
pip install -e .

# ❌ Wrong - Do not use PyPI
pip install smus-cicd-cli  # May contain malicious code
```

Support & Community

- **Documentation**: docs/
- **Examples**: examples/
- **Issues**: GitHub Issues
- **Contributing**: Development Guide

Quick Navigation - All READMEs

📚 Documentation

- **Getting Started Hub** - Role-based navigation for DevOps teams and admins
- **Quick Start Guide** - Deploy your first pipeline in 10 minutes
- **Admin Quick Start** - Infrastructure setup in 15 minutes

📖 Reference Guides

- **Pipeline Manifest** - Complete YAML configuration reference
- **CLI Commands** - All available commands and options
- **Substitutions & Variables** - Dynamic configuration guide
- **GitHub Actions Integration** - CI/CD automation setup
- **Airflow AWS Operators** - Custom operator reference
- **Pipeline Manifest Schema** - YAML schema validation

💡 Examples

- **Examples Overview** - All available examples and usage
- **ETL Pipeline** - Glue jobs with Airflow orchestration
- **ML Pipeline** - SageMaker training with MLflow tracking
- **ML Training Code** - Training script details
- **Serverless Example** - Airflow Serverless workflows
- **MWAA Example** - Managed Airflow workflows

🧪 Testing & Development

- **Tests Overview** - Testing infrastructure and guidelines
- **Test Scripts** - Helper scripts for testing
- **Development Guide** - Contributing and local development

License

CLI Commands Reference

← Back to Main README

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

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 --bundle bundle.zip`
`run`	Execute workflow commands or trigger workflows	`smus-cli run --workflow my_dag`
`logs`	Fetch workflow logs from CloudWatch	`smus-cli logs --workflow arn:aws:airflow-serverless:region:account:workflow/name`
`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`

Detailed Command Examples

1. Describe Pipeline Configuration

```
# Basic describe
smus-cli describe --pipeline pipeline.yaml

# Describe with connection details and AWS connectivity
smus-cli describe --pipeline pipeline.yaml --connect
```

Example Output:

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

Targets:
  - dev: dev-marketing
    Project Name: dev-marketing
```

```
    Project ID: <dev-project-id>
    Status: ACTIVE
    Owners: Admin, eng1
    Connections:
      project.workflow_mwaa:
        connectionId: 6f58emph2gtciv
        type: WORKFLOWS_MWAA
        region: us-east-1
        awsAccountId: <aws-account-id>
        description: Connection for MWAA environment
        environmentName: DataZoneMWAAEnv-<domain-id>-<project-id>-dev
      project.workflow_serverless:
        connectionId: 7g69fnqi3hukjw
        type: WORKFLOWS_SERVERLESS
        region: us-east-1
        awsAccountId: <aws-account-id>
        description: Serverless workflows connection
      default.s3_shared:
        connectionId: dqbxjn28zehzjb
        type: S3
        region: us-east-1
        awsAccountId: <aws-account-id>
        description: This is the connection to interact with s3 shared
storage location if enabled in the project.
        s3Uri: s3://sagemaker-unified-studio-<aws-account-id>-us-east-
1-your-domain-name/<domain-id>/<dev-project-id>/shared/
        status: READY

Manifest Workflows:
  - test_dag
    Connection: project.workflow_mwaa
    Engine: MWAA
  - execute_notebooks_dag
    Connection: project.workflow_mwaa
    Engine: MWAA
```

2. Bundle Creation

```
# Bundle for specific target
smus-cli bundle --targets dev --output-dir ./bundles

# Bundle for multiple targets
smus-cli bundle --targets dev,test --output-dir /tmp/bundles
```

3. Deploy Bundle

```
# Deploy using auto-created bundle
smus-cli deploy --targets test

# Deploy using pre-created bundle file
smus-cli deploy --targets test --bundle /path/to/bundle.zip

# Deploy with JSON output
smus-cli deploy --targets test --bundle bundle.zip --output JSON
```

4. Run Commands and Workflows

Execute Airflow CLI Commands (MWAA)

```
# Get Airflow version
smus-cli run --workflow test_dag --command version

# List all DAGs
smus-cli run --workflow sample_dag --command "dags list"

# Get DAG state
smus-cli run --workflow sample_dag --command "dags state sample_dag"
```

Trigger Workflows

```
# Trigger single workflow (works with both MWAA and serverless
        Airflow)
smus-cli run --workflow test_dag

# Trigger workflow on specific target
smus-cli run --workflow test_dag --targets prod

# Trigger with JSON output
smus-cli run --workflow test_dag --output JSON
```

Example Output (TEXT format - MWAA):

```
🔍 Checking MWAA health for target 'test' (project: integration-test-
test)
🎯 Target: test
🚀 Triggering workflow: test_dag
🔧 Connection: project.workflow_mwaa (DataZoneMWAAEnv-
dzd_6je2k8b63qse07-broygppc8vw17r-dev)
📋 Command: dags trigger test_dag
✅ Command executed successfully
📤 Output:
2.10.1
```

Example Output (TEXT format - Serverless Airflow):

```
🎯 Target: test (Serverless Airflow)
🚀 Starting workflow run: MyPipeline_test_test_dag
🔗 ARN: arn:aws:airflow-serverless:us-east-
2:123456789012:workflow/MyPipeline_test_test_dag
✅ Workflow run started successfully
📋 Run ID: manual__2025-10-15T15:45:00+00:00
📊 Status: STARTING
```

Example Output (JSON format):

```json
{
  "workflows": ["test_dag"],
  "command": "dags trigger test_dag",
  "results": [
    {
      "target": "test",
      "connection": "project.workflow_mwaa",
      "environment": "DataZoneMWAAEnv-dzd_6je2k8b63qse07-
```

```
broygppc8vw17r-dev",
      "success": true,
      "status_code": 200,
      "command": "dags trigger test_dag",
      "raw_stdout": "...",
      "raw_stderr": "..."
    }
  ],
  "success": true
}
```

5. Fetch Workflow Logs

```
# Fetch logs for serverless Airflow workflow
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyPipeline_test_test_dag

# Fetch logs with live monitoring
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyPipeline_test_test_dag --live

# Fetch specific number of log lines
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyPipeline_test_test_dag --lines 50

# Fetch logs with JSON output
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyPipeline_test_test_dag --output JSON
```

Example Output:

```
📋 Fetching logs for workflow: MyPipeline_test_test_dag
🔗 ARN: arn:aws:airflow-serverless:us-east-
2:123456789012:workflow/MyPipeline_test_test_dag
🔄 Live monitoring enabled - Press Ctrl+C to stop
================================================================================
📁 Log Group: /aws/mwaa-serverless/MyPipeline_test_test_dag/
📊 Workflow Status: ACTIVE
--------------------------------------------------------------------------------
----------
📄 Showing 15 log events:

[15:45:23] [scheduler] Starting workflow execution
[15:45:24] [task-runner] Initializing S3ListOperator task
[15:45:25] [task-runner] Task completed successfully
```

6. Monitor Workflows

```
# Monitor all targets
smus-cli monitor --pipeline pipeline.yaml

# Monitor specific targets with JSON output
smus-cli monitor --targets test --output JSON
```

6. Test Pipeline

```
# Run tests for all targets
smus-cli test --pipeline pipeline.yaml
```

```
# Run tests for specific targets with verbose output
smus-cli test --targets test --verbose

# Stream test output directly to console
smus-cli test --targets test --test-output console
```

8. Delete Resources

```
# Delete with confirmation
smus-cli delete --targets test

# Force delete without confirmation
smus-cli delete --targets test --force

# Async delete (don't wait for completion)
smus-cli delete --targets test --force --async
```

Universal Options

All commands support these universal options:

Option	Short	Description	Example
`--pipeline`	`-p`	Path to pipeline manifest file	`--pipeline my-pipeline.yaml`
`--targets`	`-t`	Target environment(s)	`--targets dev,test`
`--output`	`-o`	Output format (TEXT/JSON)	`--output JSON`

Output Formats

TEXT Format (Default)

- Human-readable output with emojis and formatting
- Raw stdout/stderr for run commands
- Suitable for interactive use

JSON Format

- Structured data output
- Suitable for automation and scripting
- All commands support JSON output via `--output JSON`

Error Handling

The CLI provides comprehensive error handling: - **Exit Code 0**: Success - **Exit Code 1**: Error occurred - **Graceful Failures**: Commands handle missing infrastructure gracefully - **Detailed Error Messages**: Clear indication of what went wrong and how to fix it

MWAA Integration

The CLI automatically validates MWAA environment health before executing workflow commands: - ✅ **MWAA Available**: Commands execute successfully - ❌ **MWAA Unavailable**: Commands fail with clear error message - 🔍 **Auto-Detection**: CLI

automatically finds and validates MWAA connections workgroup: workgroup--xyz123 project.spark.compatibility: connectionId: 6236xbz8cowo4n type: SPARK region: us-east-1 awsAccountId: description: Glue-ETL compute with Permission Mode set to compatibility. (Auto-created by project). glueVersion: 5.0 workerType: G.1X numberOfWorkers: 10 project.workflow_mwaa: connectionId: d5jq3vs4ol9s13 type: WORKFLOWS_MWAA region: us-east-1 awsAccountId: description: Connection for MWAA environment environmentName: SageMaker Unified StudioMWAAEnv---dev project.workflow_serverless: connectionId: e6kr4wt5pm0t24 type: WORKFLOWS_SERVERLESS region: us-east-1 awsAccountId: description: Serverless workflows connection

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.

2. Create Bundle from Dev Environment

```bash
smus-cli bundle --pipeline pipeline.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
✅ Bundle created: s3://my-datazone-
bucket/bundles/IntegrationTestMultiTarget.zip (279462 bytes)

📦 Bundle Contents:
==================================================
├── storage/
│   └── src/
│       └── test-notebook1.ipynb
└── workflows/
    ├── .visual/
    │   └── runGettingStartedNotebook.wf
    ├── dags/
    │   ├── test_dag.py
    │   └── visual/
    │       └── runGettingStartedNotebook.py
    └── config/
        ├── requirements.txt
        └── startup.sh
==================================================
📊 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. When using S3 bundle storage (configured via `bundlesDirectory: s3://bucket/path`), the bundle is automatically uploaded to S3 after creation. This creates a centralized bundle that can be accessed by team members and CI/CD systems from anywhere.

3. Deploy to Test Environment

```
smus-cli deploy --targets test --pipeline pipeline.yaml
```

Example Output:

```
Deploying to target: test
Project: integration-test-test
Domain: cicd-test-domain
Region: us-east-1
🔧 Auto-initializing target infrastructure...
✅ Target infrastructure ready
✅ Project 'integration-test-test' exists
Bundle file: s3://my-datazone-
bucket/bundles/IntegrationTestMultiTarget.zip
Downloading bundle from S3...
Deploying storage to: default.s3_shared/src (append: False)
  S3 Location: s3://sagemaker-unified-studio-<aws-account-id>-us-east-
1-your-domain-name/.../shared/src/
    Synced: test-notebook1.ipynb
  Storage files synced: 1
Deploying workflows to: default.s3_shared/workflows (append: True)
  S3 Location: s3://sagemaker-unified-studio-<aws-account-id>-us-east-
1-your-domain-name/.../shared/workflows/
    Synced: test_dag.py
    Synced: runGettingStartedNotebook.py
  Workflow files synced: 17
✅ Deployment complete! Total files synced: 18

📦 Processing 1 catalog assets...

--- Asset 1/1 ---
🔍 Processing asset access for: covid19_db.countries_aggregated
✅ Found asset: 3ljuj2gtiziwx3, listing: 3r1ch3l4y6dx9j
✅ Using existing subscription
⏳ Waiting for grants to be created... (60s remaining)
📊 Grant bs1gp0rd7ud7l3 status: COMPLETED
✅ Asset access successfully configured!

✅ Successfully processed 1/1 catalog assets

🚀 Starting workflow validation...
✅ MWAA environment is available
🆕 New DAGs detected: runGettingStartedNotebook
```

What this shows: The deploy command downloads the bundle from S3 (if using S3 bundle storage) and uploads the files to the target environment's SageMaker Unified Studio project storage and workflow connections. It also processes catalog assets defined in the pipeline manifest, requesting access to required data tables and waiting for subscription approval. The deployment shows progress for file uploads, catalog asset

access, and validates that the MWAA environment can access the new workflows. This ensures your code changes and data access are properly configured and ready for execution.

4. Monitor Workflow Status

```
smus-cli monitor --pipeline pipeline.yaml
```

Example Output:

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

🔍 Monitoring Status:

🎯 Target: test
   Project: integration-test-test
   Project ID: <test-project-id>
   Status: ACTIVE
   Owners: Admin, eng1

   📊 Workflow Status:
      🔧 project.workflow_mwaa (SageMaker Unified StudioMWAAEnv-
<domain-id>-<test-project-id>-dev)
         🌐 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.

5. Trigger Workflow Execution

```
smus-cli run --pipeline pipeline.yaml --targets test --workflow
        test_dag --command trigger
```

Example Output:

```
🎯 Target: test
🔧 Connection: project.workflow_mwaa (SageMaker Unified StudioMWAAEnv-
<domain-id>-<test-project-id>-dev)
📋 Command: trigger
✅ Workflow triggered successfully
📤 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. Run Tests

```
smus-cli test --pipeline pipeline.yaml --targets marketing-test-stage
```

Example Output:

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

🎯 Target: test
  📁 Test folder: tests/
  🔧 Project: integration-test-test (your-project-id)
  🧪 Running tests...
  ✅ Tests passed

🎯 Test Summary:
  ✅ Passed: 1
  ❌ Failed: 0
  ⚠️  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.

8. Clean Up Resources

```
smus-cli delete --targets test --pipeline pipeline.yaml --force
```

Example Output:

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

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

🗑️  Deleting target: test
✅ Successfully deleted project: integration-test-test

🎯 Deletion Summary
  ✅ 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
```

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 workflow commands
6. **`logs`** - Fetch workflow logs from CloudWatch
7. **`delete`** - Delete projects and environments

Command Details

0. create - Create New Pipeline Manifest

Creates a new pipeline manifest file with basic structure.

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

Options

- **`-o, --output`**: Output file path for the pipeline manifest (default: `pipeline.yaml`)
- **`-n, --name`**: Pipeline name (optional, defaults to 'YourPipelineName')
- **`--domain-id`**: SageMaker Unified Studio domain ID (optional)
- **`--dev-project-id`**: Development project ID to base other targets on (optional)
- **`--stages`**: Comma-separated list of stages to create targets for (default: `dev,test,prod`)
- **`--region`**: AWS region (default: `us-east-1`)
- **`--help`**: Show command help

Examples

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

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

# Create with specific stages and region
smus-cli create --output pipeline.yaml --stages dev,test,prod --region
        us-west-2
```

1. describe - Describe Pipeline Configuration

Validates and displays information about your pipeline manifest.

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

Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipeline.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

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
```

2. bundle - Create Deployment Packages

Creates bundle zip files by downloading from S3.

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

Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipeline.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

Positional Arguments

- **`TARGET_POSITIONAL`**: Target name (positional argument for backward compatibility)

Bundle Storage Locations

The bundle command supports both local and S3 storage locations via the `bundlesDirectory` configuration in your pipeline manifest:

Local Storage:

```
bundlesDirectory: ./bundles
```

- Bundles are created directly in the specified local directory
- Suitable for development and single-user workflows

S3 Storage:

```
bundlesDirectory: s3://my-datazone-bucket/bundles
```

- Bundles are created locally then uploaded to S3
- Enables team collaboration and CI/CD integration
- Works with DataZone domain S3 buckets
- Requires appropriate S3 permissions

Examples

```
# Bundle default target
smus-cli bundle

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

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

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

```
# Bundle using positional argument (backward compatibility)
smus-cli bundle dev
```

3. deploy - Deploy to Targets

Deploys bundle files to target environments (auto-initializes if needed). The deploy command performs the following operations:

1. **Bundle Deployment**: Uploads workflow and storage files to target project connections
2. **Catalog Asset Access**: Processes catalog assets defined in the pipeline manifest:
   - Searches for assets in the DataZone catalog
   - Creates subscription requests for required access
   - Waits for subscription approval (up to 5 minutes)
   - Verifies subscription grants are completed
   - Fails deployment if catalog access cannot be obtained
3. **Workflow Validation**: Ensures deployed workflows are accessible by the target environment

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

Options

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

Positional Arguments

- **`TARGET_POSITIONAL`**: Target name (positional argument for backward compatibility)

Examples

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

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

# Deploy with pre-created bundle
smus-cli deploy --targets test --bundle /path/to/bundle.zip

# Deploy using positional argument (backward compatibility)
smus-cli deploy test
```

4. monitor - Monitor Workflow Status

Monitors workflow status across target environments.

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

Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipeline.yaml`)
- **`-t, --targets`**: Target name(s) - single target or comma-separated list (shows all targets if not specified)
- **`-l, --live`**: Keep monitoring until all workflows complete
- **`-o, --output`**: Output format: TEXT (default) or JSON
- **`--help`**: Show command help

Examples

```
# Monitor all targets (one-time snapshot)
smus-cli monitor

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

# Live monitoring - continuously poll until workflows complete
smus-cli monitor --live

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

Live Monitoring

When using `--live`, the monitor command: 1. Displays initial table with all workflow statuses 2. Polls every 10 seconds for status changes 3. Reports status changes as new lines: `[HH:MM:SS] workflow_name (run id): OLD_STATUS → NEW_STATUS` 4. Exits automatically when no workflows are RUNNING or QUEUED 5. Can be stopped manually with Ctrl+C

Example Live Output:

```
🔄 Starting live monitoring... (Press Ctrl+C to stop)

Pipeline: IntegrationTestMLWorkflow

      Workflow                                  Status     Trigger
Run ID       Run Status   Start Time           Duration
      ----------------------------------------- ---------- ------------
------------ ------------ -------------------- ----------
      IntegrationTestMLWorkflow_test_market... READY      scheduled
ZG9hFOTB...  RUNNING      2025-11-02 22:23:24  2m

[22:25:34] IntegrationTestMLWorkflow_test_marketing_ml_dev_workflow_v3
(run ZG9hFOTB...): RUNNING → SUCCEEDED

✅ All workflows completed
```

6. logs - Fetch Workflow Logs

Fetches and displays workflow logs from CloudWatch (supports serverless Airflow workflows).

```
smus-cli logs [OPTIONS]
```

Options

- **`-w, --workflow`**: Workflow ARN to fetch logs for (required)
- **`-l, --live`**: Keep fetching logs until workflow terminates
- **`-o, --output`**: Output format: TEXT (default) or JSON
- **`-n, --lines`**: Number of log lines to fetch (default: 100)
- **`--help`**: Show command help

Examples

```
# Fetch logs for serverless Airflow workflow
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow
```

```
# Live log monitoring (streams logs in real-time)
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow --live

# Fetch specific number of lines
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow --lines 50

# Fetch logs with JSON output
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow --output JSON
```

Example Output:

```
📋 Fetching logs for workflow:
IntegrationTestMLWorkflow_test_marketing_ml_dev_workflow_v3
🔗 ARN: arn:aws:airflow-serverless:us-east-
1:198737698272:workflow/IntegrationTestMLWorkflow_test_marketing_ml_dev
A3zE3YBMKo
================================================================================
📁 Log Group: /aws/mwaa-
serverless/IntegrationTestMLWorkflow_test_marketing_ml_dev_workflow_v3-
A3zE3YBMKo/
📊 Workflow Status: READY
--------------------------------------------------------------------------------
📄 Showing 100 log events:

[2025-11-02 15:52:34]
[workflow_id=IntegrationTestMLWorkflow.../task_id=ml_orchestrator_notebo
{"timestamp":"2025-11-
02T20:52:34.124324Z","level":"info","event":"Executing workload"...}
[2025-11-02 15:52:35]
[workflow_id=IntegrationTestMLWorkflow.../task_id=ml_orchestrator_notebo
{"timestamp":"2025-11-02T20:52:35.035022","level":"info","event":"DAG
bundles loaded: dags-folder"...}
```

5. run - Run Workflow Commands

Executes workflow commands on target environments (supports both MWAA and serverless Airflow).

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

Options

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

Examples

```
# Trigger workflow (works with both MWAA and serverless Airflow)
smus-cli run --workflow my_dag

# Run Airflow CLI command (MWAA only)
```

```
smus-cli run --workflow my_dag --command version

# Run on specific target with JSON output
smus-cli run --workflow my_dag --targets prod --output JSON
```

6. logs - Fetch Workflow Logs

Fetches and displays workflow logs from CloudWatch (supports serverless Airflow workflows).

```
smus-cli logs [OPTIONS]
```

Options

- **`-w, --workflow`**: Workflow ARN to fetch logs for (required)
- **`-l, --live`**: Keep fetching logs until workflow terminates
- **`-o, --output`**: Output format: TEXT (default) or JSON
- **`-n, --lines`**: Number of log lines to fetch (default: 100)
- **`--help`**: Show command help

Examples

```
# Fetch logs for serverless Airflow workflow
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow

# Live log monitoring
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow --live

# Fetch specific number of lines with JSON output
smus-cli logs --workflow arn:aws:airflow-serverless:us-east-
        2:123456789012:workflow/MyWorkflow --lines 50 --output JSON
```

8. delete - Delete Target Environments

Deletes DataZone projects and associated resources for specified targets.

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

Options

- **`-p, --pipeline`**: Path to pipeline manifest file (default: `pipeline.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

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
```

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

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

Global Options

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

Exit Codes

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

Configuration Files

Pipeline Manifest

- Default location: `pipeline.yaml` (current directory)
- Override with `--pipeline` option
- See Pipeline Manifest Reference for format
- **Error handling**: CLI will error if the default file doesn't exist and no alternative is specified

AWS Configuration

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

Common Workflows

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
```

Cleanup Workflow

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

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

Pipeline Manifest Reference

← Back to Main README

The pipeline manifest is a YAML file that defines your CI/CD pipeline configuration for Amazon SageMaker Unified Studio, including bundle content, target environments, workflows, and deployment settings.

Quick Links

- **Pipeline Manifest Schema Documentation** - Complete schema reference with validation rules
- **Substitutions and Variables** - Dynamic variable resolution in workflows
- **CLI Commands Reference** - Detailed command documentation

Minimal Example

For simple use cases, the manifest can be very straightforward:

```yaml
pipelineName: SimpleDataPipeline

# Bundle what to deploy
bundle:
  bundlesDirectory: ./bundles
  storage:
    - name: code
      connectionName: default.s3_shared
      include: ['src/']

# Where to deploy
targets:
  test:
    stage: TEST
    domain:
      name: my-studio-domain
      region: us-east-1
    project:
      name: test-project
```

```
    bundle_target_configuration:
      storage:
        - name: code
          connectionName: default.s3_shared
          targetDirectory: 'src'

# Workflows to create after deployment
workflows:
  - workflowName: my_dag
    engine: airflow-serverless
```

This minimal example: - Bundles source files from the `src/` directory with name “code” - Deploys to a single `test` target - Creates a serverless Airflow workflow after deployment - Uses an existing project (no initialization needed) - No connections creation, no catalog assets, no tests - Perfect for serverless Airflow pipelines

Note: For workflows (DAGs), add them as storage items with `append: true`. The `workflows` section at the root tells the CLI which workflows to create after deployment.

Comprehensive Example

This example demonstrates most features of the pipeline manifest:

```
pipelineName: MarketingDataPipeline

# Bundle storage location (local or S3)
bundlesDirectory: s3://sagemaker-unified-studio-123456789012-us-east-
        1-domain/bundles

# Bundle configuration - what to include in deployments
bundle:
  # Storage files (unified - includes workflows and source code)
  storage:
    - name: code
      connectionName: default.s3_shared
      append: false
      include: ['src/', 'data/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc',
        '.DS_Store']

    - name: workflows
      connectionName: default.s3_shared
      append: true
      include: ['workflows/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc']

  # Git repositories
  git:
    - repository: MarketingDataPipeline
      url: https://github.com/myorg/marketing-pipeline.git

  # DataZone catalog assets
  catalog:
    assets:
      - selector:
          search:
            assetType: GlueTable
            identifier: covid19_db.countries_aggregated
        permission: READ
        requestReason: Required for marketing analytics pipeline
```

```yaml
# Target environments
targets:
  dev:
    stage: DEV
    domain:
      name: ${DOMAIN_NAME:my-studio-domain}
      region: ${AWS_REGION:us-east-1}
    project:
      name: dev-marketing

    # Target-specific bundle destinations
    bundle_target_configuration:
      storage:
        - name: code
          connectionName: default.s3_shared
          targetDirectory: 'src'
        - name: workflows
          connectionName: default.s3_shared
          targetDirectory: 'workflows'

    # Target-specific workflow parameters
    environment_variables:
      S3_PREFIX: "dev"
      DEBUG_MODE: true
      MAX_RETRIES: 3

    # Integration tests
    tests:
      folder: tests/integration/

  test:
    stage: TEST
    domain:
      name: ${DOMAIN_NAME:my-studio-domain}
      region: ${AWS_REGION:us-east-1}
    project:
      name: test-marketing

    # Auto-create project and resources
    initialization:
      project:
        create: true
        profileName: 'All capabilities'
        owners: ['alice@company.com']
        contributors: ['bob@company.com']
        role:
          arn: arn:aws:iam::123456789012:role/MyProjectRole
        userParameters:
          - EnvironmentConfigurationName: 'Lakehouse Database'
            parameters:
              - name: glueDbName
                value: test_marketing_db

      environments:
        - EnvironmentConfigurationName: 'OnDemand Workflows'

      # Create connections
      connections:
        - name: s3-data-lake
          type: S3
          properties:
```

```yaml
            s3Uri: "s3://test-data-bucket/data/"

        - name: athena-analytics
          type: ATHENA
          properties:
            workgroupName: "test-workgroup"

        - name: spark-processing
          type: SPARK_GLUE
          properties:
            glueVersion: "4.0"
            workerType: "G.1X"
            numberOfWorkers: 5

        - name: mwaa-workflows
          type: WORKFLOWS_MWAA
          properties:
            mwaaEnvironmentName: "test-airflow-env"

        - name: serverless-workflows
          type: WORKFLOWS_SERVERLESS
          properties: {}

    bundle_target_configuration:
      storage:
        - name: code
          connectionName: default.s3_shared
          targetDirectory: 'src'
        - name: workflows
          connectionName: default.s3_shared
          targetDirectory: 'workflows'

    environment_variables:
      S3_PREFIX: "test"
      DEBUG_MODE: false
      MAX_RETRIES: 5

    tests:
      folder: tests/integration/

  prod:
    stage: PROD
    domain:
      name: ${DOMAIN_NAME:my-studio-domain}
      region: ${AWS_REGION:us-east-1}
    project:
      name: prod-marketing

    initialization:
      project:
        create: true
        profileName: 'All capabilities'
        owners: ['alice@company.com']
        contributors: []
      environments:
        - EnvironmentConfigurationName: 'OnDemand Workflows'

    bundle_target_configuration:
      storage:
        - name: code
          connectionName: default.s3_shared
```

```
          targetDirectory: 'src'
        - name: workflows
          connectionName: default.s3_shared
          targetDirectory: 'workflows'
      catalog:
        disable: true  # Disable catalog processing for prod

    environment_variables:
      S3_PREFIX: "prod"
      DEBUG_MODE: false
      MAX_RETRIES: 10

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

Bundle Section

The `bundle` section defines what content to package and deploy to target environments.

Bundle Storage Location

```
bundlesDirectory: ./bundles  # Local directory
# OR
bundlesDirectory: s3://my-bucket/bundles  # S3 location
```

- **Local Storage**: Use for development and testing (`./bundles`, `~/bundles`)
- **S3 Storage**: Use for production and CI/CD systems
  - Centralized storage across teams
  - Version history with S3 versioning
  - Cross-region access
  - Integration with DataZone domain S3 buckets

Storage Bundles

Package source code, libraries, data files, and workflows (unified configuration):

```
bundle:
  storage:
    - name: code
      connectionName: default.s3_shared
      append: false
      include: ['src/', 'lib/', 'data/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc',
        '.DS_Store']

    - name: workflows
      connectionName: default.s3_shared
      append: true
      include: ['workflows/', 'dags/']
      exclude: ['.ipynb_checkpoints/', '__pycache__/', '*.pyc']
```

Properties: - `name` (required): Unique identifier for this bundle item - `connectionName` (required): S3 connection name in source project - `append` (optional): Append to existing files (default: `false`) - `include` (optional): Paths/patterns to include - `exclude` (optional): Paths/patterns to exclude

Best Practices: - Use `append: true` for workflows (incremental updates) - Use `append: false` for source code (clean deployments) - Always exclude temporary files: `.ipynb_checkpoints/`, `__pycache__/`, `*.pyc` - Use descriptive names: `code`, `workflows`, `data`, `models`

Note: Workflows are now part of the unified storage configuration. Use `append: true` for workflow items to enable incremental updates.

Git Repositories

Include Git repositories in bundles:

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

Properties: - `repository` (required): Repository name (used in bundle path: `repositories/{repository-name}/`) - `url` (required): Git repository URL

Note: Git repositories are always cloned to `repositories/{repository-name}/` in the bundle. No `targetDir` configuration needed.

Catalog Assets

Request access to DataZone catalog assets:

```yaml
bundle:
  catalog:
    assets:
      - selector:
          search:
            assetType: GlueTable
            identifier: covid19_db.countries_aggregated
        permission: READ
        requestReason: Required for analytics pipeline

      - selector:
          search:
            assetType: GlueTable
            identifier: sales_db.customer_data
        permission: READ
        requestReason: Customer analytics
```

Properties: - `assets` (required): List of catalog assets - `selector.search.assetType` (required): Asset type (currently `GlueTable`) - `selector.search.identifier` (required): Asset identifier (`database.table`) - `permission` (required): Access level (`READ`, `WRITE`) - `requestReason` (required): Business justification

Deployment Process: 1. Search for assets in DataZone catalog 2. Create subscription requests if needed 3. Wait for approval (up to 5 minutes) 4. Verify subscription grants 5. Fail deployment if access cannot be obtained

---

Target Section

Targets represent deployment environments (dev, test, prod). Each target defines domain, project, and environment-specific configurations.

Basic Target Configuration

```yaml
targets:
  dev:
    stage: DEV
    domain:
      name: my-studio-domain
      region: us-east-1
    project:
      name: dev-marketing
```

Required Properties: - `stage` (required): Deployment stage name (`DEV`, `TEST`, `PROD`) - `domain.name` (required): SageMaker Unified Studio domain name - `domain.region` (required): AWS region where domain exists - `project.name` (required): Project name in the domain

Target Initialization

Auto-create projects, environments, and connections:

```yaml
targets:
  test:
    stage: TEST
    domain:
      name: my-studio-domain
      region: us-east-1
    project:
      name: test-marketing

    initialization:
      project:
        create: true
        profileName: 'All capabilities'
        owners: ['alice@company.com']
        contributors: ['bob@company.com', 'charlie@company.com']
        userParameters:
          - EnvironmentConfigurationName: 'Lakehouse Database'
            parameters:
              - name: glueDbName
                value: my_unique_db_name

      environments:
        - EnvironmentConfigurationName: 'OnDemand Workflows'
        - EnvironmentConfigurationName: 'Lakehouse Database'

      connections:
        - name: s3-data-lake
          type: S3
          properties:
            s3Uri: "s3://my-data-bucket/data/"
```

Project Properties: - `create` (optional): Auto-create project (default: `false`) - `profileName` (required if create=true): Project profile name - `owners` (optional): List of owner email addresses or IAM ARNs - Use `*` as wildcard for account ID: `arn:aws:iam::*:role/MyRole` (replaced with current account) - `contributors`

(optional): List of contributor email addresses or IAM ARNs - Use * as wildcard for account ID: `arn:aws:iam::*:role/MyRole` (replaced with current account) - `role` (optional): Customer-provided IAM role for the project - `arn`: IAM role ARN (e.g., `arn:aws:iam::123456789012:role/MyProjectRole`) - Use * as wildcard for account ID: `arn:aws:iam::*:role/MyProjectRole` (replaced with current account) - The role must have a trust policy allowing DataZone and Airflow Serverless service principals - `userParameters` (optional): Override project profile parameters - `EnvironmentConfigurationName`: Environment configuration to override - `parameters`: Array of name/value pairs

Environments: - List of environment configurations to create - `EnvironmentConfigurationName`: Name of environment configuration

Connections: - See Connections section for all supported types

Bundle Target Configuration

Specify where bundles are deployed in each target using name-based matching:

```yaml
targets:
  test:
    stage: TEST
    domain:
      name: my-studio-domain
      region: us-east-1
    project:
      name: test-marketing

    bundle_target_configuration:
      storage:
        - name: code                    # Matches
        bundle.storage[name=code]
          connectionName: default.s3_shared
          targetDirectory: 'src'
        - name: workflows               # Matches
        bundle.storage[name=workflows]
          connectionName: default.s3_shared
          targetDirectory: 'workflows'
      git:
        - connectionName: default.s3_shared
          targetDirectory: 'repos'      # All git repos deploy here
      catalog:
        disable: false
```

Properties: - `storage` (list): Storage deployment configuration - `name` (required): Name matching bundle storage item - `connectionName` (required): Target S3 connection - `targetDirectory` (required): Target directory path (use `.` or `''` for root) - `git` (list): Git deployment configuration - `connectionName` (required): Target S3 connection - `targetDirectory` (required): Target directory path - `catalog.disable` (optional): Disable catalog asset processing (default: `false`)

Note: Use `targetDirectory: '.'` or `targetDirectory: ''` to deploy to the connection root without a subdirectory.

Target Tests

Run integration tests after deployment:

```yaml
targets:
  test:
    stage: TEST
    domain:
```

```
    name: my-studio-domain
    region: us-east-1
  project:
    name: test-marketing

  tests:
    folder: tests/integration/
```

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

Environment Variables Available to Tests: - `SMUS_DOMAIN_ID`: 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

---

Workflow Section

Workflows define DAGs or pipelines to trigger after deployment.

Global Workflows

Apply to all targets unless overridden:

```
workflows:
  - workflowName: marketing_etl_dag
    connectionName: project.workflow_mwaa
    engine: MWAA
    triggerPostDeployment: true
    logging: console
    parameters:
      data_source: s3://marketing-data/
      output_bucket: s3://marketing-results/
```

Properties: - `workflowName` (required): Workflow/DAG name in the workflow engine - `connectionName` (required): Workflow connection name (required for MWAA, optional for airflow-serverless) - `engine` (optional): Workflow engine type (`MWAA`, `airflow-serverless`) (default: `MWAA`) - `triggerPostDeployment` (optional): Trigger after deployment (default: `false`) - `logging` (optional): Logging level (`console`, `file`, `none`) (default: `console`) - `parameters` (optional): Global workflow parameters

---

Connections

Connections define integrations with AWS services and data sources. They are created during target initialization.

Connection Configuration

```
initialization:
  connections:
    - name: connection-name
      type: CONNECTION_TYPE
      properties:
        # Type-specific properties
```

Supported Connection Types

S3 - Object Storage

```yaml
- name: s3-data-lake
  type: S3
  properties:
    s3Uri: "s3://my-data-bucket/data/"
```

Properties: - `s3Uri` (required): S3 bucket URI with optional prefix

IAM - Identity and Access Management

```yaml
- name: iam-lineage-sync
  type: IAM
  properties:
    glueLineageSyncEnabled: true
```

Properties: - `glueLineageSyncEnabled` (required): Enable Glue lineage sync (`true`/`false`)

SPARK_GLUE - Spark on AWS Glue

```yaml
- name: spark-processing
  type: SPARK_GLUE
  properties:
    glueVersion: "4.0"
    workerType: "G.1X"
    numberOfWorkers: 5
    maxRetries: 1
```

Properties: - `glueVersion` (required): Glue version (`"4.0"`, `"5.0"`) - `workerType` (required): Worker type (`"G.1X"`, `"G.2X"`) - `numberOfWorkers` (required): Number of workers - `maxRetries` (optional): Maximum retries

ATHENA - SQL Query Engine

```yaml
- name: athena-analytics
  type: ATHENA
  properties:
    workgroupName: "primary"
```

Properties: - `workgroupName` (required): Athena workgroup name

REDSHIFT - Data Warehouse

```yaml
- name: redshift-warehouse
  type: REDSHIFT
  properties:
    storage:
      clusterName: "analytics-cluster"
      # OR
      workgroupName: "analytics-workgroup"
    databaseName: "analytics"
    host: "analytics-cluster.abc123.us-east-1.redshift.amazonaws.com"
    port: 5439
```

Properties: - `storage.clusterName` (required for provisioned): Redshift cluster name - `storage.workgroupName` (required for serverless): Redshift serverless workgroup - `databaseName` (required): Database name - `host` (required): Redshift endpoint hostname - `port` (required): Port number (typically `5439`)

SPARK_EMR - Spark on EMR

```yaml
- name: spark-emr-processing
  type: SPARK_EMR
  properties:
    computeArn: "arn:aws:emr-serverless:us-east-
        1:123456789012:/applications/00abc123def456"
    runtimeRole:
        "arn:aws:iam::123456789012:role/EMRServerlessExecutionRole"
```

Properties: - `computeArn` (required): EMR compute ARN (serverless application or cluster) - `runtimeRole` (required): IAM role ARN for execution

MLFLOW - ML Experiment Tracking

```yaml
- name: mlflow-experiments
  type: MLFLOW
  properties:
    trackingServerName: "wine-classification-mlflow-v2"
    trackingServerArn: "arn:aws:sagemaker:us-east-
        1:123456789012:mlflow-tracking-server/wine-classification-
        mlflow-v2"
```

Properties: - `trackingServerName` (required): MLflow tracking server name - `trackingServerArn` (required): MLflow tracking server ARN

WORKFLOWS_MWAA - Apache Airflow Workflows

```yaml
- name: mwaa-workflows
  type: WORKFLOWS_MWAA
  properties:
    mwaaEnvironmentName: "production-airflow-env"
```

Properties: - `mwaaEnvironmentName` (required): MWAA environment name

WORKFLOWS_SERVERLESS - Serverless Airflow Workflows

```yaml
- name: serverless-workflows
  type: WORKFLOWS_SERVERLESS
  properties: {}
```

Properties: - No properties required (empty structure)

Connection Examples

Complete example with multiple connection types:

```yaml
initialization:
  connections:
    - name: s3-raw-data
      type: S3
      properties:
        s3Uri: "s3://raw-data-bucket/incoming/"

    - name: iam-lineage
      type: IAM
      properties:
        glueLineageSyncEnabled: true

    - name: spark-etl
      type: SPARK_GLUE
```

```yaml
    properties:
      glueVersion: "4.0"
      workerType: "G.2X"
      numberOfWorkers: 10

  - name: athena-queries
    type: ATHENA
    properties:
      workgroupName: "analytics-workgroup"

  - name: redshift-dw
    type: REDSHIFT
    properties:
      storage:
        clusterName: "analytics-cluster"
      databaseName: "analytics"
      host: "analytics-cluster.abc123.us-east-
      1.redshift.amazonaws.com"
      port: 5439

  - name: emr-processing
    type: SPARK_EMR
    properties:
      computeArn: "arn:aws:emr-serverless:us-east-
      1:123456789012:/applications/00abc123def456"
      runtimeRole:
      "arn:aws:iam::123456789012:role/EMRServerlessExecutionRole"

  - name: ml-tracking
    type: MLFLOW
    properties:
      trackingServerName: "ml-experiments-server"
      trackingServerArn: "arn:aws:sagemaker:us-east-
      1:123456789012:mlflow-tracking-server/ml-experiments-server"

  - name: airflow-orchestration
    type: WORKFLOWS_MWAA
    properties:
      mwaaEnvironmentName: "production-airflow-env"

  - name: serverless-workflows
    type: WORKFLOWS_SERVERLESS
    properties: {}
```

Parameterization

The pipeline manifest supports two types of parameterization:

1. **Manifest-level parameterization**: Environment variables in the manifest itself
2. **Workflow-level parameterization**: Parameters passed to workflow DAG files

Manifest-Level Parameterization

Use environment variables in the manifest with `${VAR_NAME:default_value}` syntax:

```yaml
pipelineName: ${PIPELINE_NAME:MarketingPipeline}

targets:
  dev:
```

```
    domain:
      name: ${DOMAIN_NAME:my-studio-domain}
      region: ${AWS_REGION:us-east-1}
    project:
      name: ${PROJECT_PREFIX:dev}-${TEAM_NAME:marketing}-project
```

Usage:

```
export DOMAIN_NAME=prod-datazone-domain
export AWS_REGION=us-west-2
export PROJECT_PREFIX=analytics
export TEAM_NAME=datascience

smus-cli deploy --pipeline pipeline.yaml --target dev
```

Resolution Rules: 1. If environment variable is set: Use the value 2. If environment variable is not set: Use default value 3. If no default value: Use empty string

Workflow-Level Parameterization

Parameters can be defined at two levels:

1. Global Workflow Parameters

Defined in the `workflows` section, apply to all targets:

```
workflows:
  - workflowName: marketing_etl_dag
    connectionName: project.workflow_mwaa
    parameters:
      data_source: s3://marketing-data/
      output_bucket: s3://marketing-results/
      timeout: 3600
```

2. Target Environment Variables

Defined in `targets[].environment_variables`, substituted in workflow files:

```
targets:
  dev:
    environment_variables:
      S3_PREFIX: "dev"
      DEBUG_MODE: true
      MAX_RETRIES: 3

  test:
    environment_variables:
      S3_PREFIX: "test"
      DEBUG_MODE: false
      MAX_RETRIES: 5
```

Using Parameters in Workflow DAG Files

Environment variables are substituted in workflow files using `${VAR_NAME}` or `$VAR_NAME` syntax:

```
# workflows/dags/marketing_dag.yaml
my_dag:
  dag_id: "data_processing"
  tasks:
    extract_data:
```

```
      operator:
        "airflow.providers.amazon.aws.operators.s3.S3ListOperator"
      bucket: "my-data-bucket"
      prefix: ${S3_PREFIX}        # Resolves to "dev" or "test"

    process_data:
      operator: "airflow.operators.python.PythonOperator"
      python_callable: "process_data"
      op_kwargs:
        debug: $DEBUG_MODE       # Resolves to true or false
        retries: $MAX_RETRIES    # Resolves to 3 or 5
```

Parameter Resolution Process

During deployment, the CLI: 1. Reads target's `environment_variables` configuration 2. Scans workflow files for `${VAR_NAME}` and `$VAR_NAME` patterns 3. Replaces variables with target-specific values 4. Uploads resolved files to the deployment environment

Example Resolution:

Source workflow file:

```
prefix: ${S3_PREFIX}
debug: $DEBUG_MODE
```

Dev target result:

```
prefix: dev
debug: true
```

Test target result:

```
prefix: test
debug: false
```

Supported Variable Types

Environment variables support multiple data types:

```
environment_variables:
  STRING_VAR: "my-string"      # String value
  BOOLEAN_VAR: true            # Boolean value
  NUMBER_VAR: 42               # Numeric value
  PREFIX_VAR: "data/staging"   # Path/prefix strings
```

Complete Parameterization Example

```
pipelineName: ${PIPELINE_NAME:DataPipeline}

targets:
  dev:
    domain:
      name: ${DOMAIN_NAME}
      region: ${AWS_REGION:us-east-1}
    project:
      name: ${PROJECT_PREFIX}-dev-project

    # Environment variables for workflow file substitution
    environment_variables:
      S3_PREFIX: "dev"
      DEBUG_MODE: true
```

```
      MAX_RETRIES: 3
      DB_HOST: "dev-db.company.com"

workflows:
  # Global workflow parameters
  - workflowName: etl_dag
    connectionName: project.workflow_mwaa
    parameters:
      data_source: s3://data-bucket/
      timeout: 3600
```

Workflow DAG file (`workflows/dags/etl_dag.yaml`):

```
etl_dag:
  dag_id: "etl_pipeline"
  default_args:
    retries: $MAX_RETRIES
  tasks:
    extract:
      operator:
        "airflow.providers.amazon.aws.operators.s3.S3ListOperator"
      bucket: "data-bucket"
      prefix: ${S3_PREFIX}

    transform:
      operator: "airflow.operators.python.PythonOperator"
      python_callable: "transform_data"
      op_kwargs:
        debug: $DEBUG_MODE
        db_host: ${DB_HOST}
```

Final merged parameters for dev target: - From global: `data_source`, `timeout` - From environment_variables (substituted in DAG): `S3_PREFIX`, `DEBUG_MODE`, `MAX_RETRIES`, `DB_HOST`

Validation Rules

Required Fields

- `pipelineName`
- `targets` (at least one target)
- Each target must have: `stage`, `domain.name`, `domain.region`, `project.name`

Optional Sections

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

Connection Names

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

File Patterns

- Support glob patterns: `*.py`, `**/*.yaml`

- Exclude patterns take precedence over include patterns
- Paths are relative to bundle source directories

Best Practices

Bundle Configuration

- Always exclude temporary files: `.ipynb_checkpoints/`, `__pycache__/`, `*.pyc`
- Use `append: true` for workflows (incremental updates)
- Use `append: false` for storage (clean deployments)
- Use S3 storage for production and CI/CD systems

Target Organization

- Use initialization only for non-production environments
- Keep production targets minimal and explicit
- Use consistent naming: `dev`, `test`, `prod`

Parameterization

- Use manifest-level parameters for infrastructure configuration
- Use workflow-level parameters for application configuration
- Use environment variables for target-specific values in workflows
- Always provide default values for optional parameters
- Use descriptive variable names (e.g., `DEV_DOMAIN_REGION` not `REGION`)

Workflow Parameters

- Define common parameters globally
- Override with target-specific parameters for environment differences
- Use environment variables for values that change per target
- Avoid hardcoded values - use parameters instead

Connection Management

- Create connections during initialization for new projects
- Use descriptive connection names
- Document connection requirements in project README
- Test connections after creation

Substitutions and Variables

← Back to Main README | Pipeline Manifest Reference

Dynamic variable substitution system for workflow YAMLs that resolves project-specific and environment values at deployment time.

Overview

The context resolver allows workflow YAMLs to use placeholder variables that are automatically replaced with actual values from the target project during deployment. This enables:

- **Single workflow definition** works across all targets (dev, test, prod)

- **No hardcoded values** in workflow YAMLs
- **Automatic discovery** of project connections and properties
- **Type-safe resolution** with validation

Variable Syntax

Variables use the format: `{namespace.property.path}`

Supported Namespaces

1. **`env`** - Environment variables
2. **`proj`** - Project properties and connections

Available Variables

Environment Variables

Access any environment variable defined in the pipeline manifest:

```
{env.VARIABLE_NAME}
```

Example:

```
region_name: '{env.AWS_REGION}'
s3_prefix: '{env.S3_PREFIX}'
```

Project Properties

Direct project-level properties:

Variable	Description	Example Value
`{proj.id}`	Project ID	`5vqwb22pn2da5j`
`{proj.name}`	Project name	`test-marketing`
`{proj.domain_id}`	Domain ID	`dzd-5b6m4h6c1yfch3`
`{proj.iam_role}`	Project IAM role ARN	`arn:aws:iam::123:role/ProjectRole`
`{proj.iam_role_arn}`	Project IAM role ARN (alias)	`arn:aws:iam::123:role/ProjectRole`
`{proj.iam_role_name}`	Project IAM role name only	`ProjectRole`
`{proj.kms_key_arn}`	KMS key ARN	`arn:aws:kms:us-east-1:123:key/abc`

Connection Properties

Access properties from any project connection using:

```
{proj.connection.<connection-name>.<property>}
```

S3 Shared Connection (`default.s3_shared`)

Variable	Description
`{proj.connection.default.s3_shared.s3Uri}`	Shared S3 bucket path
`{proj.connection.default.s3_shared.environmentUserRole}`	Connection IAM role

MLflow Connection (`project.mlflow-server.mlflow`)

Variable	Description
`{proj.connection.project.mlflow-server.mlflow.trackingServerArn}`	MLflow tracking server ARN
`{proj.connection.project.mlflow-server.mlflow.trackingServerName}`	MLflow tracking server name

Spark Connection (`default.spark`)

Variable	Description
`{proj.connection.default.spark.glueVersion}`	Glue version (e.g., `4.0`)
`{proj.connection.default.spark.workerType}`	Worker type (e.g., `G.1X`)
`{proj.connection.default.spark.numberOfWorkers}`	Number of workers

Athena Connection (`default.sql`)

Variable	Description
`{proj.connection.default.sql.workgroupName}`	Athena workgroup name

Usage Example

Before (Hardcoded Values)

```yaml
tasks:
  covid_database_discovery:
    operator:
        airflow.providers.amazon.aws.operators.glue.GlueJobOperator
    script_location: 's3://demo-bucket/glue-
        scripts/glue_s3_list_job.py'
    iam_role_name: OverdriveExecutionRole
    region_name: us-east-1
    create_job_kwargs:
      GlueVersion: '4.0'
```

After (Context Variables)

```yaml
tasks:
  covid_database_discovery:
    operator:
        airflow.providers.amazon.aws.operators.glue.GlueJobOperator
    script_location:
        '{proj.connection.default.s3_shared.s3Uri}etl/glue_s3_list_job.py'
    iam_role_name: '{proj.iam_role}'
    region_name: '{env.AWS_REGION}'
    create_job_kwargs:
      GlueVersion: '{proj.connection.default.spark.glueVersion}'
```

How It Works

1. Deployment Time Resolution

When you run `deploy test`, the system:

1. Loads the target configuration from the pipeline manifest
2. Initializes the MaxDome Project SDK for the target project
3. Queries all project connections dynamically
4. Downloads workflow YAMLs from S3
5. Resolves all `{env.*}` and `{proj.*}` variables
6. Uploads resolved YAMLs back to S3
7. Creates Overdrive workflows with resolved content

2. Target-Specific Values

The same workflow YAML automatically gets different values per target:

Dev Target: - `{proj.name}` → `dev-marketing` - `{proj.connection.default.s3_shared.s3Uri}` → `s3://dev-bucket/` - `{env.AWS_REGION}` → `us-west-2`

Test Target: - `{proj.name}` → `test-marketing` - `{proj.connection.default.s3_shared.s3Uri}` → `s3://test-bucket/` - `{env.AWS_REGION}` → `us-east-1`

3. Dynamic Connection Discovery

The resolver automatically discovers ALL connections in the project, not just predefined ones. If you add a new connection to your project, it's immediately available:

```yaml
# New Redshift connection added to project
script_location: '{proj.connection.my-redshift.host}'
```

Pipeline Manifest Configuration

Define environment variables in your pipeline manifest:

```yaml
targets:
  test:
    project:
      name: test-marketing
    environment_variables:
      AWS_REGION: us-east-1
      S3_PREFIX: test
      CUSTOM_VAR: value
```

Error Handling

Missing Variables

If a variable cannot be resolved, it remains unchanged in the output:

```yaml
# Input
role: '{proj.connection.does-not-exist.role}'

# Output (if connection not found)
role: '{proj.connection.does-not-exist.role}'
```

Invalid Syntax

Variables must follow the exact syntax `{namespace.path}`:

- ✅ `{env.AWS_REGION}`
- ✅ `{proj.iam_role}`
- ✅ `{proj.connection.default.s3_shared.s3Uri}`
- ❌ `${env.AWS_REGION}` (wrong brackets)
- ❌ `{AWS_REGION}` (missing namespace)

Best Practices

1. Use Descriptive Environment Variables

```yaml
environment_variables:
  AWS_REGION: us-east-1
  GLUE_DATABASE_NAME: analytics_db
  DATA_BUCKET_PREFIX: raw-data
```

2. Leverage Project Connections

Instead of hardcoding S3 paths, use connection properties:

```yaml
# ✅ Good
input_path: '{proj.connection.default.s3_shared.s3Uri}input/'

# ❌ Bad
input_path: 's3://hardcoded-bucket/input/'
```

3. Keep Workflow YAMLs Generic

Write workflows that work for any project:

```yaml
tasks:
  process_data:
    script_location:
        '{proj.connection.default.s3_shared.s3Uri}scripts/process.py'
    iam_role: '{proj.iam_role}'
    region: '{env.AWS_REGION}'
    output_path: '{proj.connection.default.s3_shared.s3Uri}output/'
```

4. Document Required Connections

In your workflow README, document which connections are required:

```
## Required Connections

This workflow requires the following project connections:

- `default.s3_shared` - For script storage and output
- `default.spark` - For Glue job configuration
- `project.mlflow-server.mlflow` - For experiment tracking
```

Implementation Details

Context Resolver Class

Located in: `src/smus_cicd/helpers/context_resolver.py`

Pipeline Test Configuration

When running `smus test`, a configuration file `.smus_test_config.json` is created in the test folder with resolved project context:

```
{
  "region": "us-east-1",
  "project_id": "4pg255jku47vdz",
  "project_name": "test-marketing",
  "domain_id": "dzd-5b6m4h6c1yfch3",
  "domain_name": "BETA_10282025_Domain",
  "target_name": "test",
  "iam_role": "arn:aws:iam::123456789:role/ProjectRole",
  "iam_role_arn": "arn:aws:iam::123456789:role/ProjectRole",
  "iam_role_name": "ProjectRole",
  "kms_key_arn": "arn:aws:kms:us-east-1:123456789:key/abc",
  "connections": {
    "default.s3_shared": {
      "s3Uri": "s3://bucket/shared/",
      "bucket_name": "bucket",
      "environmentUserRole": "arn:aws:iam::123:role/Role"
    },
    "default.spark": {
      "sparkGlueProperties": {...}
    }
  },
  "env": {
    "AWS_REGION": "us-east-1"
  }
}
```

Access in tests via the `smus_config` fixture (provided by `conftest.py`):

```
def test_example(smus_config):
    region = smus_config['region']
    project_id = smus_config['project_id']
    s3_uri = smus_config['connections']['default.s3_shared']['s3Uri']

from sagemaker_studio.project import Project
from smus_cicd.helpers.context_resolver import ContextResolver

# Initialize with project
project = Project(name="test-marketing", domain_id="dzd-123")
resolver = ContextResolver(project, env_vars={"AWS_REGION": "us-east-
        1"})

# Resolve content
yaml_content = "role: '{proj.iam_role}'"
resolved = resolver.resolve(yaml_content)
# Output: "role: 'arn:aws:iam::123:role/ProjectRole'"
```

Integration with Deploy Command

The deploy command automatically applies context resolution:

```
# In deploy.py
from sagemaker_studio.project import Project
from ..helpers.context_resolver import ContextResolver

project = Project(name=project_name, domain_id=domain_id)
resolver = ContextResolver(project,
        env_vars=target_config.environment_variables)

# Download, resolve, upload workflow YAML
resolved_content = resolver.resolve(original_content)
```

Testing

Unit tests: `tests/unit/test_context_resolver.py`

```
pytest tests/unit/test_context_resolver.py -v
```

Future Enhancements

Potential additions:

1. **Conditional resolution**: `{proj.connection.optional?.property}`
2. **Default values**: `{env.VAR:default_value}`
3. **Nested references**: `{proj.connection.{env.CONN_NAME}.property}`
4. **Validation mode**: Pre-deployment check for missing variables
5. **Custom functions**: `{upper(proj.name)}`, `{concat(proj.s3.root, '/data')}`

Troubleshooting

Variable Not Resolving

1. Check variable syntax matches exactly: `{namespace.property}`
2. Verify connection exists in project: `smus describe --pipeline manifest.yaml --connect`
3. Check environment variable is defined in pipeline manifest
4. Review deploy logs for resolution errors

Connection Property Not Found

1. List all connections: `smus describe --pipeline manifest.yaml --connect`
2. Check connection type matches expected properties
3. Verify connection is in READY status
4. Check property name spelling (case-sensitive)

Wrong Value After Resolution

1. Verify you're deploying to correct target
2. Check target configuration in pipeline manifest
3. Confirm project has correct connections configured
4. Review resolved YAML in S3 after deployment

Monitoring and Alerts with EventBridge

The SMUS CICD CLI emits deployment lifecycle events to Amazon EventBridge, enabling you to build custom monitoring, alerting, and automation workflows.

Overview

During deployment, the CLI emits events at key stages: - Deploy started/completed/failed - Project initialization started/completed/failed - Bundle upload started/completed/failed - Workflow creation started/completed/failed - Catalog assets processing started/completed/failed

Configuration

Pipeline Manifest

Add monitoring configuration to your `pipeline.yaml`:

```yaml
pipelineName: MyPipeline

monitoring:
  eventbridge:
    enabled: true               # Enable/disable event emission
    eventBusName: default       # Event bus name or ARN
    includeMetadata: true       # Include git, CI/CD, user metadata

targets:
  prod:
    # ... target configuration
```

CLI Options

Override manifest configuration with CLI flags:

```bash
# Enable events
smus-cli deploy --targets prod --emit-events

# Disable events
smus-cli deploy --targets prod --no-events

# Use custom event bus
smus-cli deploy --targets prod --event-bus-name my-custom-bus
```

Event Schema

All events follow this structure:

```
{
  "version": "1.0",
  "timestamp": "2025-11-04T12:30:00Z",
  "pipelineName": "MyPipeline",
  "stage": "deploy|project-init|bundle-upload|workflow-
        creation|catalog-assets",
  "status": "started|completed|failed",
  "target": {
    "name": "prod",
```

```
    "stage": "PROD",
    "domain": {
      "name": "my-domain",
      "region": "us-east-1"
    },
    "project": {
      "name": "prod-project"
    }
  },
  "metadata": {
    "user": "username",
    "gitCommit": "abc123",
    "gitBranch": "main",
    "gitRepository": "https://github.com/...",
    "cicdPlatform": "github-actions",
    "runId": "12345",
    "runUrl": "https://github.com/.../actions/runs/12345"
  }
}
```

Event Types

Deploy Events

- `SMUS-CICD-Deploy-Started` - Deployment begins
- `SMUS-CICD-Deploy-Completed` - Deployment succeeds
- `SMUS-CICD-Deploy-Failed` - Deployment fails

Project Initialization Events

- `SMUS-CICD-Deploy-ProjectInitialization-Started`
- `SMUS-CICD-Deploy-ProjectInitialization-Completed`
- `SMUS-CICD-Deploy-ProjectInitialization-Failed`

Bundle Upload Events

- `SMUS-CICD-Deploy-BundleUpload-Started`
- `SMUS-CICD-Deploy-BundleUpload-Completed`
- `SMUS-CICD-Deploy-BundleUpload-Failed`

Workflow Creation Events

- `SMUS-CICD-Deploy-WorkflowCreation-Started`
- `SMUS-CICD-Deploy-WorkflowCreation-Completed`
- `SMUS-CICD-Deploy-WorkflowCreation-Failed`

Catalog Assets Events

- `SMUS-CICD-Deploy-CatalogAssets-Started`
- `SMUS-CICD-Deploy-CatalogAssets-Completed`
- `SMUS-CICD-Deploy-CatalogAssets-Failed`

Integration Examples

SNS Notification on Production Failures

```
{
  "EventPattern": {
    "source": ["com.amazon.smus.cicd"],
    "detail-type": ["SMUS-CICD-Deploy-Failed"],
    "detail": {
      "target": {"stage": ["PROD"]}
    }
  },
  "Targets": [{
    "Arn": "arn:aws:sns:us-east-1:123456789012:prod-alerts",
    "Id": "1"
  }]
}
```

Lambda Post-Deployment Validation

```
{
  "EventPattern": {
    "source": ["com.amazon.smus.cicd"],
    "detail-type": ["SMUS-CICD-Deploy-Completed"],
    "detail": {
      "pipelineName": ["MyPipeline"],
      "target": {"stage": ["PROD"]}
    }
  },
  "Targets": [{
    "Arn": "arn:aws:lambda:us-east-1:123456789012:function:validate-
        deployment",
    "Id": "1"
  }]
}
```

Step Functions Smoke Tests

```
{
  "EventPattern": {
    "source": ["com.amazon.smus.cicd"],
    "detail-type": ["SMUS-CICD-Deploy-WorkflowCreation-Completed"]
  },
  "Targets": [{
    "Arn": "arn:aws:states:us-east-1:123456789012:stateMachine:smoke-
        tests",
    "RoleArn":
        "arn:aws:iam::123456789012:role/EventBridgeToStepFunctions",
    "Id": "1"
  }]
}
```

CloudWatch Logs for All Events

```
{
  "EventPattern": {
    "source": ["com.amazon.smus.cicd"],
    "detail-type": [{"prefix": "SMUS-CICD-Deploy-"}]
  },
```

```
  "Targets": [{
    "Arn": "arn:aws:logs:us-east-1:123456789012:log-
        group:/aws/events/smus-deploy",
    "Id": "1"
  }]
}
```

IAM Permissions

The CLI requires `events:PutEvents` permission:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "events:PutEvents",
      "Resource": "arn:aws:events:*:*:event-bus/default"
    }
  ]
}
```

Metadata Collection

When `includeMetadata: true`, events include:

- User: Current system user
- Git Info: Commit hash, branch, repository URL
- CI/CD Platform: Detected platform (GitHub Actions, GitLab CI, Jenkins, etc.)
- Run Info: Run ID and URL (when available)

Supported CI/CD platforms: - GitHub Actions - GitLab CI - Jenkins - CircleCI - AWS CodeBuild - Travis CI

Best Practices

1. Use specific event patterns - Filter by pipeline name and stage to reduce noise
2. Enable metadata - Helps with debugging and audit trails
3. Monitor event delivery - Set up CloudWatch alarms for failed event deliveries
4. Test integrations - Use test deployments to verify event rules work correctly
5. Secure event buses - Use IAM policies to control who can put events

Troubleshooting

Events not appearing

1. Check IAM permissions for `events:PutEvents`
2. Verify event bus name is correct
3. Check EventBridge rule patterns match event structure
4. Enable CLI debug logging: `export SMUS_LOG_LEVEL=DEBUG`

Missing metadata

1. Ensure `includeMetadata: true` in manifest
2. Check git repository is initialized
3. Verify CI/CD environment variables are set

Example Workflows

Slack Notifications

Use EventBridge → SNS → Lambda → Slack webhook to send deployment notifications.

Automated Rollback

Use EventBridge → Lambda to trigger rollback on deployment failures.

Deployment Dashboard

Use EventBridge → CloudWatch Logs → CloudWatch Insights to build deployment dashboards.

Approval Workflows

Use EventBridge → Step Functions to implement manual approval gates for production deployments.

CI/CD Integration for SMUS Pipeline Management

← Back to Main README

This guide explains how to integrate the SMUS CLI with CI/CD platforms like GitHub Actions and GitLab CI for automated pipeline management across multiple environments with proper security boundaries and approval workflows.

Overview: CI/CD with SMUS CLI

The SMUS CLI enables automated deployment and management of SageMaker Unified Studio projects through any CI/CD platform. The CLI provides intelligent infrastructure management, multi-environment support, and seamless integration with existing DevOps workflows.

Key Benefits

- **Automated Infrastructure**: CLI creates domains, projects, and environments as needed
- **Idempotent Operations**: Safe to re-run without duplicating resources
- **Multi-Environment Support**: Seamless progression from dev → test → prod
- **Manual Approval Gates**: Production deployments with reviewer approval
- **Catalog Asset Integration**: Automatic subscription management for DataZone assets
- **Bundle Management**: Consistent artifact deployment across environments

Pipeline Flow

```
graph LR
    A[Source Code] --> B[Validate Configuration]
    B --> C[Create Bundle]
    C --> D[Deploy to Test]
    D --> E[Run Tests]
```

```
    E --> F[Manual Approval Gate]
    F --> G[Deploy to Production]
    G --> H[Production Validation]
    H --> I[Monitor & Alert]
```

Infrastructure Setup Requirements

AWS Prerequisites

Before setting up CI/CD integration, ensure you have the necessary AWS infrastructure and permissions:

1. AWS Account Setup

- AWS account with appropriate permissions
- SageMaker Unified Studio enabled in target regions
- DataZone service activated (if using catalog assets)

2. IAM Roles and Permissions

Create IAM roles for CI/CD with these minimum permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:*",
        "datazone:*",
        "mwaa:*",
        "s3:*",
        "cloudformation:*",
        "iam:PassRole",
        "iam:GetRole",
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

3. OIDC Provider Setup (GitHub Actions)

For GitHub Actions, create an OIDC identity provider:

```
# Deploy OIDC integration stack
aws cloudformation deploy \
  --template-file github-oidc-role.yaml \
  --stack-name smus-cli-github-integration \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    GitHubOrg=your-org \
    GitHubRepo=your-repo \
    GitHubEnvironment=aws-env
```

Environment Separation

Development Environment

- **Purpose**: Development and testing deployments
- **AWS Role**: Broader permissions for experimentation
- **Protection Rules**: None (automatic deployment)
- **Resources**: Development domains, test projects

Production Environment

- **Purpose**: Production deployments
- **AWS Role**: Restricted permissions, production-only access
- **Protection Rules**: Required reviewers, manual approval
- **Resources**: Production domains, live projects

CLI Infrastructure Management

Automatic Infrastructure Creation

The SMUS CLI intelligently manages infrastructure creation and updates:

Idempotent Operations

- **Safe Re-runs**: CLI detects existing resources and skips creation
- **No Duplicates**: Won't create domains, projects, or environments that already exist
- **Consistent State**: Ensures infrastructure matches pipeline configuration

Domain Management

```
# CLI automatically creates DataZone domain if missing
smus-cli deploy --pipeline pipeline.yaml --targets test
# Creates domain "my-studio-domain" if it doesn't exist
# Uses existing domain if already present
```

Project Creation

```
# CLI creates SageMaker projects as needed
smus-cli deploy --pipeline pipeline.yaml --targets test,prod
# Creates "my-project-test" and "my-project-prod" projects
# Skips creation if projects already exist
```

Environment Provisioning

- **Lakehouse Environments**: Automatically provisioned for data access
- **MWAA Environments**: Created for Airflow workflow execution
- **Connections**: Established between projects and environments
- **Permissions**: Proper IAM roles and policies applied

Benefits of Automatic Infrastructure

- **Reduced Setup Time**: No manual infrastructure preparation required
- **Consistency**: Infrastructure matches pipeline configuration exactly
- **Error Prevention**: Eliminates manual configuration mistakes
- **Team Onboarding**: New team members can deploy immediately

Infrastructure Detection Logic

The CLI uses intelligent detection to determine what needs to be created:

1. **Domain Resolution**: Looks up domain by name, creates if missing
2. **Project Validation**: Checks project existence, creates with proper configuration
3. **Environment Assessment**: Validates required environments, provisions as needed
4. **Connection Verification**: Ensures proper connectivity between components

CI/CD Environment Configuration

GitHub Environments

Configure different GitHub environments for deployment stages:

Development Environment (`aws-env`)

```yaml
# Repository Settings → Environments → aws-env
Protection Rules: None
Secrets:
  AWS_ROLE_ARN: arn:aws:iam::ACCOUNT:role/GitHubActions-Dev
Variables:
  AWS_REGION: us-east-1
  DOMAIN_REGION: us-east-1
```

Production Environment (`aws-env-prod`)

```yaml
# Repository Settings → Environments → aws-env-prod
Protection Rules:
  - Required reviewers: 2
  - Wait timer: 5 minutes
  - Deployment branches: main only
Secrets:
  AWS_ROLE_ARN: arn:aws:iam::ACCOUNT:role/GitHubActions-Prod
Variables:
  AWS_REGION: us-west-2
  DOMAIN_REGION: us-west-2
```

Secrets Management

Required Secrets

- `AWS_ROLE_ARN`: IAM role for AWS authentication
- `AWS_ACCESS_KEY_ID`: Alternative to OIDC (less secure)
- `AWS_SECRET_ACCESS_KEY`: Alternative to OIDC (less secure)

Environment Variables

- `AWS_REGION`: Target AWS region
- `DOMAIN_NAME`: SageMaker Unified Studio domain name
- `PROJECT_PREFIX`: Prefix for project naming

Branch Protection Rules

Configure branch protection to ensure code quality:

```yaml
# Repository Settings → Branches → main
Branch Protection Rules:
```

```
- Require pull request reviews
- Require status checks to pass
- Require branches to be up to date
- Restrict pushes to matching branches
```

Pipeline Stages and Workflow

Stage Progression

The SMUS CLI supports a standard deployment progression with automatic infrastructure management:

Development → Test → Production Flow

```
graph TD
    A[Development Environment] --> B[Create Bundle]
    B --> C[Deploy to Test]
    C --> D[Run Tests]
    D --> E{Tests Pass?}
    E -->|Yes| F[Manual Approval Gate]
    E -->|No| G[Fix Issues]
    G --> A
    F --> H[Deploy to Production]
    H --> I[Production Validation]
    I --> J[Monitor & Alert]
```

Bundle Creation and Management

Bundle Creation Process

```
# CLI packages all deployment artifacts
smus-cli bundle --pipeline pipeline.yaml --targets test
```

What gets bundled: - Airflow DAGs and dependencies - Jupyter notebooks and scripts - Configuration files and schemas - DataZone catalog asset references - Environment-specific parameters

Artifact Management

- **S3 Storage**: Bundles uploaded to environment-specific S3 buckets
- **Versioning**: Each bundle tagged with commit SHA or version
- **Reuse**: Same bundle deployed across test → production
- **Rollback**: Previous bundles available for quick rollback

Infrastructure-Aware Deployment

Automatic Infrastructure Provisioning

The CLI handles infrastructure creation during deployment:

```
# Single command creates all required infrastructure
smus-cli deploy --pipeline pipeline.yaml --targets test

# CLI automatically:
# 1. Creates DataZone domain if missing
# 2. Creates SageMaker project if missing
# 3. Provisions required environments (Lakehouse, MWAA)
```

```
# 4. Establishes connections between components
# 5. Deploys bundle to target environment
```

Infrastructure Detection Logic

1. **Domain Resolution**: Resolves domain by name, creates if not found
2. **Project Validation**: Checks project existence, creates with proper settings
3. **Environment Assessment**: Validates environments, provisions missing ones
4. **Connection Setup**: Ensures proper connectivity and permissions

Idempotent Operations

- **Safe Re-runs**: Multiple deployments won't create duplicate resources
- **State Consistency**: Infrastructure matches pipeline configuration
- **Error Recovery**: Failed deployments can be safely retried

Catalog Asset Integration

Automatic Subscription Management

The CLI handles DataZone catalog asset access throughout the pipeline:

```
# In pipeline.yaml - catalog assets are automatically managed
bundle:
  catalog:
    assets:
      - selector:
          search:
            assetType: GlueTable
            identifier: covid19_db.countries_aggregated
        permission: READ
        requestReason: Required for pipeline deployment
```

CLI Automation: 1. **Asset Discovery**: Identifies required catalog assets in bundle 2. **Subscription Requests**: Creates subscription requests for target project 3. **Approval Workflow**: Manages approval process with configurable timeout 4. **Grant Verification**: Ensures access is granted before proceeding 5. **Cross-Environment**: Handles different subscriptions per environment

Manual Approval Gates

Production Deployment Protection

Configure manual approval for production deployments using CI/CD platform features:

GitHub Actions Example:

```
deploy-production:
  name: "Deploy to Production"
  runs-on: ubuntu-latest
  environment: aws-env-prod  # Triggers manual approval
  needs: [test-validation]
  steps:
    - name: Deploy to Production
      run: |
        smus-cli deploy --pipeline pipeline.yaml --targets prod
```

GitLab CI Example:

```yaml
deploy-production:
  stage: deploy
  environment:
    name: production
    action: start
  when: manual  # Requires manual trigger
  script:
    - smus-cli deploy --pipeline pipeline.yaml --targets prod
```

Approval Workflow Process

1. **Workflow Pause**: CI/CD pipeline pauses at production deployment
2. **Notification**: Designated reviewers receive notification
3. **Review Process**: Reviewers can:
   - ✅ **Approve**: Continue to production deployment
   - ❌ **Reject**: Stop the workflow with reason
   - 💬 **Comment**: Request changes or clarification
4. **Audit Trail**: All approvals logged for compliance
5. **Deployment**: Approved deployments proceed automatically

CLI Integration Points

1. Configuration Validation

```bash
# Validates pipeline configuration and connectivity
smus-cli describe --pipeline pipeline.yaml --connect
```

2. Bundle Creation

```bash
# Creates deployment bundle from source environment
smus-cli bundle --pipeline pipeline.yaml --targets test
```

3. Infrastructure-Aware Deployment

```bash
# Deploys with automatic infrastructure creation
smus-cli deploy --pipeline pipeline.yaml --targets test
```

4. Test Execution

```bash
# Runs validation tests on deployed environment
smus-cli test --pipeline pipeline.yaml --targets test
```

5. Production Deployment

```bash
# Deploys to production with approval gates
smus-cli deploy --pipeline pipeline.yaml --targets prod
```

6. Monitoring and Status

```bash
# Monitors deployment status and health
smus-cli monitor --pipeline pipeline.yaml --targets prod
```

Manual Reviewers and Approval Process

Setting Up Approval Gates

GitHub Environment Protection Rules

1. **Navigate to Repository Settings**
   - Go to **Settings** → **Environments**
   - Select or create production environment

2. **Configure Protection Rules**

```
Environment: aws-env-prod
Protection Rules:
  - Required reviewers: 2-3 team members
  - Wait timer: 5-10 minutes (optional)
  - Deployment branches: main, release/* only
  - Prevent self-review: enabled
```

3. **Reviewer Assignment**
   - **Team Leads**: Senior developers or architects
   - **DevOps Engineers**: Infrastructure and deployment experts
   - **Product Owners**: Business stakeholders for critical releases
   - **Security Team**: For compliance-sensitive deployments

GitLab Environment Protection

```yaml
# .gitlab-ci.yml
deploy-production:
  stage: deploy
  environment:
    name: production
    deployment_tier: production
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
      when: manual
      allow_failure: false
  script:
    - smus-cli deploy --pipeline pipeline.yaml --targets prod
```

Approval Workflow Mechanics

Notification Process

1. **Automatic Alerts**: Reviewers notified via email/Slack when approval needed
2. **Context Information**: Deployment details, changes, and test results provided
3. **Review Dashboard**: Centralized view of pending approvals

Review Criteria

- **Code Quality**: All tests passing, code review completed
- **Security Validation**: Security scans passed, no vulnerabilities
- **Business Impact**: Change aligns with business requirements
- **Rollback Plan**: Clear rollback strategy documented

Approval Actions

```
# Reviewer options in GitHub/GitLab UI
Actions Available:
  - Approve Deployment: ✅ Proceed with production deployment
  - Request Changes: 🔄 Block deployment, request modifications
  - Reject Deployment: ❌ Stop deployment with reason
  - Add Comments: 💬 Provide feedback or ask questions
```

Audit Trail and Compliance

Deployment History

- **Who**: User who initiated deployment
- **What**: Changes being deployed (commit SHA, bundle version)
- **When**: Timestamp of deployment request and approval
- **Why**: Deployment reason and business justification
- **Reviewers**: Who approved/rejected with comments

Compliance Features

- **SOX Compliance**: Separation of duties between developers and approvers
- **Change Management**: Integration with ITSM tools for change requests
- **Audit Logs**: Immutable logs of all deployment activities
- **Rollback Tracking**: Complete history of rollbacks and reasons

Specific Implementation Example: aws/Unified-Studio-for-Amazon-Sagemaker

This section demonstrates how the general concepts above are implemented in the specific GitHub repository `https://github.com/aws/Unified-Studio-for-Amazon-Sagemaker/` using the `.github/workflows/full-pipeline-lifecycle.yml` workflow.

Repository-Specific Configuration

GitHub Environments Used

- **`aws-env`**: Development environment for testing and validation
- **`aws-env-amirbo-6778`**: Production environment with manual approval gates

Multi-Environment AWS Role Configuration

```yaml
# Development Role (broader permissions for testing)
- name: Configure AWS credentials (Dev)
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-region: us-east-1
    role-to-assume: ${{ secrets.AWS_ROLE_ARN_DEV }}
    role-session-name: smus-pipeline-lifecycle-dev
    role-duration-seconds: 43200

# Production Role (restricted permissions)
- name: Configure AWS credentials (Prod)
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-region: us-east-1
    role-to-assume: ${{ secrets.AWS_ROLE_ARN_PROD }}
```

```
    role-session-name: smus-pipeline-lifecycle-prod
    role-duration-seconds: 43200
```

Workflow Implementation Details

The `full-pipeline-lifecycle.yml` workflow demonstrates a complete end-to-end pipeline with the following jobs:

Job 1: Setup and Environment Resolution

```
setup:
  name: "Step 1 - Setup and Resolve Environment"
  runs-on: ubuntu-latest
  environment: aws-env
  outputs:
    domain-id: ${{ steps.resolve.outputs.domain-id }}
    project-id: ${{ steps.resolve.outputs.project-id }}
```

Purpose: Resolves DataZone domain and project IDs from user-friendly names **Infrastructure Management**: CLI automatically detects existing domains or creates them if missing

Job 2: Bundle Creation

```
create-bundle:
  name: "Step 2 - Create Bundle"
  runs-on: ubuntu-latest
  environment: aws-env
  needs: setup
```

Purpose: Packages deployment artifacts including DAGs, notebooks, and catalog assets **CLI Command**: `smus-cli bundle --pipeline pipeline.yaml --targets test` **Infrastructure Impact**: Creates S3 storage for bundles if not present

Job 3: Test Environment Deployment

```
deploy-test:
  name: "Step 3 - Deploy to Test Environment"
  runs-on: ubuntu-latest
  environment: aws-env
  needs: [setup, create-bundle]
```

Purpose: Deploys to test environment with automatic infrastructure provisioning **CLI Command**: `smus-cli deploy --pipeline pipeline.yaml --targets test` **Infrastructure Created**: - SageMaker project (if missing) - Lakehouse environment (if missing) - MWAA environment (if missing) - Required IAM roles and policies

Job 4: Test Execution

```
run-tests:
  name: "Step 4 - Run Tests on Test Environment"
  runs-on: ubuntu-latest
  environment: aws-env
  needs: [setup, create-bundle, deploy-test]
```

Purpose: Validates deployment through automated testing **CLI Command**: `smus-cli test --pipeline pipeline.yaml --targets test`

Job 5: Production Deployment (Manual Approval)

```yaml
deploy-prod:
  name: "Step 5 - Deploy to Production Environment"
  runs-on: ubuntu-latest
  environment: aws-env-amirbo-6778  # Triggers manual approval
  needs: [setup, create-bundle, deploy-test, run-tests]
```

Manual Approval Gate: Uses GitHub environment `aws-env-amirbo-6778` with protection rules **CLI Command**: `smus-cli deploy --pipeline pipeline.yaml --targets prod` **Infrastructure Management**: Creates production infrastructure idempotently

Repository-Specific Features

Workflow Dispatch Inputs

```yaml
on:
  workflow_dispatch:
    inputs:
      domain_name:
        description: 'SageMaker Unified Studio domain name'
        required: true
        default: 'cicd-test-domain'
      project_name:
        description: 'Development project name'
        required: true
        default: 'cicd-test-project'
      pipeline_name:
        description: 'Pipeline name'
        required: true
        default: 'TestPipeline'
```

Environment Variables

```yaml
env:
  DEV_DOMAIN_REGION: ${{ vars.DEV_DOMAIN_REGION }}
  PROD_DOMAIN_REGION: ${{ vars.PROD_DOMAIN_REGION }}
  SMUS_LOG_LEVEL: WARNING
```

Artifact Management

```yaml
- name: Upload Pipeline Manifest
  uses: actions/upload-artifact@v4
  with:
    name: pipeline-manifest
    path: pipeline.yaml
    retention-days: 30
```

Security Implementation

OIDC Integration

The repository uses AWS OIDC for secure authentication without long-lived credentials:

```yaml
permissions:
  id-token: write
  contents: read
```

Environment Protection Rules

- **Development (`aws-env`)**: No protection, automatic deployment
- **Production (`aws-env-amirbo-6778`)**:
  - Required reviewers: 2 team members
  - Manual approval required
  - Deployment branch restrictions

Catalog Asset Integration Example

The workflow demonstrates automatic catalog asset management:

```
# CLI automatically handles catalog asset subscriptions
- name: Deploy with Catalog Assets
  run: |
    smus-cli deploy --pipeline pipeline.yaml --targets prod
    # CLI automatically:
    # 1. Identifies catalog assets in bundle
    # 2. Requests subscriptions for production project
    # 3. Waits for approval (with timeout)
    # 4. Verifies grants before proceeding
```

Monitoring and Cleanup

Pipeline Monitoring

```
monitor-pipeline:
  name: "Step 6 - Monitor Pipeline Status"
  runs-on: ubuntu-latest
  environment: aws-env-amirbo-6778
  needs: [deploy-prod]
```

Automatic Cleanup

```
cleanup:
  name: "Step 7 - Cleanup Test Resources"
  runs-on: ubuntu-latest
  environment: aws-env
  if: always()  # Runs even if previous jobs fail
```

Repository Setup for This Implementation

Required Secrets

```
# Repository Settings → Secrets and Variables → Actions
AWS_ROLE_ARN_DEV = arn:aws:iam::588738596778:role/GitHubActions-Dev
AWS_ROLE_ARN_PROD = arn:aws:iam::588738596778:role/GitHubActions-Prod
```

Environment Variables

```
# Environment: aws-env
DEV_DOMAIN_REGION = us-east-1

# Environment: aws-env-amirbo-6778
PROD_DOMAIN_REGION = us-east-2
```

GitHub Environment Configuration

1. **aws-env** (Development)
   - No protection rules
   - Used for: setup, bundling, test deployment, testing
2. **aws-env-amirbo-6778** (Production)
   - Protection rules: Required reviewers (2)
   - Used for: production deployment, monitoring

This implementation showcases how the SMUS CLI's infrastructure management capabilities work in practice, automatically creating and managing AWS resources while providing secure, auditable deployments with proper approval workflows.

For more detailed information about CLI commands and pipeline configuration, see: - CLI Commands Documentation - Pipeline Manifest Guide - Development Guide

AWS Airflow Operators Reference

This document provides a comprehensive list of AWS operators available for Apache Airflow workflows in SageMaker Unified Studio.

Core Operators

EmptyOperator

- **EmptyOperator** - `airflow.operators.empty.EmptyOperator` - No-op operator for workflow control

AWS Service Operators

AWS Batch

- **BatchCreateComputeEnvironmentOperator** - `airflow.providers.amazon.aws.operators.batch.BatchCreateComputeEnvironmentOperator` - BatchCreateComputeEnvironment operations
- **BatchOperator** - `airflow.providers.amazon.aws.operators.batch.BatchOperator` - Batch operations

AWS CloudFormation

- **CloudFormationCreateStackOperator** - `airflow.providers.amazon.aws.operators.cloudformation.CloudFormationCreateStackOperator` - CloudFormationCreateStack operations
- **CloudFormationDeleteStackOperator** - `airflow.providers.amazon.aws.operators.cloudformation.CloudFormationDeleteStackOperator` - CloudFormationDeleteStack operations

AWS DMS

- **DmsCreateTaskOperator** - `airflow.providers.amazon.aws.operators.dms.DmsCreateTaskOperator` - DmsCreateTask operations
- **DmsDeleteTaskOperator** - `airflow.providers.amazon.aws.operators.dms.DmsDeleteTaskOperator` - DmsDeleteTask operations

- **DmsDescribeTasksOperator** - `airflow.providers.amazon.aws.operators.dms.DmsDescribeTasksOperator` - DmsDescribeTasks operations
- **DmsStartTaskOperator** - `airflow.providers.amazon.aws.operators.dms.DmsStartTaskOperator` - DmsStartTask operations
- **DmsStopTaskOperator** - `airflow.providers.amazon.aws.operators.dms.DmsStopTaskOperator` - DmsStopTask operations

AWS DataSync

- **DataSyncOperator** - `airflow.providers.amazon.aws.operators.datasync.DataSyncOperator` - DataSync operations

AWS Glue

- **GlueCrawlerOperator** - `airflow.providers.amazon.aws.operators.glue.GlueCrawlerOperator` - Glue Crawler operations
- **GlueDataBrewStartJobOperator** - `airflow.providers.amazon.aws.operators.glue.GlueDataBrewStartJobOperator` - Glue DataBrewStartJob operations
- **GlueDataQualityOperator** - `airflow.providers.amazon.aws.operators.glue.GlueDataQualityOperator` - Glue DataQuality operations
- **GlueDataQualityRuleRecommendationRunOperator** - `airflow.providers.amazon.aws.operators.glue.GlueDataQualityRuleRecommendationRunOperator` - Glue DataQualityRuleRecommendationRun operations
- **GlueDataQualityRuleSetEvaluationRunOperator** - `airflow.providers.amazon.aws.operators.glue.GlueDataQualityRuleSetEvaluationRunOperator` - Glue DataQualityRuleSetEvaluationRun operations
- **GlueJobOperator** - `airflow.providers.amazon.aws.operators.glue.GlueJobOperator` - Glue Job operations

AWS Lambda

- **LambdaCreateFunctionOperator** - `airflow.providers.amazon.aws.operators.lambda_function.LambdaCreateFunctionOperator` - Lambda CreateFunction operations
- **LambdaInvokeFunctionOperator** - `airflow.providers.amazon.aws.operators.lambda_function.LambdaInvokeFunctionOperator` - Lambda InvokeFunction operations

AWS Step Functions

- **StepFunctionGetExecutionOutputOperator** - `airflow.providers.amazon.aws.operators.step_function.StepFunctionGetExecutionOutputOperator` - Step Function GetExecutionOutput operations
- **StepFunctionStartExecutionOperator** - `airflow.providers.amazon.aws.operators.step_function.StepFunctionStartExecutionOperator` - Step Function StartExecution operations

Amazon AppFlow

- **AppflowBaseOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowBaseOperator` - AppflowBase operations

- **AppflowRecordsShortCircuitOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRecordsShortCircuitOperator` - AppflowRecordsShortCircuit operations
- **AppflowRunAfterOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRunAfterOperator` - AppflowRunAfter operations
- **AppflowRunBeforeOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRunBeforeOperator` - AppflowRunBefore operations
- **AppflowRunDailyOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRunDailyOperator` - AppflowRunDaily operations
- **AppflowRunFullOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRunFullOperator` - AppflowRunFull operations
- **AppflowRunOperator** - `airflow.providers.amazon.aws.operators.appflow.AppflowRunOperator` - AppflowRun operations

Amazon Athena

- **AthenaOperator** - `airflow.providers.amazon.aws.operators.athena.AthenaOperator` - Athena operations

Amazon Bedrock

- **BedrockCreateDataSourceOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockCreateDataSourceOperator` - BedrockCreateDataSource operations
- **BedrockCreateKnowledgeBaseOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockCreateKnowledgeBaseOperator` - BedrockCreateKnowledgeBase operations
- **BedrockCreateProvisionedModelThroughputOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockCreateProvisionedModelThroughputOperator` - BedrockCreateProvisionedModelThroughput operations
- **BedrockCustomizeModelOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockCustomizeModelOperator` - BedrockCustomizeModel operations
- **BedrockIngestDataOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockIngestDataOperator` - BedrockIngestData operations
- **BedrockInvokeModelOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockInvokeModelOperator` - BedrockInvokeModel operations
- **BedrockRaGOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockRaGOperator` - BedrockRaG operations
- **BedrockRetrieveOperator** - `airflow.providers.amazon.aws.operators.bedrock.BedrockRetrieveOperator` - BedrockRetrieve operations

Amazon Comprehend

- **ComprehendBaseOperator** - `airflow.providers.amazon.aws.operators.comprehend.ComprehendBaseOperator` - ComprehendBase operations
- **ComprehendCreateDocumentClassifierOperator** - `airflow.providers.amazon.aws.operators.comprehend.ComprehendCreateDocumentClassifierOperator` - ComprehendCreateDocumentClassifier operations

- **ComprehendStartPiiEntitiesDetectionJobOperator** - `airflow.providers.amazon.aws.operators.comprehend.ComprehendStartPiiEntitiesDetectionJobOperator` - ComprehendStartPiiEntitiesDetectionJob operations

Amazon EC2

- **EC2CreateInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2CreateInstanceOperator` - EC2 CreateInstance operations
- **EC2HibernateInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2HibernateInstanceOperator` - EC2 HibernateInstance operations
- **EC2RebootInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2RebootInstanceOperator` - EC2 RebootInstance operations
- **EC2StartInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2StartInstanceOperator` - EC2 StartInstance operations
- **EC2StopInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2StopInstanceOperator` - EC2 StopInstance operations
- **EC2TerminateInstanceOperator** - `airflow.providers.amazon.aws.operators.ec2.EC2TerminateInstanceOperator` - EC2 TerminateInstance operations

Amazon ECS

- **EcsBaseOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsBaseOperator` - ECS Base operations
- **EcsCreateClusterOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsCreateClusterOperator` - ECS CreateCluster operations
- **EcsDeleteClusterOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsDeleteClusterOperator` - ECS DeleteCluster operations
- **EcsDeregisterTaskDefinitionOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsDeregisterTaskDefinitionOperator` - ECS DeregisterTaskDefinition operations
- **EcsRegisterTaskDefinitionOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsRegisterTaskDefinitionOperator` - ECS RegisterTaskDefinition operations
- **EcsRunTaskOperator** - `airflow.providers.amazon.aws.operators.ecs.EcsRunTaskOperator` - ECS RunTask operations

Amazon EKS

- **EksCreateClusterOperator** - `airflow.providers.amazon.aws.operators.eks.EksCreateClusterOperator` - EKS CreateCluster operations
- **EksCreateFargateProfileOperator** - `airflow.providers.amazon.aws.operators.eks.EksCreateFargateProfileOperator` - EKS CreateFargateProfile operations
- **EksCreateNodegroupOperator** - `airflow.providers.amazon.aws.operators.eks.EksCreateNodegroupOperator` - EKS CreateNodegroup operations
- **EksDeleteClusterOperator** - `airflow.providers.amazon.aws.operators.eks.EksDeleteClusterOperator` - EKS DeleteCluster operations

- **EksDeleteFargateProfileOperator** - `airflow.providers.amazon.aws.operators.eks.EksDeleteFargateProfileOperator` - EKS DeleteFargateProfile operations
- **EksDeleteNodegroupOperator** - `airflow.providers.amazon.aws.operators.eks.EksDeleteNodegroupOperator` - EKS DeleteNodegroup operations
- **EksPodOperator** - `airflow.providers.amazon.aws.operators.eks.EksPodOperator` - EKS Pod operations

Amazon EMR

- **EmrAddStepsOperator** - `airflow.providers.amazon.aws.operators.emr.EmrAddStepsOperator` - EMR AddSteps operations
- **EmrContainerOperator** - `airflow.providers.amazon.aws.operators.emr.EmrContainerOperator` - EMR Container operations
- **EmrCreateJobFlowOperator** - `airflow.providers.amazon.aws.operators.emr.EmrCreateJobFlowOperator` - EMR CreateJobFlow operations
- **EmrEksCreateClusterOperator** - `airflow.providers.amazon.aws.operators.emr.EmrEksCreateClusterOperator` - EMR EKS CreateCluster operations
- **EmrModifyClusterOperator** - `airflow.providers.amazon.aws.operators.emr.EmrModifyClusterOperator` - EMR ModifyCluster operations
- **EmrServerlessCreateApplicationOperator** - `airflow.providers.amazon.aws.operators.emr.EmrServerlessCreateApplicationOperator` - EMR ServerlessCreateApplication operations
- **EmrServerlessDeleteApplicationOperator** - `airflow.providers.amazon.aws.operators.emr.EmrServerlessDeleteApplicationOperator` - EMR ServerlessDeleteApplication operations
- **EmrServerlessStartJobOperator** - `airflow.providers.amazon.aws.operators.emr.EmrServerlessStartJobOperator` - EMR ServerlessStartJob operations
- **EmrServerlessStopApplicationOperator** - `airflow.providers.amazon.aws.operators.emr.EmrServerlessStopApplicationOperator` - EMR ServerlessStopApplication operations
- **EmrStartNotebookExecutionOperator** - `airflow.providers.amazon.aws.operators.emr.EmrStartNotebookExecutionOperator` - EMR StartNotebookExecution operations
- **EmrStopNotebookExecutionOperator** - `airflow.providers.amazon.aws.operators.emr.EmrStopNotebookExecutionOperator` - EMR StopNotebookExecution operations
- **EmrTerminateJobFlowOperator** - `airflow.providers.amazon.aws.operators.emr.EmrTerminateJobFlowOperator` - EMR TerminateJobFlow operations

Amazon EventBridge

- **EventBridgeDisableRuleOperator** - `airflow.providers.amazon.aws.operators.eventbridge.EventBridgeDisableRuleOperator` - EventBridgeDisableRule operations
- **EventBridgeEnableRuleOperator** - `airflow.providers.amazon.aws.operators.eventbridge.EventBridgeEnableRuleOperator` - EventBridgeEnableRule operations
- **EventBridgePutEventsOperator** - `airflow.providers.amazon.aws.operators.eventbridge.EventBridgePutEventsOperator` - EventBridgePutEvents operations
- **EventBridgePutRuleOperator** - `airflow.providers.amazon.aws.operators.eventbridge.EventBridgePutRuleOperator` - EventBridgePutRule operations

Amazon Glacier

- **GlacierCreateJobOperator** - `airflow.providers.amazon.aws.operators.glacier.GlacierCreateJobOperator` - GlacierCreateJob operations
- **GlacierUploadArchiveOperator** - `airflow.providers.amazon.aws.operators.glacier.GlacierUploadArchiveOperator` - GlacierUploadArchive operations

Amazon Kinesis Analytics

- **KinesisAnalyticsV2CreateApplicationOperator** - `airflow.providers.amazon.aws.operators.kinesis.KinesisAnalyticsV2CreateApplicationOperator` - KinesisAnalyticsV2CreateApplication operations
- **KinesisAnalyticsV2StartApplicationOperator** - `airflow.providers.amazon.aws.operators.kinesis.KinesisAnalyticsV2StartApplicationOperator` - KinesisAnalyticsV2StartApplication operations
- **KinesisAnalyticsV2StopApplicationOperator** - `airflow.providers.amazon.aws.operators.kinesis.KinesisAnalyticsV2StopApplicationOperator` - KinesisAnalyticsV2StopApplication operations

Amazon Neptune

- **NeptuneStartDbClusterOperator** - `airflow.providers.amazon.aws.operators.neptune.NeptuneStartDbClusterOperator` - NeptuneStartDbCluster operations
- **NeptuneStopDbClusterOperator** - `airflow.providers.amazon.aws.operators.neptune.NeptuneStopDbClusterOperator` - NeptuneStopDbCluster operations

Amazon QuickSight

- **QuickSightCreateIngestionOperator** - `airflow.providers.amazon.aws.operators.quicksight.QuickSightCreateIngestionOperator` - QuickSightCreateIngestion operations

Amazon RDS

- **RdsBaseOperator** - `airflow.providers.amazon.aws.operators.rds.RdsBaseOperator` - RdsBase operations
- **RdsCancelExportTaskOperator** - `airflow.providers.amazon.aws.operators.rds.RdsCancelExportTaskOperator` - RdsCancelExportTask operations
- **RdsCopyDbSnapshotOperator** - `airflow.providers.amazon.aws.operators.rds.RdsCopyDbSnapshotOperator` - RdsCopyDbSnapshot operations
- **RdsCreateDbInstanceOperator** - `airflow.providers.amazon.aws.operators.rds.RdsCreateDbInstanceOperator` - RdsCreateDbInstance operations
- **RdsCreateDbSnapshotOperator** - `airflow.providers.amazon.aws.operators.rds.RdsCreateDbSnapshotOperator` - RdsCreateDbSnapshot operations
- **RdsCreateEventSubscriptionOperator** - `airflow.providers.amazon.aws.operators.rds.RdsCreateEventSubscriptionOperator` - RdsCreateEventSubscription operations
- **RdsDeleteDbInstanceOperator** - `airflow.providers.amazon.aws.operators.rds.RdsDeleteDbInstanceOperator` - RdsDeleteDbInstance operations
- **RdsDeleteDbSnapshotOperator** - `airflow.providers.amazon.aws.operators.rds.RdsDeleteDbSnapshotOpera`

tor - RdsDeleteDbSnapshot operations
- **RdsDeleteEventSubscriptionOperator** - airflow.providers.amazon.aws.operators.rds.RdsDeleteEventSubscriptionOperator - RdsDeleteEventSubscription operations
- **RdsStartDbOperator** - airflow.providers.amazon.aws.operators.rds.RdsStartDbOperator - RdsStartDb operations
- **RdsStartExportTaskOperator** - airflow.providers.amazon.aws.operators.rds.RdsStartExportTaskOperator - RdsStartExportTask operations
- **RdsStopDbOperator** - airflow.providers.amazon.aws.operators.rds.RdsStopDbOperator - RdsStopDb operations

Amazon Redshift

- **RedshiftCreateClusterOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftCreateClusterOperator - Redshift CreateCluster operations
- **RedshiftCreateClusterSnapshotOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftCreateClusterSnapshotOperator - Redshift CreateClusterSnapshot operations
- **RedshiftDataOperator** - airflow.providers.amazon.aws.operators.redshift_data.RedshiftDataOperator - Redshift Data operations
- **RedshiftDeleteClusterOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftDeleteClusterOperator - Redshift DeleteCluster operations
- **RedshiftDeleteClusterSnapshotOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftDeleteClusterSnapshotOperator - Redshift DeleteClusterSnapshot operations
- **RedshiftPauseClusterOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftPauseClusterOperator - Redshift PauseCluster operations
- **RedshiftResumeClusterOperator** - airflow.providers.amazon.aws.operators.redshift_cluster.RedshiftResumeClusterOperator - Redshift ResumeCluster operations

Amazon S3

- **S3CopyObjectOperator** - airflow.providers.amazon.aws.operators.s3.S3CopyObjectOperator - S3 CopyObject operations
- **S3CreateBucketOperator** - airflow.providers.amazon.aws.operators.s3.S3CreateBucketOperator - S3 CreateBucket operations
- **S3CreateObjectOperator** - airflow.providers.amazon.aws.operators.s3.S3CreateObjectOperator - S3 CreateObject operations
- **S3DeleteBucketOperator** - airflow.providers.amazon.aws.operators.s3.S3DeleteBucketOperator - S3 DeleteBucket operations
- **S3DeleteBucketTaggingOperator** - airflow.providers.amazon.aws.operators.s3.S3DeleteBucketTaggingOperator - S3 DeleteBucketTagging operations
- **S3DeleteObjectsOperator** - airflow.providers.amazon.aws.operators.s3.S3DeleteObjectsOperator - S3 DeleteObjects operations
- **S3GetBucketTaggingOperator** - airflow.providers.amazon.aws.operators.s3.S3GetBucketTaggingOperator - S3 GetBucketTagging operations
- **S3ListOperator** - airflow.providers.amazon.aws.operators.s3.S3ListOperator - S3 List

operations
- **S3ListPrefixesOperator** - `airflow.providers.amazon.aws.operators.s3.S3ListPrefixesOperator` - S3 ListPrefixes operations
- **S3PutBucketTaggingOperator** - `airflow.providers.amazon.aws.operators.s3.S3PutBucketTaggingOperator` - S3 PutBucketTagging operations

Amazon SNS

- **SnsPublishOperator** - `airflow.providers.amazon.aws.operators.sns.SnsPublishOperator` - SNS Publish operations

Amazon SQS

- **SqsPublishOperator** - `airflow.providers.amazon.aws.operators.sqs.SqsPublishOperator` - SQS Publish operations

Amazon SageMaker

- **SageMakerAutoMLOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerAutoMLOperator` - SageMaker AutoML operations
- **SageMakerBaseOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerBaseOperator` - SageMaker Base operations
- **SageMakerCreateExperimentOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerCreateExperimentOperator` - SageMaker CreateExperiment operations
- **SageMakerCreateNotebookOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerCreateNotebookOperator` - SageMaker CreateNotebook operations
- **SageMakerDeleteModelOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerDeleteModelOperator` - SageMaker DeleteModel operations
- **SageMakerDeleteNotebookOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerDeleteNotebookOperator` - SageMaker DeleteNotebook operations
- **SageMakerEndpointConfigOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerEndpointConfigOperator` - SageMaker EndpointConfig operations
- **SageMakerEndpointOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerEndpointOperator` - SageMaker Endpoint operations
- **SageMakerModelOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerModelOperator` - SageMaker Model operations
- **SageMakerNotebookOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerNotebookOperator` - SageMaker Notebook operations
- **SageMakerProcessingOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerProcessingOperator` - SageMaker Processing operations
- **SageMakerRegisterModelVersionOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerRegisterModelVersionOperator` - SageMaker RegisterModelVersion operations
- **SageMakerStartNoteBookOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerStartNoteBookOperator` - SageMaker StartNoteBook operations

- **SageMakerStartPipelineOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerStartPipelineOperator` - SageMaker StartPipeline operations
- **SageMakerStopNotebookOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerStopNotebookOperator` - SageMaker StopNotebook operations
- **SageMakerStopPipelineOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerStopPipelineOperator` - SageMaker StopPipeline operations
- **SageMakerTrainingOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerTrainingOperator` - SageMaker Training operations
- **SageMakerTransformOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerTransformOperator` - SageMaker Transform operations
- **SageMakerTuningOperator** - `airflow.providers.amazon.aws.operators.sagemaker.SageMakerTuningOperator` - SageMaker Tuning operations
- **SageMakerUnifiedStudioOperator** - `airflow.providers.amazon.aws.operators.sagemaker_unified_studio.SageMakerUnifiedStudioOperator` - SageMaker Unified Studio operations

Sensors

AWS Batch

- **BatchComputeEnvironmentSensor** - `airflow.providers.amazon.aws.sensors.batch.BatchComputeEnvironmentSensor` - Monitor BatchComputeEnvironment operations
- **BatchJobQueueSensor** - `airflow.providers.amazon.aws.sensors.batch.BatchJobQueueSensor` - Monitor BatchJobQueue operations
- **BatchSensor** - `airflow.providers.amazon.aws.sensors.batch.BatchSensor` - Monitor Batch operations

AWS CloudFormation

- **CloudFormationCreateStackSensor** - `airflow.providers.amazon.aws.sensors.cloudformation.CloudFormationCreateStackSensor` - Monitor CloudFormationCreateStack operations
- **CloudFormationDeleteStackSensor** - `airflow.providers.amazon.aws.sensors.cloudformation.CloudFormationDeleteStackSensor` - Monitor CloudFormationDeleteStack operations

AWS DMS

- **DmsTaskBaseSensor** - `airflow.providers.amazon.aws.sensors.dms.DmsTaskBaseSensor` - Monitor DmsTaskBase operations
- **DmsTaskCompletedSensor** - `airflow.providers.amazon.aws.sensors.dms.DmsTaskCompletedSensor` - Monitor DmsTaskCompleted operations

AWS Glue

- **GlueCatalogPartitionSensor** - `airflow.providers.amazon.aws.sensors.glue.GlueCatalogPartitionSensor` - Monitor Glue CatalogPartition operations
- **GlueCrawlerSensor** - `airflow.providers.amazon.aws.sensors.glue.GlueCrawlerSensor` -

Monitor Glue Crawler operations
- **GlueDataQualityRuleRecommendationRunSensor** - `airflow.providers.amazon.aws.sensors.glue.GlueDataQualityRuleRecommendationRunSensor` - Monitor Glue DataQualityRuleRecommendationRun operations
- **GlueDataQualityRuleSetEvaluationRunSensor** - `airflow.providers.amazon.aws.sensors.glue.GlueDataQualityRuleSetEvaluationRunSensor` - Monitor Glue DataQualityRuleSetEvaluationRun operations
- **GlueJobSensor** - `airflow.providers.amazon.aws.sensors.glue.GlueJobSensor` - Monitor Glue Job operations

AWS Lambda

- **LambdaFunctionStateSensor** - `airflow.providers.amazon.aws.sensors.lambda_function.LambdaFunctionStateSensor` - Monitor Lambda FunctionState operations

AWS Step Functions

- **StepFunctionExecutionSensor** - `airflow.providers.amazon.aws.sensors.step_function.StepFunctionExecutionSensor` - Monitor Step Function Execution operations

Amazon Athena

- **AthenaSensor** - `airflow.providers.amazon.aws.sensors.athena.AthenaSensor` - Monitor Athena operations

Amazon Bedrock

- **BedrockBaseSensor** - `airflow.providers.amazon.aws.sensors.bedrock.BedrockBaseSensor` - Monitor BedrockBase operations
- **BedrockCustomizeModelCompletedSensor** - `airflow.providers.amazon.aws.sensors.bedrock.BedrockCustomizeModelCompletedSensor` - Monitor BedrockCustomizeModelCompleted operations
- **BedrockIngestionJobSensor** - `airflow.providers.amazon.aws.sensors.bedrock.BedrockIngestionJobSensor` - Monitor BedrockIngestionJob operations
- **BedrockKnowledgeBaseActiveSensor** - `airflow.providers.amazon.aws.sensors.bedrock.BedrockKnowledgeBaseActiveSensor` - Monitor BedrockKnowledgeBaseActive operations
- **BedrockProvisionModelThroughputCompletedSensor** - `airflow.providers.amazon.aws.sensors.bedrock.BedrockProvisionModelThroughputCompletedSensor` - Monitor BedrockProvisionModelThroughputCompleted operations

Amazon Comprehend

- **ComprehendBaseSensor** - `airflow.providers.amazon.aws.sensors.comprehend.ComprehendBaseSensor` - Monitor ComprehendBase operations
- **ComprehendCreateDocumentClassifierCompletedSensor** - `airflow.providers.amazon.aws.sensors.comprehend.ComprehendCreateDocumentClassifierCompletedSensor` - Monitor ComprehendCreateDocumentClassifierCompleted operations
- **ComprehendStartPiiEntitiesDetectionJobCompletedSensor** - `airflow.providers.amazon.aws.sensors.comprehend.ComprehendStartPiiE`

ntitiesDetectionJobCompletedSensor - Monitor ComprehendStartPiiEntitiesDetectionJobCompleted operations

Amazon EC2

- **EC2InstanceStateSensor** - airflow.providers.amazon.aws.sensors.ec2.EC2InstanceStateSensor - Monitor EC2 InstanceState operations

Amazon ECS

- **EcsBaseSensor** - airflow.providers.amazon.aws.sensors.ecs.EcsBaseSensor - Monitor ECS Base operations
- **EcsClusterStateSensor** - airflow.providers.amazon.aws.sensors.ecs.EcsClusterStateSensor - Monitor ECS ClusterState operations
- **EcsTaskDefinitionStateSensor** - airflow.providers.amazon.aws.sensors.ecs.EcsTaskDefinitionStateSensor - Monitor ECS TaskDefinitionState operations
- **EcsTaskStateSensor** - airflow.providers.amazon.aws.sensors.ecs.EcsTaskStateSensor - Monitor ECS TaskState operations

Amazon EKS

- **EksBaseSensor** - airflow.providers.amazon.aws.sensors.eks.EksBaseSensor - Monitor EKS Base operations
- **EksClusterStateSensor** - airflow.providers.amazon.aws.sensors.eks.EksClusterStateSensor - Monitor EKS ClusterState operations
- **EksFargateProfileStateSensor** - airflow.providers.amazon.aws.sensors.eks.EksFargateProfileStateSensor - Monitor EKS FargateProfileState operations
- **EksNodegroupStateSensor** - airflow.providers.amazon.aws.sensors.eks.EksNodegroupStateSensor - Monitor EKS NodegroupState operations

Amazon EMR

- **EmrBaseSensor** - airflow.providers.amazon.aws.sensors.emr.EmrBaseSensor - Monitor EMR Base operations
- **EmrContainerSensor** - airflow.providers.amazon.aws.sensors.emr.EmrContainerSensor - Monitor EMR Container operations
- **EmrJobFlowSensor** - airflow.providers.amazon.aws.sensors.emr.EmrJobFlowSensor - Monitor EMR JobFlow operations
- **EmrNotebookExecutionSensor** - airflow.providers.amazon.aws.sensors.emr.EmrNotebookExecutionSensor - Monitor EMR NotebookExecution operations
- **EmrServerlessApplicationSensor** - airflow.providers.amazon.aws.sensors.emr.EmrServerlessApplicationSensor - Monitor EMR ServerlessApplication operations
- **EmrServerlessJobSensor** - airflow.providers.amazon.aws.sensors.emr.EmrServerlessJobSensor - Monitor EMR ServerlessJob operations
- **EmrStepSensor** - airflow.providers.amazon.aws.sensors.emr.EmrStepSensor - Monitor EMR

Step operations

Amazon Glacier

- **GlacierJobOperationSensor** - `airflow.providers.amazon.aws.sensors.glacier.GlacierJobOperationSensor` - Monitor GlacierJobOperation operations

Amazon Kinesis Analytics

- **KinesisAnalyticsV2BaseSensor** - `airflow.providers.amazon.aws.sensors.kinesis.KinesisAnalyticsV2BaseSensor` - Monitor KinesisAnalyticsV2Base operations
- **KinesisAnalyticsV2StartApplicationCompletedSensor** - `airflow.providers.amazon.aws.sensors.kinesis.KinesisAnalyticsV2StartApplicationCompletedSensor` - Monitor KinesisAnalyticsV2StartApplicationCompleted operations
- **KinesisAnalyticsV2StopApplicationCompletedSensor** - `airflow.providers.amazon.aws.sensors.kinesis.KinesisAnalyticsV2StopApplicationCompletedSensor` - Monitor KinesisAnalyticsV2StopApplicationCompleted operations

Amazon QuickSight

- **QuickSightSensor** - `airflow.providers.amazon.aws.sensors.quicksight.QuickSightSensor` - Monitor QuickSight operations

Amazon RDS

- **RdsBaseSensor** - `airflow.providers.amazon.aws.sensors.rds.RdsBaseSensor` - Monitor RdsBase operations
- **RdsDbSensor** - `airflow.providers.amazon.aws.sensors.rds.RdsDbSensor` - Monitor RdsDb operations
- **RdsExportTaskExistenceSensor** - `airflow.providers.amazon.aws.sensors.rds.RdsExportTaskExistenceSensor` - Monitor RdsExportTaskExistence operations
- **RdsSnapshotExistenceSensor** - `airflow.providers.amazon.aws.sensors.rds.RdsSnapshotExistenceSensor` - Monitor RdsSnapshotExistence operations

Amazon Redshift

- **RedshiftClusterSensor** - `airflow.providers.amazon.aws.sensors.redshift_cluster.RedshiftClusterSensor` - Monitor Redshift Cluster operations

Amazon S3

- **S3KeySensor** - `airflow.providers.amazon.aws.sensors.s3.S3KeySensor` - Monitor S3 Key operations
- **S3KeysUnchangedSensor** - `airflow.providers.amazon.aws.sensors.s3.S3KeysUnchangedSensor` - Monitor S3 KeysUnchanged operations

Amazon SQS

- **SqsSensor** - `airflow.providers.amazon.aws.sensors.sqs.SqsSensor` - Monitor SQS operations

Amazon SageMaker

- **SageMakerAutoMLSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerAutoMLSensor` - Monitor SageMaker AutoML operations
- **SageMakerBaseSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerBaseSensor` - Monitor SageMaker Base operations
- **SageMakerEndpointSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerEndpointSensor` - Monitor SageMaker Endpoint operations
- **SageMakerNotebookSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerNotebookSensor` - Monitor SageMaker Notebook operations
- **SageMakerPipelineSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerPipelineSensor` - Monitor SageMaker Pipeline operations
- **SageMakerProcessingSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerProcessingSensor` - Monitor SageMaker Processing operations
- **SageMakerTrainingSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerTrainingSensor` - Monitor SageMaker Training operations
- **SageMakerTransformSensor** - `airflow.providers.amazon.aws.sensors.sagemaker.SageMakerTransformSensor` - Monitor SageMaker Transform operations

Other AWS Services

- **AwsBaseSensor** - `airflow.providers.amazon.aws.sensors.unknown.AwsBaseSensor` - Monitor AwsBase operations
- **DynamoDBValueSensor** - `airflow.providers.amazon.aws.sensors.unknown.DynamoDBValueSensor` - Monitor DynamoDBValue operations
- **OpenSearchServerlessCollectionActiveSensor** - `airflow.providers.amazon.aws.sensors.unknown.OpenSearchServerlessCollectionActiveSensor` - Monitor OpenSearchServerlessCollectionActive operations

Usage Example

```
from airflow import DAG
from airflow.providers.amazon.aws.operators.sagemaker import
        SageMakerTrainingOperator
from airflow.providers.amazon.aws.operators.s3 import
        S3CopyObjectOperator

dag = DAG('ml_pipeline', schedule_interval='@daily')

train_model = SageMakerTrainingOperator(
    task_id='train_model',
    config={...},
    dag=dag
)
```

```
copy_results = S3CopyObjectOperator(
    task_id='copy_results',
    source_bucket_key='model/output',
    dest_bucket_key='production/model',
    dag=dag
)

train_model >> copy_results
```

Resources

- Apache Airflow AWS Provider Documentation
- SageMaker Unified Studio User Guide