OS Migrate Documentation

Table of Contents

1. Welcome to OS Migrate

OS Migrate provides a framework and toolsuite for exporting and importing resources between two clouds. It's a collection of Ansible playbooks that provide the basic functionality, but may not fit each use case out of the box. You can craft custom playbooks using the OS Migrate collection pieces (roles and modules) as building blocks.

At present OS Migrate supports migration from VMware clouds to OpenStack, and OpenStack to OpenStack.

OS Migrate strictly uses the official OpenStack API and does not utilize direct database access or other methods to export or import data. The Ansible playbooks contained in OS Migrate are idempotent. If a command fails, you can retry with the same command.

2. VMware to OpenStack Guide

An important function included in OS Migrate is our VMware tooling, which allows you to migrade a virtual machine from an ESXi/Vcenter environment to OpenStack environments.

The code used os-migrate Ansible collection in order to deploy conversion host and setup correctly the prerequists in the Openstack destination cloud. It also used the vmware community collection in order to gather informations from the source VMWare environment.

The Ansible collection provides different steps to scale your migration from VMWare to Openstack:

- A discovery phase where it analizes the VMWare source environment and provides collected data to help for the migration.
- A pre-migration phase where it make sure the destionation cloud is ready to perform the migration, by creating the conversion host for example or the required network if needed.
- A migration phase with different workflow where the user can basicaly scale the migration with a very number of virtual machines as entry point, or can migrate sensitive virtual machine by using a near zero down time with the change block tracking VMWare option (CBT) and so perform the virtual machine migration in two steps. The migration can also be done without conversion host.

2.1. Workflow

There is different ways to run the migration from VMWare to OpenStack.

- The default is by using nbdkit server with a conversion host (an Openstack instance hosted in the destination cloud). This way allow the user to use the CBT option and approach a zero downtime. It can also run the migration in one time cycle.
- The second one by using virt-v2v binding with a conversion host. Here you can use a conversion host (Openstack instance) already deployed or you can let OS-Migrate deployed a conversion host for you.
- A third way is available where you can skip the conversion host and perform the migration on a Linux machine, the volume migrated and converted will be upload a Glance image or can be use later as a Cinder volume. This way is not recommended if you have big disk or a huge amount of VMs to migrate: the performance are really slower than with the other ways.

All of these are configurable with Ansible boolean variables.

2.2. Features and supported OS

2.2.1. Features

The following features are availables:

- Discovery mode
- Network mapping
- Port creation and mac addresses mapping
- Openstack flavor mapping and creation
- Migration with nbdkit server with change block tracking feature (CBT)
- Migration with virt-v2v
- Upload migrate volume via Glance
- Multi disks migration
- Multi nics
- Parallel migration on a same conversion host
- Ansible Automation Platform (AAP)

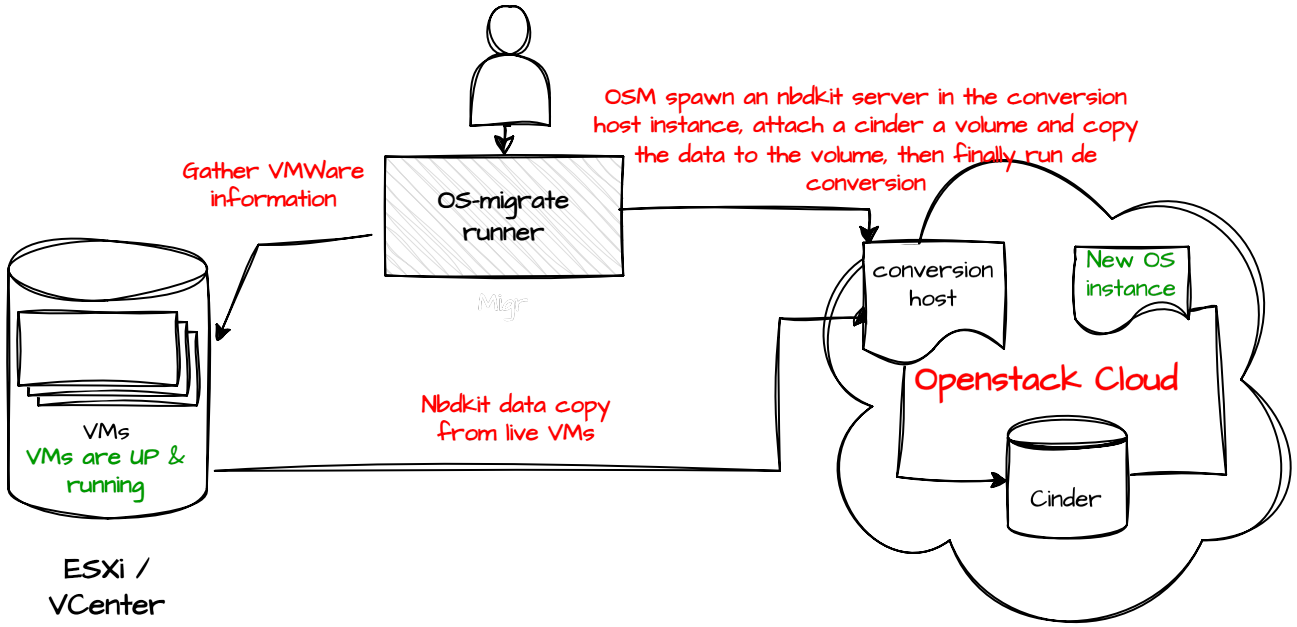
2.2.2. Supported OS

Currently we are supporting the following matrice:

OS Family	Version	Supported & Tested	Not Tested Yet
RHEL	9.4	Yes	-
RHEL	9.3 and lower	Yes	-

OS Family	Version	Supported & Tested	Not Tested Yet
RHEL	8.5	Yes	-
RHEL	8.4 and lower	-	Yes
CentOS	9	Yes	-
CentOS	8	Yes	-
Ubuntu Server	24	Yes	-
Windows	10	Yes	-
Windows Server	2k22	Yes	-
Suse	X	Yes	-

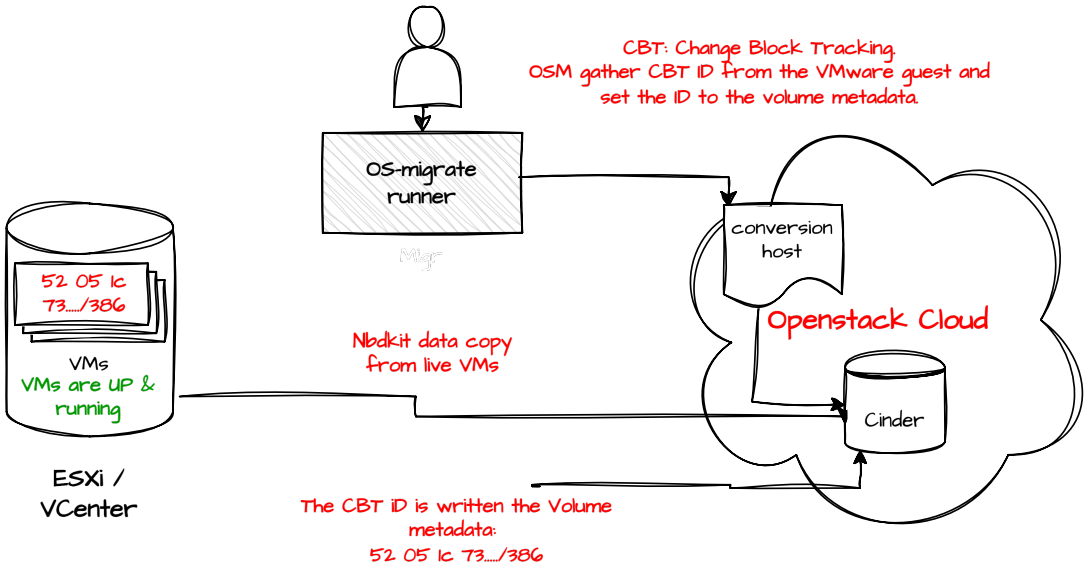
2.2.3. Nbdkit migration example



2.2.4. Nbdkit migration example with the Change Block Tracking

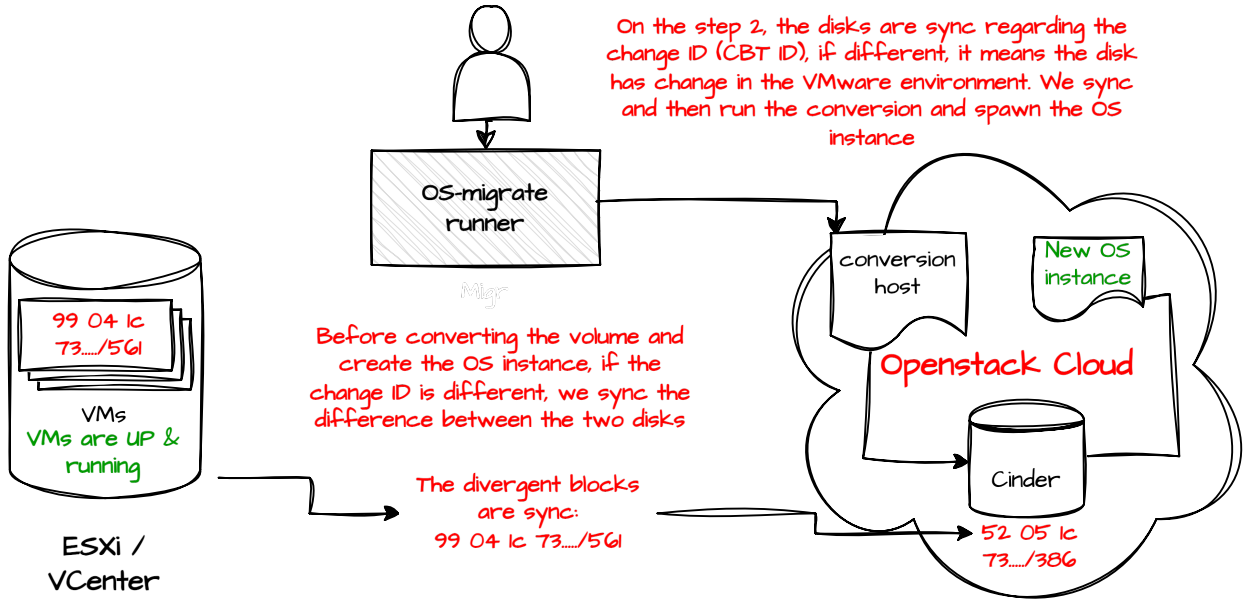
Step 1: The data are copied and the change ID from the VMware disk are set to the Cinder volume as metadata

The conversion cannot be made at this moment, and the OS instance is not created. This functionality can be used for large disks with a lot of data to transfer. It helps avoid a prolonged service interruption.



==== Step 2: OSM compare the source (VMware disk) and the destination (Openstack Volume) change ID

If the change IDs are not equal, the changed blocks between the source and destination are synced. Then, the conversion to libvirt/KVM is triggered, and the OpenStack instance is created. This allows for minimal downtime for the VMs.



=== Migration demo from an AEE

The content of the Ansible Execution Environment could be find here:

https://github.com/os-migrate/aap/blob/main/aae-container-file

And the live demo here:

Migration from VMware to OpenStack

=== Running migration

==== Conversion host

You can use os_migrate.os_migration collection to deploy a conversion, but you can easily create your conversion host manually.

A conversion host is basically an OpenStack instance.

Important: If you want to take benefit of the current supported OS, it's highly recommended to use a **CentOS-10** release or **RHEL-9.5** and superior. If you want to use other Linux distribution, make sure the virtio-win package is equal or higher than 1.40 version.

```
curl -O -k https://cloud.centos.org/centos/10-stream/x86_64/images/CentOS-Stream-
GenericCloud-10-20250217.0.x86_64.qcow2

# Create OpenStack image:
openstack image create --disk-format qcow2 --file CentOS-Stream-GenericCloud-10-
20250217.0.x86_64.qcow2 CentOS-Stream-GenericCloud-10-20250217.0.x86_64.qcow2

# Create flavor, security group and network if needed
openstack server create --flavor x.medium --image 14b1a895-5003-4396-888e-1fa55cd4adf8
\
  --key-name default --network private   vmware-conv-host
openstack server add floating ip vmware-conv-host 192.168.18.205
```

Inventory, Variables files and Ansible command:

inventory.yml

```
migrator:
  hosts:
    localhost:
      ansible_connection: local
      ansible_python_interpreter: "{{ ansible_playbook_python }}"
conversion_host:
  hosts:
    192.168.18.205:
      ansible_ssh_user: cloud-user
      ansible_ssh_private_key_file: key
```

myvars.yml:

```
# if you run the migration from an Ansible Execution Environment (AEE)
# set this to true:
runner_from_aee: true
```

```
# osm working directory:
os_migrate_vmw_data_dir: /opt/os-migrate
copy_openstack_credentials_to_conv_host: false

# Re-use an already deployed conversion host:
already_deploy_conversion_host: true

# If no mapped network then set the openstack network:
openstack_private_network: 81cc01d2-5e47-4fad-b387-32686ec71fa4

# Security groups for the instance:
security_groups: ab7e2b1a-b9d3-4d31-9d2a-bab63f823243
use_existing_flavor: true
# key pair name, could be left blank
ssh_key_name: default
# network settings for openstack:
os_migrate_create_network_port: true
copy_metadata_to_conv_host: true
used_mapped_networks: false

vms_list:
  - rhel-9.4-1
```

secrets.yml:

```
# VMware parameters:
esxi_hostname: 10.0.0.7
vcenter_hostname: 10.0.0.7
vcenter_username: root
vcenter_password: root
vcenter_datacenter: Datacenter

os_cloud_environ: psi-rhos-upgrades-ci
dst_cloud:
  auth:
    auth_url: https://keystone-public-openstack.apps.ocp-4-16.standalone
    username: admin
    project_id: xyz
    project_name: admin
    user_domain_name: Default
    password: openstack
  region_name: regionOne
  interface: public
  insecure: true
  identity_api_version: 3
```

Ansible command:

```
ansible-playbook -i inventory.yml os_migrate.vmware_migration_kit.migration -e
```

```
@secrets.yml -e @myvars.yml
```

2.3. Usage

You can find a "how to" here, to start from sratch with a container: https://gist.github.com/matbu/003c300fd99ebfbf383729c249e9956f

Clone repository or install from ansible galaxy

```
git clone https://github.com/os-migrate/vmware-migration-kit
ansible-galaxy collection install os_migrate.vmware_migration_kit
```

2.3.1. Nbdkit (default)

Edit vars.yaml file and add our own setting:

```
esxi_hostname: ********
vcenter_hostname: *******
vcenter_username: root
vcenter_password: *****
vcenter_datacenter: Datacenter
```

If you already have a conversion host, or if you want to re-used a previously deployed one:

```
already_deploy_conversion_host: true
```

Then specify the Openstack credentials:

```
# OpenStack destination cloud auth parameters:
dst_cloud:
  auth:
    auth_url: https://openstack.dst.cloud:13000/v3
    username: tenant
    project_id: xyz
    project_name: migration
    user_domain_name: osm.com
    password: password
  region_name: regionOne
  interface: public
  identity_api_version: 3

# OpenStack migration parameters:
# Use mapped networks or not:
used_mapped_networks: true
network_map:
```

```
  VM Network: private

# If no mapped network then set the openstack network:
openstack_private_network: 81cc01d2-5e47-4fad-b387-32686ec71fa4

# Security groups for the instance:
security_groups: 4f077e64-bdf6-4d2a-9f2c-c5588f4948ce
use_existing_flavor: true

os_migrate_create_network_port: false

# OS-migrate parameters:
# osm working directory:
os_migrate_vmw_data_dir: /opt/os-migrate

# Set this to true if the Openstack "dst_cloud" is a clouds.yaml file
# other, if the dest_cloud is a dict of authentication parameters, set
# this to false:
copy_openstack_credentials_to_conv_host: false

# Teardown
# Set to true if you want osm to delete everything on the destination cloud.
os_migrate_tear_down: true

# VMs list
vms_list:
  - rhel-1
  - rhel-2
```

2.3.2. Running migration from local shared NFS

OS-Migrate can migrate directly from a local shared directory mounted on the conversion host. If the VMware virtual machines are located on an NFS datastore that is accessible to the conversion host, you can mount the NFS storage on the conversion host and provide the path to the NFS mount point.

OS-Migrate will then directly consume the disks of the virtual machines located on the NFS mount point. Configure the Ansible variable to specify your mount point as follows:

```
import_workloads_local_disk_path: "/srv/nfs"
```

In this mode, only cold migration is supported.

2.3.3. Ansible configuration

Create an invenvoty file, and replace the conv_host_ip by the ip address of your conversion host:

```
migrator:
```

```
  hosts:
    localhost:
      ansible_connection: local
      ansible_python_interpreter: "{{ ansible_playbook_python }}"
conversion_host:
  hosts:
    conv_host_ip:
      ansible_ssh_user: cloud-user
      ansible_ssh_private_key_file: /home/stack/.ssh/conv-host
```

Then run the migration with:

```
ansible-playbook -i localhost_inventory.yml os_migrate.vmware_migration_kit.migration
-e @vars.yaml
```

2.3.4. Running Migration outside of Ansible

You can also run migration outside of Ansible because the Ansible module are written in Golang. The binaries are located in the plugins directory.

From your conversion host (or an Openstack instance inside the destination cloud) you need to export Openstack variables:

```
export OS_AUTH_URL=https://keystone-public-openstack.apps.ocp-4-16.standalone
export OS_PROJECT_NAME=admin
export OS_PASSWORD=admin
export OS_USERNAME=admin
export OS_DOMAIN_NAME=Default
export OS_PROJECT_ID=xyz
```

Then create the argument json file, for example:

```
cat <<EOF > args.json
{
        "user": "root",
        "password": "root",
        "server": "10.0.0.7",
        "vmname": "rhel-9.4-3",
        "cbtsync": false,
        "dst_cloud": {
            "auth": {
                "auth_url": "https://keystone-public-openstack.apps.ocp-4-
16.standalone",
                "username": "admin",
                "project_id": "xyz",
                "project_name": "admin",
                "user_domain_name": "Default",
```

```
                "password": "admin"
            },
            "region_name": "regionOne",
            "interface": "public",
            "identity_api_version": 3
        }
}
EOF
```

Then execute the `migrate` binary:

```
pushd vmware-migration-kit/vmware_migration_kit
./plugins/modules/migrate/migrate
```

You can see the logs into:

```
tail -f /tmp/osm-nbdkit.log
```

3. Ansible Execution Environment (AEE) Images

os-migrate and vmware-migration-kit provide containerized Ansible Execution Environment (AEE) images that encapsulate all necessary dependencies for running migration playbooks in a consistent, isolated environment.

3.1. Overview

Ansible Execution Environments are container images that provide a standardized runtime environment for Ansible automation. They include:

- Ansible Core and Ansible Runner
- Python runtime and required packages
- os-migrate and vmware-migration-kit collections
- All necessary dependencies and tools

This approach ensures consistent behavior across different environments and simplifies deployment and maintenance.

3.2. Available AEE Images

3.2.1. os-migrate AEE

The os-migrate AEE image contains:

- Ansible Core
- os-migrate Ansible collection
- OpenStack SDK and related Python packages
- All required dependencies for OpenStack resource migration

3.2.2. vmware-migration-kit AEE

The vmware-migration-kit AEE image contains:

- Ansible Core
- vmware-migration-kit Ansible collection
- VMware SDK and related Python packages
- All required dependencies for VMware to OpenStack migration

3.3. Building AEE Images

3.3.1. Prerequisites

Before building AEE images, ensure you have the following tools installed:

- `ansible-builder` - Tool for building execution environments
- `podman` or `docker` - Container runtime
- `git` - Version control system
- `python3` - Python runtime (version 3.8 or higher)

Setting Up a Virtual Environment

It's recommended to use a Python virtual environment to isolate dependencies and avoid conflicts with system packages.

Create and activate a virtual environment:

```
# Create virtual environment
python3 -m venv .venv

# Activate virtual environment (Linux/macOS)
source .venv/bin/activate

# Activate virtual environment (Windows)
.venv\Scripts\activate
```

Installing Dependencies

Install the required dependencies using the project-specific requirements files:

For os-migrate:

```
# Clone the repository
git clone https://github.com/os-migrate/os-migrate.git
cd os-migrate

# Create and activate virtual environment
python3 -m venv .venv
source .venv/bin/activate

# Install build dependencies
pip install -r requirements-build.txt
```

For vmware-migration-kit:

```
# Clone the repository
git clone https://github.com/os-migrate/vmware-migration-kit.git
cd vmware-migration-kit

# Create and activate virtual environment
python3 -m venv .venv
source .venv/bin/activate

# Install build dependencies
pip install -r requirements-build.txt
```

Requirements Files

Both repositories provide `requirements-build.txt` files that contain all necessary dependencies for building AEE images:

- **os-migrate requirements**: https://github.com/os-migrate/os-migrate/blob/main/requirements-build.txt
- **vmware-migration-kit requirements**: https://github.com/os-migrate/vmware-migration-kit/blob/main/requirements-build.txt

These files include: * `ansible-builder` - Core tool for building execution environments * `ansible-core` - Ansible runtime * `ansible-runner` - Execution environment runner * Additional Python packages required for the build process

Collection Requirements in AEE

The AEE images use `requirements.yml` files to specify which Ansible collections to install. The collection installation method depends on the build context:

For main branch builds (development):

Install collections directly from Git repositories using the main branch:

```yaml
# requirements.yml for main branch builds
collections:
  - name: https://github.com/os-migrate/os-migrate.git
    type: git
    version: main
  - name: https://github.com/os-migrate/vmware-migration-kit.git
    type: git
    version: main
```

For stable/tagged builds (production):

Install collections from Ansible Galaxy using specific version tags:

```yaml
# requirements.yml for stable/tagged builds
collections:
  - name: os_migrate.os_migrate
    version: "1.0.1"
  - name: os_migrate.vmware_migration_kit
    version: "2.0.4"
```

Benefits of this approach:

- **Main branch builds**: Always get the latest development code with latest features and fixes
- **Stable builds**: Use tested, released versions for production stability
- **Version consistency**: AEE image tags match the collection versions they contain
- **Reproducible builds**: Same collection versions produce identical AEE images

Alternative Installation Methods

If you prefer not to use virtual environments, you can install ansible-builder globally:

```
# Install ansible-builder globally
pip install ansible-builder

# Or install from requirements file
pip install -r requirements-build.txt
```

Note: Global installation may cause dependency conflicts with other Python projects on your system.

Virtual Environment Management

After completing your work, you can deactivate the virtual environment:

```
# Deactivate virtual environment
```

```
deactivate
```

To reactivate the virtual environment in future sessions:

```
# Navigate to the project directory
cd /path/to/os-migrate  # or vmware-migration-kit

# Activate the virtual environment
source .venv/bin/activate
```

Troubleshooting Virtual Environment Issues

Virtual environment not found

Ensure you're in the correct directory and the virtual environment was created successfully.

Permission denied

On some systems, you may need to use `python3` instead of `python` to create the virtual environment.

Dependencies not found

Make sure the virtual environment is activated before installing dependencies or building AEE images.

```
# Check if virtual environment is active
echo $VIRTUAL_ENV

# Verify ansible-builder is installed
which ansible-builder
ansible-builder --version
```

3.3.2. Building os-migrate AEE

Navigate to the os-migrate repository and build the AEE:

```
# Navigate to the repository
cd /path/to/os-migrate

# Activate virtual environment (if using one)
source .venv/bin/activate

# Navigate to AEE directory
cd aee

# Build the AEE image
ansible-builder build --tag os-migrate:latest
```

3.3.3. Building vmware-migration-kit AEE

Navigate to the vmware-migration-kit repository and build the AEE:

```
# Navigate to the repository
cd /path/to/vmware-migration-kit

# Activate virtual environment (if using one)
source .venv/bin/activate

# Navigate to AEE directory
cd aee

# Build the AEE image
ansible-builder build --tag vmware-migration-kit:latest
```

3.3.4. Automated Build Process

Both repositories include GitHub Actions workflows that automatically build and test AEE images:

- `os-migrate/.github/workflows/build-aee.yml`
- `vmware-migration-kit/.github/workflows/build-aee.yml`

These workflows:

- Trigger on pushes to main branch and pull requests
- Build the AEE image using ansible-builder
- Run basic validation tests
- Push images to container registries (when configured)

3.3.5. Release Versioning and Tagging Strategy

The GitHub Actions workflows implement a sophisticated versioning strategy for AEE images:

Main Branch Builds

Images built from the `main` branch are tagged as `latest`:

```
# When building from main branch
- name: Build and push AEE image
  if: github.ref == 'refs/heads/main'
  run: |
    ansible-builder build --tag ${{ github.repository }}:latest
    podman push ${{ github.repository }}:latest
```

Tag-based Builds

When building from Git tags, images receive multiple tags for maximum compatibility:

```
# When building from tags
- name: Build and push AEE image with version tags
  if: startsWith(github.ref, 'refs/tags/')
  run: |
    TAG_VERSION=${GITHUB_REF#refs/tags/}
    ansible-builder build --tag ${{ github.repository }}:$TAG_VERSION
    ansible-builder build --tag ${{ github.repository }}:stable

    podman push ${{ github.repository }}:$TAG_VERSION
    podman push ${{ github.repository }}:stable
```

Tagging Strategy

The versioning strategy follows these rules:

- `latest` - Always points to the most recent build from `main` branch
- `stable` - Points to the most recent tagged release (production-ready)
- `1.2.3` - Version without 'v' prefix for compatibility

Usage Examples

Use the appropriate tag based on your requirements:

```
# Use latest development version
podman run --rm os-migrate:latest ansible --version

# Use latest stable release
podman run --rm os-migrate:stable ansible --version

# Use specific version
podman run --rm os-migrate:1.2.3 ansible --version
```

Workflow Triggers

The GitHub Actions workflows are triggered by:

- `push` to `main` branch → builds `latest` tag
- `push` of tags → builds version-specific and `stable` tags
- `pull_request` to `main` → builds and tests (no push to registry)

Registry Configuration

Configure the container registry in the workflow using environment variables and secrets:

```
env:
  REGISTRY: quay.io
  IMAGE_NAME: os-migrate/os-migrate

- name: Login to Container Registry
  run: |
    podman login -u ${{ secrets.REGISTRY_USERNAME }} \
                 -p ${{ secrets.REGISTRY_PASSWORD }} \
                 ${{ env.REGISTRY }}

- name: Build and Push
  run: |
    ansible-builder build --tag ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{
steps.version.outputs.tag }}
    podman push ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{
steps.version.outputs.tag }}
```

Configuring Secrets and Environment Variables

GitHub Actions supports multiple levels of configuration for secrets and variables. Understanding these levels is crucial for proper AEE workflow configuration.

Repository-Level Secrets

Create secrets at the repository level for AEE workflows:

1. Navigate to your repository on GitHub
2. Click **Settings** → **Secrets and variables** → **Actions**
3. Click **New repository secret**
4. Add the following secrets for AEE workflows:

```
# Required secrets for AEE workflows
REGISTRY_USERNAME: your-registry-username
REGISTRY_PASSWORD: your-registry-password
REGISTRY_TOKEN: your-registry-token  # Alternative to username/password
```

Environment-Level Secrets

For production deployments, use environment-level secrets:

1. Go to **Settings** → **Environments**
2. Create environments like `production`, `staging`, `development`
3. Configure environment-specific secrets:

```
# Environment-specific secrets
production:
```

```
  REGISTRY_USERNAME: prod-registry-user
  REGISTRY_PASSWORD: prod-registry-password

staging:
  REGISTRY_USERNAME: staging-registry-user
  REGISTRY_PASSWORD: staging-registry-password
```

Organization-Level Variables

Use organization-level variables for shared configuration:

1. Go to organization **Settings** → **Secrets and variables** → **Actions**
2. Add organization variables:

```
# Organization variables (not secrets)
DEFAULT_REGISTRY: quay.io
DEFAULT_IMAGE_PREFIX: os-migrate
ANSIBLE_BUILDER_VERSION: 3.0.0
```

Repository-Level Variables

Create repository-level variables for project-specific configuration:

1. Navigate to your repository on GitHub
2. Click **Settings** → **Secrets and variables** → **Actions**
3. Click **Variables** tab → **New repository variable**
4. Add variables for AEE workflows:

```
# Repository variables for AEE workflows
IMAGE_NAME: os-migrate
BASE_IMAGE: quay.io/ansible/ansible-runner:latest
ANSIBLE_VERSION: 6.0.0
PYTHON_VERSION: 3.11
BUILD_CONTEXT: ./aee
```

Environment-Level Variables

Configure environment-specific variables:

1. Go to **Settings** → **Environments**
2. Select an environment (e.g., `production`)
3. Add environment-specific variables:

```
# Environment-specific variables
production:
```

```
  IMAGE_TAG: latest
  REGISTRY_URL: quay.io
  BUILD_ARGS: --no-cache --compress

staging:
  IMAGE_TAG: staging
  REGISTRY_URL: ghcr.io
  BUILD_ARGS: --no-cache

development:
  IMAGE_TAG: dev
  REGISTRY_URL: ghcr.io
  BUILD_ARGS: --progress=plain
```

Using Variables in Workflows

Access variables using the `vars` context in your workflows:

```
name: AEE Build with Variables
on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    environment: production

    steps:
      - uses: actions/checkout@v4

      - name: Set up Podman
        uses: redhat-actions/setup-podman@v1

      - name: Build AEE Image
        run: |
          cd ${{ vars.BUILD_CONTEXT }}
          ansible-builder build \
            --tag ${{ vars.REGISTRY_URL }}/${{ vars.IMAGE_NAME }}:${{ vars.IMAGE_TAG
}} \
            ${{ vars.BUILD_ARGS }}

      - name: Push Image
        run: |
          podman push ${{ vars.REGISTRY_URL }}/${{ vars.IMAGE_NAME }}:${{
vars.IMAGE_TAG }}
```

Variable Precedence

GitHub Actions follows this precedence order for variables and secrets:

1. **Environment variables** (highest priority)
2. **Environment-level secrets/variables**
3. **Repository-level secrets/variables**
4. **Organization-level secrets/variables**
5. **System variables** (lowest priority)

```
# Example showing variable precedence
name: Variable Precedence Example
on: push

jobs:
  test:
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Show Variable Values
        run: |
          echo "Repository variable: ${{ vars.IMAGE_NAME }}"
          echo "Environment variable: ${{ vars.IMAGE_TAG }}"
          echo "Organization variable: ${{ vars.DEFAULT_REGISTRY }}"
          echo "System variable: ${{ github.ref_name }}"
        env:
          # This overrides all other variables
          IMAGE_NAME: override-from-env
```

Workflow Configuration Examples

Basic Registry Authentication

```
name: Build AEE Image
on:
  push:
    branches: [main]
    tags: ['v*']

jobs:
  build:
    runs-on: ubuntu-latest
    environment: production  # Uses environment-level secrets

    steps:
      - uses: actions/checkout@v4
```

```
      - name: Set up Podman
        uses: redhat-actions/setup-podman@v1
        with:
          podman-version: latest

      - name: Login to Registry
        run: |
          echo ${{ secrets.REGISTRY_PASSWORD }} | podman login \
            --username ${{ secrets.REGISTRY_USERNAME }} \
            --password-stdin \
            ${{ vars.DEFAULT_REGISTRY }}

      - name: Build AEE Image
        run: |
          cd aee
          ansible-builder build --tag ${{ vars.DEFAULT_REGISTRY }}/${{
vars.DEFAULT_IMAGE_PREFIX }}:${{ github.ref_name }}

      - name: Push Image
        run: |
          podman push ${{ vars.DEFAULT_REGISTRY }}/${{ vars.DEFAULT_IMAGE_PREFIX
}}:${{ github.ref_name }}
```

Multi-Registry Support

```
name: Build and Push to Multiple Registries
on:
  push:
    tags: ['v*']

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        registry: [quay.io, ghcr.io, docker.io]

    steps:
      - uses: actions/checkout@v4

      - name: Set up Podman
        uses: redhat-actions/setup-podman@v1

      - name: Login to ${{ matrix.registry }}
        run: |
          case "${{ matrix.registry }}" in
            "quay.io")
              echo ${{ secrets.QUAY_TOKEN }} | podman login --username ${{
secrets.QUAY_USERNAME }} --password-stdin quay.io
              ;;
```

```
            "ghcr.io")
              echo ${{ secrets.GITHUB_TOKEN }} | podman login --username ${{
github.actor }} --password-stdin ghcr.io
              ;;
            "docker.io")
              echo ${{ secrets.DOCKERHUB_TOKEN }} | podman login --username ${{
secrets.DOCKERHUB_USERNAME }} --password-stdin docker.io
              ;;
          esac

      - name: Build and Push
        run: |
          cd aee
          ansible-builder build --tag ${{ matrix.registry }}/os-migrate:${{
github.ref_name }}
          podman push ${{ matrix.registry }}/os-migrate:${{ github.ref_name }}
```

Secure Secret Handling

Follow security best practices when using secrets:

```
- name: Secure Secret Usage
  run: |
    # ✅ Good: Use environment variables
    export REGISTRY_PASSWORD="${{ secrets.REGISTRY_PASSWORD }}"
    podman login --username ${{ secrets.REGISTRY_USERNAME }} --password-stdin ${{
env.REGISTRY }}

    # ✅ Good: Use proper quoting
    podman login --username "${{ secrets.REGISTRY_USERNAME }}" --password "${{
secrets.REGISTRY_PASSWORD }}" ${{ env.REGISTRY }}

    # ❌ Bad: Direct command line usage without quoting
    podman login --username ${{ secrets.REGISTRY_USERNAME }} --password ${{
secrets.REGISTRY_PASSWORD }} ${{ env.REGISTRY }}
  env:
    REGISTRY: ${{ vars.DEFAULT_REGISTRY }}
```

Conditional Secret Usage

Use secrets conditionally based on workflow context:

```
- name: Conditional Registry Login
  if: github.event_name == 'push' && github.ref == 'refs/heads/main'
  run: |
    echo ${{ secrets.REGISTRY_PASSWORD }} | podman login \
      --username ${{ secrets.REGISTRY_USERNAME }} \
      --password-stdin \
      ${{ env.REGISTRY }}
```

```
- name: Build and Push (Main Branch)
  if: github.event_name == 'push' && github.ref == 'refs/heads/main'
  run: |
    cd aee
    ansible-builder build --tag ${{ env.REGISTRY }}/os-migrate:latest
    podman push ${{ env.REGISTRY }}/os-migrate:latest

- name: Build and Push (Tags)
  if: github.event_name == 'push' && startsWith(github.ref, 'refs/tags/')
  run: |
    cd aee
    TAG_VERSION=${GITHUB_REF#refs/tags/}
    ansible-builder build --tag ${{ env.REGISTRY }}/os-migrate:$TAG_VERSION
    ansible-builder build --tag ${{ env.REGISTRY }}/os-migrate:stable
    podman push ${{ env.REGISTRY }}/os-migrate:$TAG_VERSION
    podman push ${{ env.REGISTRY }}/os-migrate:stable
```

Secret Rotation and Management

Implement secret rotation strategies:

```
- name: Validate Secrets
  run: |
    if [ -z "${{ secrets.REGISTRY_USERNAME }}" ]; then
      echo "❌ REGISTRY_USERNAME secret is not set"
      exit 1
    fi

    if [ -z "${{ secrets.REGISTRY_PASSWORD }}" ]; then
      echo "❌ REGISTRY_PASSWORD secret is not set"
      exit 1
    fi

    echo "✅ All required secrets are configured"

- name: Test Registry Access
  run: |
    echo ${{ secrets.REGISTRY_PASSWORD }} | podman login \
      --username ${{ secrets.REGISTRY_USERNAME }} \
      --password-stdin \
      ${{ env.REGISTRY }} --test
    echo "✅ Registry authentication successful"
```

Environment-Specific Configuration

Use different configurations for different environments:

```
name: AEE Build Matrix
```

```yaml
on:
  push:
    branches: [main, develop]
    tags: ['v*']

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        include:
          - environment: development
            registry: ghcr.io
            image_tag: dev
          - environment: staging
            registry: quay.io
            image_tag: staging
          - environment: production
            registry: quay.io
            image_tag: latest

    environment: ${{ matrix.environment }}

    steps:
      - uses: actions/checkout@v4

      - name: Set up Podman
        uses: redhat-actions/setup-podman@v1

      - name: Login to Registry
        run: |
          echo ${{ secrets.REGISTRY_PASSWORD }} | podman login \
            --username ${{ secrets.REGISTRY_USERNAME }} \
            --password-stdin \
            ${{ matrix.registry }}

      - name: Build AEE Image
        run: |
          cd aee
          ansible-builder build --tag ${{ matrix.registry }}/os-migrate:${{
matrix.image_tag }}

      - name: Push Image
        run: |
          podman push ${{ matrix.registry }}/os-migrate:${{ matrix.image_tag }}
```

3.4. Using AEE Images

3.4.1. Running Playbooks with AEE

Execute os-migrate playbooks using the AEE container:

```
podman run --rm -it \
  -v $(pwd):/runner \
  -v ~/.ssh:/home/runner/.ssh:ro \
  os-migrate:latest \
  ansible-playbook -i inventory playbook.yml
```

3.4.2. Interactive Shell Access

Access the AEE container interactively for debugging:

```
podman run --rm -it \
  -v $(pwd):/runner \
  -v ~/.ssh:/home/runner/.ssh:ro \
  os-migrate:latest \
  /bin/bash
```

3.4.3. Volume Mounts

Common volume mounts for AEE usage:

- `$(pwd):/runner` - Mount current directory as working directory
- `~/.ssh:/home/runner/.ssh:ro` - Mount SSH keys (read-only)
- `~/.config/openstack:/home/runner/.config/openstack:ro` - Mount OpenStack credentials
- `/path/to/inventory:/runner/inventory:ro` - Mount inventory files

3.5. AEE Configuration

3.5.1. Execution Environment Definition

AEE images are defined using `execution-environment.yml` files that specify:

- Base image (typically `quay.io/ansible/ansible-runner:latest`)
- Python dependencies
- Ansible collections
- Additional system packages

Example structure:

```
version: 1
dependencies:
```

```
  galaxy:
    - name: os_migrate.os_migrate
      source: https://github.com/os-migrate/os-migrate
  python:
    - openstacksdk>=1.0.0
    - ansible>=6.0.0
  system:
    - git
    - openssh-clients
```

3.5.2. Customizing AEE Images

To customize AEE images for specific requirements:

1. Modify the `execution-environment.yml` file
2. Add custom requirements or collections
3. Rebuild the image using ansible-builder

```
ansible-builder build --tag custom-aee:latest
```

3.6. Troubleshooting

3.6.1. Secrets and Variables Issues

Common Secret Configuration Problems

Secret Not Found

Ensure the secret is created at the correct level (repository, environment, or organization) and the name matches exactly in the workflow.

Permission Denied

Verify that the workflow has access to the environment containing the secrets. Check environment protection rules and required reviewers.

Empty Secret Value

Secrets that are not set return empty strings. Always validate secret existence before use.

```
- name: Validate Required Secrets
  run: |
    if [ -z "${{ secrets.REGISTRY_USERNAME }}" ]; then
      echo "❌ REGISTRY_USERNAME secret is not configured"
      exit 1
    fi

    if [ -z "${{ secrets.REGISTRY_PASSWORD }}" ]; then
      echo "❌ REGISTRY_PASSWORD secret is not configured"
```

```
      exit 1
    fi

    echo "✅ All required secrets are available"
```

Variable Access Issues

Variable Not Defined

Check that variables are created at the appropriate level and use the correct context (`vars` for variables, `secrets` for secrets).

Wrong Variable Context

Use `${{ vars.VARIABLE_NAME }}` for variables and `${{ secrets.SECRET_NAME }}` for secrets.

```
- name: Debug Variable Access
  run: |
    echo "Repository variables:"
    echo "  IMAGE_NAME: ${{ vars.IMAGE_NAME }}"
    echo "  BUILD_CONTEXT: ${{ vars.BUILD_CONTEXT }}"

    echo "Environment variables:"
    echo "  IMAGE_TAG: ${{ vars.IMAGE_TAG }}"
    echo "  REGISTRY_URL: ${{ vars.REGISTRY_URL }}"

    echo "Organization variables:"
    echo "  DEFAULT_REGISTRY: ${{ vars.DEFAULT_REGISTRY }}"
```

Registry Authentication Troubleshooting

Authentication Failed

Verify credentials are correct and have appropriate permissions for the registry.

Token Expired

Check if the registry token has expired and needs renewal.

```
- name: Test Registry Authentication
  run: |
    echo "Testing authentication to ${{ vars.DEFAULT_REGISTRY }}"

    # Test login without pushing
    echo ${{ secrets.REGISTRY_PASSWORD }} | podman login \
      --username ${{ secrets.REGISTRY_USERNAME }} \
      --password-stdin \
      ${{ vars.DEFAULT_REGISTRY }} --test

    if [ $? -eq 0 ]; then
      echo "✅ Registry authentication successful"
    else
```

```
      echo "❌ Registry authentication failed"
      exit 1
    fi
```

3.6.2. Debugging AEE Issues

Enable verbose output for troubleshooting:

```
podman run --rm -it \
  -v $(pwd):/runner \
  os-migrate:latest \
  ansible-playbook -vvv -i inventory playbook.yml
```

Check container logs:

```
podman logs <container_id>
```

3.6.3. Performance Optimization

- Use volume mounts instead of copying files into containers
- Mount only necessary directories to reduce I/O overhead
- Consider using read-only mounts where possible
- Use appropriate resource limits for container execution

3.7. Maintenance

3.7.1. Updating AEE Images

Regular updates ensure security and compatibility:

1. Update base images in execution environment definitions
2. Update Ansible collections to latest versions
3. Update Python dependencies
4. Rebuild and test AEE images
5. Update documentation with any changes

3.7.2. Version Management

The automated GitHub Actions workflows handle version management based on Git references:

Manual Version Management

For local development, you can manually tag images:

```
# Build specific version locally
ansible-builder build --tag os-migrate:1.2.3

# Build latest development version
ansible-builder build --tag os-migrate:latest
```

Automated Version Management

The GitHub Actions workflows automatically handle versioning:

- **Main branch pushes** → `latest` tag
- **Tag pushes** → version-specific tag + `stable` tag
- **Pull requests** → build and test only (no registry push)

Creating Releases

To create a new release:

1. Create and push a Git tag: [source,bash] ---- git tag -a 1.2.3 -m "Release version 1.2.3" git push origin 1.2.3 ----
2. The GitHub Actions workflow will automatically:
   - Build the AEE image
   - Tag it with `1.2.3` and `stable`
   - Push to the configured registry

Version Tag Strategy

- `latest` - Development builds from main branch
- `stable` - Latest tagged release (production-ready)
- `1.2.3` - Specific version

3.7.3. Security Considerations

- Regularly update base images to include security patches
- Scan AEE images for vulnerabilities
- Use minimal base images when possible
- Review and audit all included dependencies

3.8. Best Practices

3.8.1. Development Workflow

1. Test changes locally using AEE containers

2. Use version-controlled execution environment definitions
3. Document any customizations or modifications
4. Test AEE images in target environments before deployment

3.8.2. Production Usage

1. Use specific version tags instead of `latest`
2. Implement proper monitoring and logging
3. Regular security updates and vulnerability scanning
4. Backup and disaster recovery planning

3.8.3. Documentation

1. Keep execution environment definitions well-documented
2. Document any custom modifications or extensions
3. Provide clear usage examples and troubleshooting guides
4. Maintain compatibility matrices for different versions

3.9. TODO

3.9.1. Collection Installation Improvements

Improve the way collections (os-migrate or vmware-migration-kit) are installed within AEE images to ensure proper version alignment:

- **Main branch builds**: When the image tag is `main`, install the collection content directly from the main branch repository as the source of installation using Git-based requirements
- **Stable/tagged builds**: When the image tag is `stable` or matches a repository tag, ensure the installation uses the corresponding tagged version of the collection from Ansible Galaxy
- **Dynamic requirements.yml**: Implement automated generation of `requirements.yml` files based on build context to ensure proper collection versioning
- **Version consistency validation**: Add build-time checks to verify that AEE image tags match the collection versions they contain

This improvement will ensure that AEE images always contain the correct version of the collection that matches the build context and tag strategy, providing better reproducibility and version alignment.

4. Community

The source code of OS Migrate is hosted on GitHub.

For issue reports please use the GitHub OS Migrate issue tracker.

To get help, feel free to also create an issue on GitHub with your question.