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. 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.

If you want to contribute to the project (code, docs, …), please refer to the developer docs.

3. Contents

4. OS Migrate Operator Guide

This guide provides comprehensive information for operators planning and executing OpenStack cloud migrations using OS Migrate.

ifdev::context[:parent-context: {context}]

4.1. os-migrate overview

OS-Migrate is a toolsuite designed to assist in the process of moving cloud workloads from one environment to another. Please read the entire workflow notes in order to be sufficiently prepared for this endeavour.

TO BE FLESHED OUT

ifdev::context[:parent-context: {context}]

4.2. OS Migrate Quickstart Guide

4.2.1. Capacity Planning & Conversion Host

ifdev::context[:parent-context: planning]

4.2.2. OS Migrate Capacity Planning

Migrator Host Requirements

Overview

The migrator host is the machine that executes OS Migrate Ansible playbooks and serves as the orchestration point for OpenStack cloud migrations. It can be deployed as:

- A source cloud instance
- A third-party machine (physical/virtual)
- An Ansible Execution Environment (container)

System Requirements

Operating System

- Recommended: CentOS Stream 10 or RHEL 9.5+
- Alternative: Any Linux distribution with virtio-win package ≥1.40 (for workload migrations)
- Container environments supported for Ansible Execution Environments

Software Dependencies

Core Requirements

- Python: 3.7+ (badges show 3.7+, container uses 3.12)
- Ansible: 2.9+ minimum (current: 11.7.0)

Python Packages

```
ansible==11.7.0
ansible-lint==25.6.1
openstacksdk==1.5.1
python-openstackclient==6.3.0
passlib==1.7.4
PyYAML==6.0.2
```

Ansible Collections

```
community.crypto: ">=1.4.0"
community.general: "<5.0.0"
openstack.cloud: ">=2.3.0,<2.4.0"
```

Storage Requirements

- Sufficient disk space in os_migrate_data_dir (default: /home/migrator/os-migrate-data)
- Separate data directories required for each source project migration
- Additional space for volume conversion during workload migrations

Network Access

- Outbound: HTTPS access to both source and destination OpenStack APIs
- Inbound: SSH access if used as remote conversion host
- Bandwidth: Adequate for volume/image data transfer during migrations

Configuration Requirements

Authentication

- Valid OpenStack credentials for both source and destination clouds
- clouds.yaml configuration or environment variables
- Support for Keystone v2/v3 authentication types

Inventory Configuration

```yaml
migrator:
  hosts:
    localhost:
      ansible_connection: local
      ansible_python_interpreter: "{{ ansible_playbook_python }}"
```

Required Variables

- os_migrate_src_auth: Source cloud authentication
- os_migrate_dst_auth: Destination cloud authentication
- os_migrate_data_dir: Migration data storage location
- Region and API version configurations as needed

Deployment Options

1. Source Cloud Instance

- Pros: Close network proximity to source resources
- Cons: Resource consumption on source cloud
- Use case: When destination cloud has limited external access

2. Third-Party Machine

- Pros: Independent resource allocation, flexible placement
- Cons: Network latency considerations
- Use case: Large migrations, dedicated migration infrastructure

3. Ansible Execution Environment

- Pros: Consistent, portable, containerized execution
- Cons: Container orchestration complexity
- Use case: CI/CD pipelines, automated migrations

Special Considerations

Workload Migration

- Additional conversion host may be required for instance migrations
- SSH key access to conversion hosts
- Volume attachment/detachment capabilities

Multi-Project Migrations

- Separate data directories for each source project
- Proper isolation of migration data
- Sequential or parallel execution planning

Security

- No direct database access (API-only operations)
- Secure credential management
- Network security between clouds

ifdev::context[:parent-context: planning]

4.2.3. Conversion Host Guide

The conversion host role is a critical component of OS Migrate for migrating OpenStack workloads (instances) and their associated volumes between OpenStack clouds. It deploys dedicated compute instances that act as intermediary hosts for data transfer and conversion operations during workload migration.

Purpose

Conversion hosts serve as temporary staging instances that:

- **Data Transfer**: Facilitate the transfer of workload disk data between source and destination clouds

- **Format Conversion**: Handle disk format conversions if needed between different storage backends
- **Network Bridging**: Provide network connectivity between source and destination environments
- **Migration Orchestration**: Execute the complex multi-step process of workload migration

Architecture

The conversion host system consists of two main Ansible roles:

`conversion_host` Role

Location: `/roles/conversion_host/`

Purpose: Deploys the infrastructure and compute instances needed for conversion hosts.

Key Responsibilities:

- Creates networking infrastructure (networks, subnets, routers, security groups)
- Generates SSH keypairs for secure communication
- Deploys conversion host instances in both source and destination clouds
- Configures floating IPs for external connectivity
- Sets up security group rules for SSH and ICMP access

`conversion_host_content` Role

Location: `/roles/conversion_host_content/`

Purpose: Installs and configures software packages on the conversion hosts.

Key Responsibilities:

- Installs OS-specific packages (supports CentOS and RHEL)
- Configures RHEL subscription management if needed
- Sets up SSH keys for inter-host communication
- Enables password access if required
- Runs pre/post installation hooks

How It Works

Deployment Process

1. **Infrastructure Setup** (`conversion_host` role):

   ```
   # Network Infrastructure
   - Creates conversion network (default: os_migrate_conv)
   - Creates subnet with CIDR 192.168.10.0/24
   ```

```
- Creates router connected to external network
- Creates security group with SSH/ICMP rules
```

2. **Instance Deployment**:

```
# Source Cloud Instance
os_migrate_src_conversion_host_name: os_migrate_conv_src

# Destination Cloud Instance
os_migrate_dst_conversion_host_name: os_migrate_conv_dst
```

3. **SSH Key Configuration**:
   - Generates keypair for conversion host access
   - Configures authorized keys on source host
   - Installs private key on destination host for src→dst communication
4. **Software Installation** (`conversion_host_content` role):
   - Installs required packages based on OS distribution
   - Configures any RHEL subscriptions
   - Sets up conversion tools and dependencies

Migration Workflow Integration

During workload migration, conversion hosts are used in the following workflow:

1. **Volume Export**: Source conversion host exports volumes from source cloud storage
2. **Data Transfer**: Volumes are transferred from source to destination conversion host
3. **Volume Import**: Destination conversion host imports volumes to destination storage
4. **Instance Creation**: New instance is created using imported volumes

Network Architecture

```
Source Cloud                  Destination Cloud
+------------------+        +------------------+
| Conversion Host  |        | Conversion Host  |
| (os_migrate_conv)|   SSH  | (os_migrate_conv)|
| 192.168.10.x     |<------>| 192.168.10.x     |
+------------------+        +------------------+
         |                           |
    [Floating IP]              [Floating IP]
         |                           |
    [External Net]             [External Net]
```

Configuration

Required Variables

```
# Must be defined
os_migrate_conversion_flavor_name: m1.large
os_migrate_conversion_external_network_name: public
```

Network Configuration

```
# Network settings (with defaults)
os_migrate_conversion_net_name: os_migrate_conv
os_migrate_conversion_subnet_name: os_migrate_conv
os_migrate_conversion_subnet_cidr: 192.168.10.0/24
os_migrate_conversion_subnet_alloc_start: 192.168.10.10
os_migrate_conversion_subnet_alloc_end: 192.168.10.99
os_migrate_conversion_router_name: os_migrate_conv
os_migrate_conversion_router_ip: 192.168.10.1
```

Host Configuration

```
# Instance settings
os_migrate_src_conversion_host_name: os_migrate_conv_src
os_migrate_dst_conversion_host_name: os_migrate_conv_dst
os_migrate_conversion_image_name: os_migrate_conv
os_migrate_conversion_host_ssh_user: cloud-user

# Boot from volume options
os_migrate_src_conversion_host_boot_from_volume: false
os_migrate_dst_conversion_host_boot_from_volume: false
os_migrate_src_conversion_host_volume_size: 20
os_migrate_dst_conversion_host_volume_size: 20
```

Security Configuration

```
# SSH and security
os_migrate_conversion_keypair_name: os_migrate_conv
os_migrate_conversion_keypair_private_path: "{{ os_migrate_data_dir
}}/conversion/ssh.key"
os_migrate_conversion_secgroup_name: os_migrate_conv

# Password access (disabled by default)
os_migrate_conversion_host_ssh_user_enable_password_access: false
```

Management Options

```yaml
# Infrastructure management
os_migrate_conversion_manage_network: true
os_migrate_conversion_manage_fip: true
os_migrate_conversion_delete_fip: true

# Content installation
os_migrate_conversion_host_content_install: true

# Deployment control
os_migrate_deploy_src_conversion_host: true
os_migrate_deploy_dst_conversion_host: true
os_migrate_link_conversion_hosts: true
os_migrate_reboot_conversion_hosts: false
```

Usage

Deploy Conversion Hosts

Use the provided playbook to deploy conversion hosts:

```
ansible-playbook os_migrate.os_migrate.deploy_conversion_hosts
```

This playbook will:

1. Deploy source conversion host infrastructure and instance
2. Deploy destination conversion host infrastructure and instance
3. Configure SSH linking between hosts
4. Install required software packages
5. Perform health checks

Delete Conversion Hosts

Clean up conversion hosts and their infrastructure:

```
ansible-playbook os_migrate.os_migrate.delete_conversion_hosts
```

Manual Role Usage

You can also use the roles directly for more control:

```yaml
# Deploy source conversion host
- name: Deploy source conversion host
  include_role:
    name: os_migrate.os_migrate.conversion_host
  vars:
    os_migrate_conversion_cloud: src
    os_migrate_conversion_host_name: "{{ os_migrate_src_conversion_host_name }}"
    # ... other source-specific variables

# Deploy destination conversion host
- name: Deploy destination conversion host
  include_role:
    name: os_migrate.os_migrate.conversion_host
  vars:
    os_migrate_conversion_cloud: dst
    os_migrate_conversion_host_name: "{{ os_migrate_dst_conversion_host_name }}"
    # ... other destination-specific variables
```

Integration with Workload Migration

Conversion hosts are automatically used during workload migration when:

1. **Workload Export/Import**: The `import_workloads` role checks conversion host status
2. **Data Copy Operations**: Volume data is transferred via conversion hosts
3. **Health Checks**: Ensures conversion hosts are active and reachable before migration

The `os_conversion_host_info` module provides runtime information about conversion hosts:

```
- name: Get conversion host info
  os_migrate.os_migrate.os_conversion_host_info:
    cloud: src
    server: "{{ os_migrate_src_conversion_host_name }}"
  register: conversion_host_info
```

Prerequisites

Cloud Requirements

- **Flavor**: Adequate flavor for conversion operations (recommended: >= 2 vCPU, 4GB RAM)
- **Image**: Compatible base image (CentOS/RHEL with cloud-init)
- **Network**: External network for floating IP assignment
- **Quotas**: Sufficient quota for additional instances, networks, and floating IPs

Permissions

The deployment requires OpenStack permissions for:

- Instance creation/deletion
- Network resource management (networks, subnets, routers)
- Security group management

- Keypair management
- Floating IP allocation

Troubleshooting

Common Issues

1. Conversion Host Not Reachable

```
# Check conversion host status
- os_conversion_host_info module will fail if host is not ACTIVE
- Verify floating IP assignment
- Check security group rules allow SSH (port 22)
```

2. Network Connectivity Issues

```
# Verify network configuration
- External network name is correct
- Router has gateway set to external network
- DNS nameservers are accessible (default: 8.8.8.8)
```

3. SSH Key Problems

```
# Key file permissions
- Private key must have 0600 permissions
- Public key must be accessible to Ansible
- Verify keypair exists in OpenStack
```

4. Package Installation Failures

```
# RHEL subscription issues
- Check RHEL subscription configuration
- Verify repository access
- Review pre/post content hooks
```

Debugging Steps

1. **Check Conversion Host Status**:

```yaml
- name: Debug conversion host
  os_migrate.os_migrate.os_conversion_host_info:
    cloud: src
    server: "{{ os_migrate_src_conversion_host_name }}"
  register: debug_info
```

```
- debug: var=debug_info
```

2. **Verify Network Connectivity**:

```
- name: Test SSH connectivity
  wait_for:
    port: 22
    host: "{{ conversion_host_ip }}"
    timeout: 60
```

3. **Check Infrastructure**:

```
# Verify OpenStack resources exist
openstack server list --name os_migrate_conv
openstack network list --name os_migrate_conv
openstack security group list --name os_migrate_conv
```

Best Practices

Security

- Use dedicated keypairs for conversion hosts
- Limit security group rules to necessary ports only
- Consider using specific subnets for conversion traffic
- Disable password authentication unless specifically required

Performance

- Choose appropriate flavors with sufficient CPU and memory
- Consider boot-from-volume for larger disk operations
- Use local storage-optimized flavors when available
- Monitor network bandwidth during large migrations

Cleanup

- Always clean up conversion hosts after migration
- Set `os_migrate_conversion_delete_fip: true` to clean floating IPs
- Use the delete playbook to ensure complete cleanup
- Monitor for orphaned resources after deletion

Testing

- Test conversion host deployment in development environment first
- Verify connectivity between source and destination hosts

- Test package installation and configuration
- Validate migration workflow with small test instances

Related Components

- `import_workloads` **role**: Uses conversion hosts for actual migration operations
- `os_conversion_host_info` **module**: Provides runtime host information
- **Volume migration modules**: Transfer data via conversion hosts
- **Workload migration utilities**: Coordinate conversion host operations

Examples

Basic Deployment

```
# Minimal configuration for conversion host deployment
os_migrate_conversion_flavor_name: m1.large
os_migrate_conversion_external_network_name: public
os_migrate_src_conversion_image_name: centos-stream-9
os_migrate_dst_conversion_image_name: centos-stream-9
```

Advanced Configuration

```
# Advanced configuration with custom networks
os_migrate_conversion_flavor_name: m1.xlarge
os_migrate_conversion_external_network_name: external
os_migrate_src_conversion_net_name: migration_src_net
os_migrate_dst_conversion_net_name: migration_dst_net
os_migrate_conversion_subnet_cidr: 10.10.10.0/24
os_migrate_src_conversion_host_boot_from_volume: true
os_migrate_dst_conversion_host_boot_from_volume: true
os_migrate_src_conversion_host_volume_size: 50
os_migrate_dst_conversion_host_volume_size: 50
```

4.2.4. Installation

4.2.5. 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.

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.

Available AEE Images

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

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

Building AEE Images

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:

```
# 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:

```
# 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
```

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
```

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
```

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)

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`:

```yaml
# 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:

```yaml
# 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:

```bash
# 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:

```yaml
# 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:

```yaml
# 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:

```yaml
# 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:

```yaml
# 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:

```yaml
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)

```yaml
# 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

```yaml
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:

```yaml
- 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:

```yaml
- 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
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 }}
```

Using AEE Images

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
```

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
```

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

AEE Configuration

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:

```yaml
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
```

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
```

Troubleshooting

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.

```yaml
- 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
```

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>
```

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

Maintenance

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

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

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

Best Practices

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

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

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

TODO

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.

4.2.6. OS Migrate VMware to OpenStack

ifdev::context[:parent-context: planning]

4.2.7. OS Migrate VMware Guide

An important auxilliary function included in OS Migrate is our VMware tooling, which allows you to migrade a virtual machine from an ESXi/Vcenter environment to OpenStack or OpenShift 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 and Openshift:

- 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.

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.

Features and supported OS

Features

The following features are availables:

- Discovery mode
- Network mapping

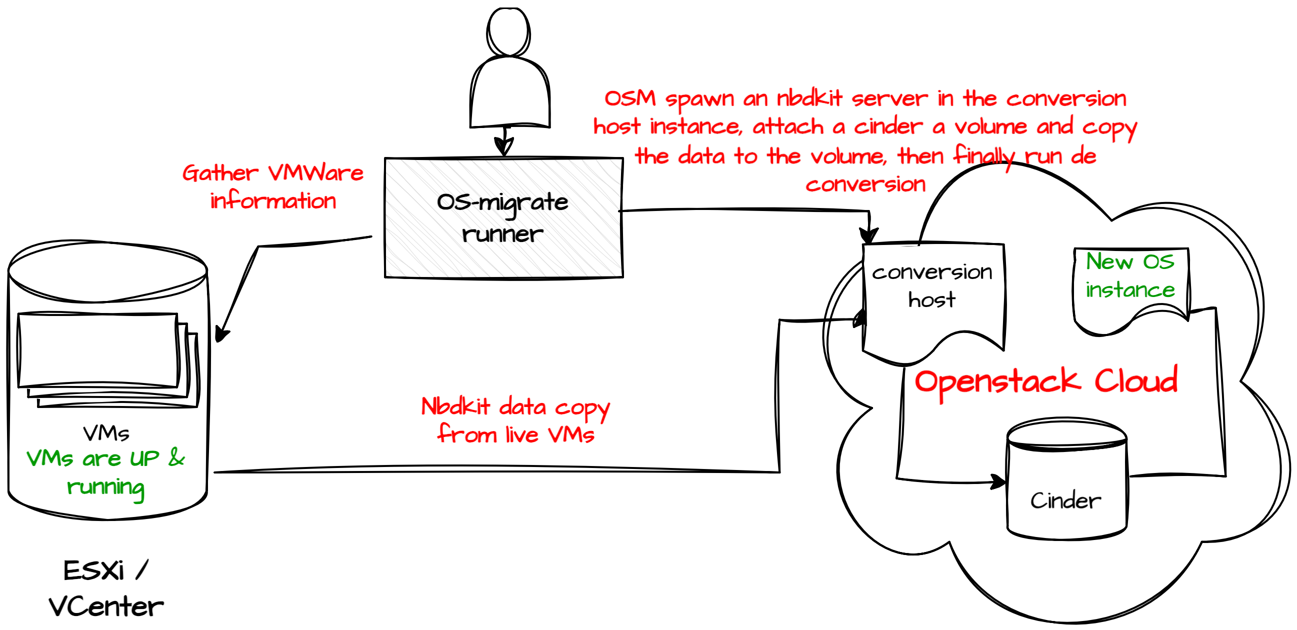
- Port creation and mac addresses mapping
- Openstack flavor mapping and creation
- Migration with nbdkit server with change block tracking feature (CBT)
- Migration with virt-v2v
- Upload migrate volume via Glance
- Multi disks migration
- Multi nics
- Parallel migration on a same conversion host
-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

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
-	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	-

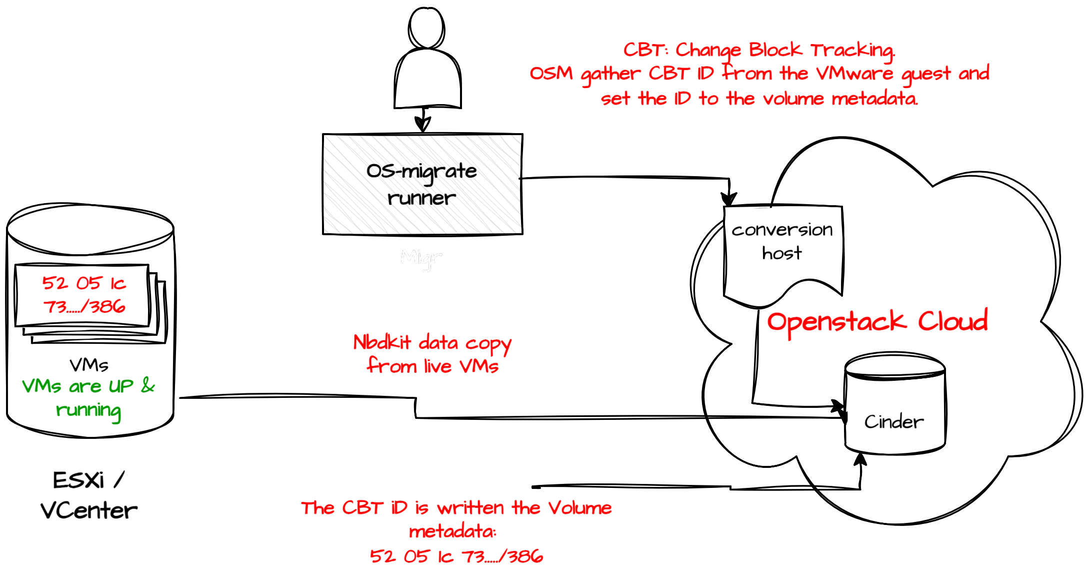
Nbdkit migration example


OS-migrate runner
OSM spawn an nbdkit server in the conversion host instance, attach a cinder a volume and copy the data to the volume, then finally run de conversion
Gather VMWare information
conversion host
New OS instance
Openstack Cloud
Nbdkit data copy from live VMs
VMs
VMs are UP & running
Cinder
ESXi / VCenter

Nbdkit migration example with the Change Block Tracking

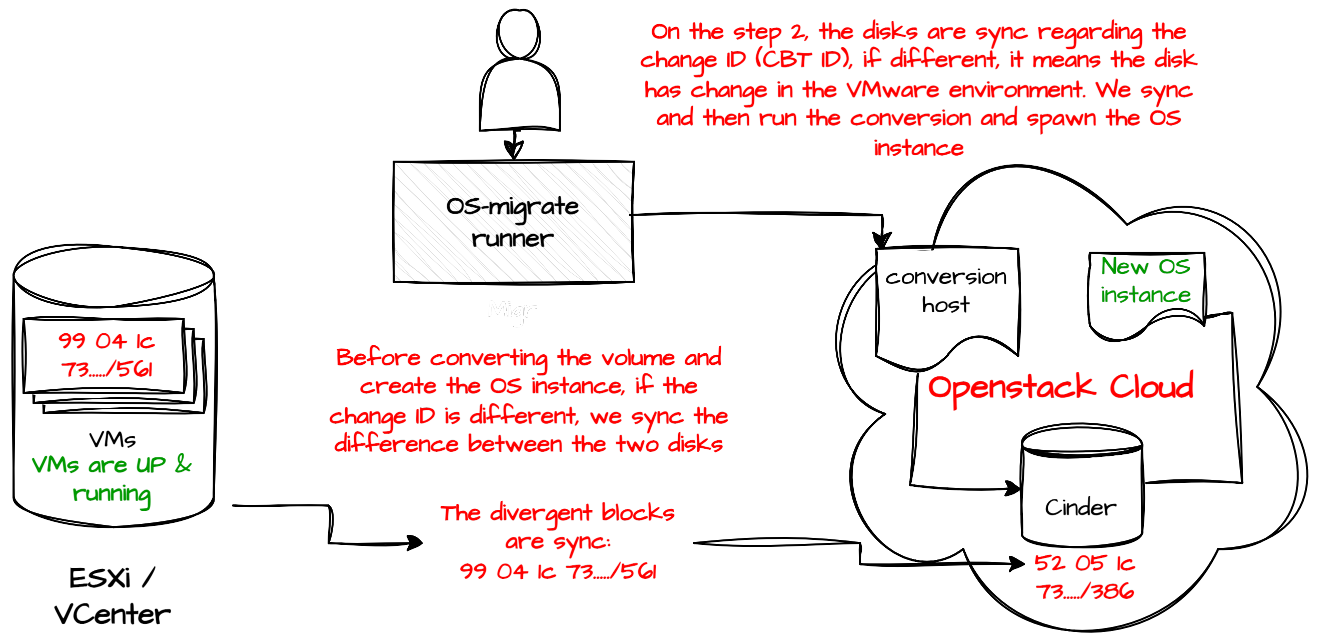
Step 1: The data are copied and the change ID from the VMware disk are set to the Cinder volume as metadata

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.


CBT: Change Block Tracking
OSM gather CBT ID from the VMware guest and set the ID to the volume metadata
OS-migrate runner
conversion host
Openstack Cloud
52 05 lc 73..../386
VMs
VMs are UP & running
Nbdkit data copy from live VMs
Cinder
ESXi / VCenter
The CBT iD is written the Volume metadata: 52 05 lc 73..../386

==== 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.


On the step 2, the disks are sync regarding the change ID (CBT ID), if different, it means the disk has change in the VMware environment. We sync and then run the conversion and spawn the OS instance
OS-migrate runner
99 04 lc 73..../561
VMs
VMs are UP & running
ESXi / VCenter
Before converting the volume and create the OS instance, if the change ID is different, we sync the difference between the two disks
The divergent blocks are sync: 99 04 lc 73..../561
conversion host
New OS instance
Openstack Cloud
Cinder
52 05 lc 73..../386

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
```

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
```

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
```

Virt-v2v

Provide the following additional informations:

```
# virt-v2v parameters
vddk_thumbprint: XX:XX:XX
vddk_libdir: /usr/lib/vmware-vix-disklib
```

In order to generate the thumbprint of your VMWare source cloud you need to use:

```
# thumbprint
openssl s_client -connect ESXI_SERVER_NAME:443 </dev/null |
   openssl x509 -in /dev/stdin -fingerprint -sha1 -noout
```

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
```

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
```

ifdev::context[:parent-context: vmware]

4.3. OS Migrate walkthrough

OS Migrate is a framework for OpenStack parallel cloud migration (migrating content between OpenStack tenants which are not necessarily in the same cloud). 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.

Parallel cloud migration is a way to modernize an OpenStack deployment. Instead of upgrading an OpenStack cluster in place, a second OpenStack cluster is deployed alongside, and tenant content is migrated from the original cluster to the new one. Parallel cloud migration is best suited for environments which are due for a hardware refresh cycle. It may also be performed without a hardware refresh, but extra hardware resources are needed to bootstrap the new cluster. As hardware resources free up in the original cluster, they can be gradually added to the new cluster.

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.

The migration is generally performed in this sequence:

- prerequisites: prepare authentication info, parameter files,
- pre-workload migration, which copies applicable resources into the destination cloud (e.g. networks, security groups, images) while workloads keep running in the source cloud,
- workload migration, which stops usage of applicable resources in the source cloud and moves them into the destination cloud (VMs, volumes).

4.3.1. Prerequisites

Authentication

Users are encouraged to use os-migrate using specific credentials for each project/tenant, this means **not using the admin user to execute the resources migration** (unless the resource is owned by the admin project, e.g. public Glance images).

In case the circumstances require migrating by the `admin` user, this user needs to have access to the respective projects. There are two options:

- Add the `admin` user as a `member` of each project.

  ```
  Depending on how many projects need to be migrated this approach seems
  to be suboptimal as there are involved several configuration updates in
  the projects that will need to be reverted after the migration
  completes.
  ```

- Create a group including the admin user and add the group to each project as member.

```
The difference with this approach is that once the migration is
completed, by removing the group, all the references in all the projects
will be removed automatically.
```

Parameter file

Let’s create an `os-migrate-vars.yml` file with Ansible variables:

YAML Ansible Variables

```yaml
os_migrate_src_auth:
  auth_url: http://192.168.0.13.199/v3
  password: srcpassword
  project_domain_name: Default
  project_name: src
  user_domain_name: Default
  username: src
os_migrate_src_region_name: regionOne
os_migrate_dst_auth:
  auth_url: http://192.167.0.16:5000/v3
  password: dstpassword
  project_domain_name: Default
  project_name: dst
  user_domain_name: Default
  username: dst
os_migrate_dst_region_name: regionOne
os_migrate_data_dir: /home/migrator/os-migrate-data
```

The file contains the source and destination tenant credentials, a directory on the migrator host (typically localhost) and a directory where the exported data will be saved.

If you are migrating content from multiple source projects, make sure to use a separate data directory for each source project. In other words, when changing `os_migrate_src_auth` or `os_migrate_src_region_name`, make sure to also change `os_migrate_data_dir`.

A note about Keystone v2

As depicted in content of the previously defined `os-migrate-vars.yml` file, the parameters `os_migrate_src_auth` and `os_migrate_dst_auth` refer to the usage of Keystone v3. In the case of a user needing to execute a migration between tenants not supporting Keystone v3 the following error will be raised:

```
keystoneauth1.exceptions.discovery.DiscoveryFailure: Cannot use v2 authentication with
domain scope
```

To fix this issue, the user must adjust their auth parameters:

YAML Keystone V2 Auth Params

```
os_migrate_src_auth:
  auth_url: http://192.168.0.13.199/v2.0
  password: srcpassword
  project_name: src
  username: src
os_migrate_src_region_name: regionOne
```

Notice that the parameters `project_domain_name` and `user_domain_name` are removed and the `auth_url` parameter points to the Keystone v2 endpoint.

Shortcuts

We will use the OS Migrate collection path and an ansible-playbook command with the following arguments routinely, so let’s save them as variables in the shell:

```
export
OSM_DIR=/home/migrator/.ansible/collections/ansible_collections/os_migrate/os_migrate
export OSM_CMD="ansible-playbook -v -i $OSM_DIR/localhost_inventory.yml -e @os-
migrate-vars.yml"
```

Pre-workload migration

Workloads require the support of several resources in a given cloud to operate properly. Some of these resources include networks, subnets, routers, router interfaces, security groups, and security group rules. The pre-workload migration process includes exporting these resources from the source cloud onto the migrator machine, the option to edit the resources if desired, and importing them into the destination cloud.

Exporting or importing resources is enabled by running the corresponding playbook from OS Migrate. Let’s look at a concrete example. To export the networks, run the "export_networks" playbook.

Export and import

To export the networks:

```
$OSM_CMD $OSM_DIR/playbooks/export_networks.yml
```

This will create networks.yml file in the data directory, similar to this:

Networks YAML example

```
os_migrate_version: 0.17.0
resources:
  - _info:
      availability_zones:
        - nova
      created_at: '2020-04-07T14:08:30Z'
      id: a1eb31f6-2cdc-4896-b582-8950dafa34aa
      project_id: 2f444c71265048f7a9d21f81db6f21a4
      qos_policy_id: null
      revision_number: 3
      status: ACTIVE
      subnet_ids:
        - a5052e10-5e00-432b-a826-29695677aca0
        - d450ffd0-972e-4398-ab49-6ba9e29e2499
      updated_at: '2020-04-07T14:08:34Z'
    params:
      availability_zone_hints: []
      description: ''
      dns_domain: null
      is_admin_state_up: true
      is_default: null
      is_port_security_enabled: true
      is_router_external: false
      is_shared: false
      is_vlan_transparent: null
      mtu: 1450
      name: osm_net
      provider_network_type: null
      provider_physical_network: null
      provider_segmentation_id: null
      qos_policy_name: null
      segments: null
    type: openstack.network.Network
```

You may edit the file as needed and then run the "import_networks" playbook to import the networks from this file into the destination cloud:

```
$OSM_CMD $OSM_DIR/playbooks/import_networks.yml
```

You can repeat this process for other resources like subnets, security groups, security group rules, routers, router interfaces, images and keypairs.

For a full list of available playbooks, run:

```
ls $OSM_DIR/playbooks
```

Diagrams

///TODO need to figure out these UMLs in the source .. figure:: ../images/render/pre-workload-migration-workflow.png :alt: Pre-workload Migration (workflow) :width: 50%

```
Pre-workload Migration (workflow)
```

a. figure:: ../images/render/pre-workload-migration-data-flow.png :alt: Pre-workload Migration (data flow) :width: 50%

   ```
   Pre-workload Migration (data flow)
   ```

Demo

///TODO: Video link `Pre-workload migration recorded demo <https://youtu.be/e7KXy5Hq4CMA>`_:

|Watch the video1|

Workload migration

Workload information is exported in a similar method to networks, security groups, etc. as in the previous sections. Run the "export_workloads" playbook, and edit the resulting workloads.yml as desired:

```
os_migrate_version: 0.17.0
resources:
- _info:
    addresses:
      external_network:
      - OS-EXT-IPS-MAC:mac_addr: fa:16:3e:98:19:a0
        OS-EXT-IPS:type: fixed
        addr: 10.19.2.41
        version: 4
    flavor_id: a96b2815-3525-4eea-9ab4-14ba58e17835
    id: 0025f062-f684-4e02-9da2-3219e011ec74
    status: SHUTOFF
  params:
    flavor_name: m1.small
    name: migration-vm
    security_group_names:
    - testing123
    - default
  type: openstack.compute.Server
```

Note that this playbook only extracts metadata about servers in the specified tenant - it does not download OpenStack volumes directly to the migration data directory. Data transfer is handled by the import_workloads playbook. The data is transfered directly between the clouds, meaning both clouds have to be running and reachable at the same time. The following sections describe the process in more detail.

Process Summary

This flowchart illustrates the high-level migration workflow, from a user’s point of view:

///TODO UML image .. figure:: ../images/render/workload-migration-workflow.png :alt: Workload migration (workflow) :width: 50%

```
Workload migration (workflow)
```

The process involves the deployment of a "conversion host" on source and destination clouds. A conversion host is an OpenStack server which will be used to transfer binary volume data from the source to the destination cloud. The conversion hosts are expected to be created from CentOS 9 or RHEL 8 cloud images.

The following diagram helps explain the need for a conversion host VM:

///TODO UML image .. figure:: ../images/render/workload-migration-data-flow.png :alt: Workload migration (data flow) :width: 80%

```
Workload migration (data flow)
```

This shows that volumes on the source and destination clouds are removed from their original VMs and attached to their respective conversion hosts, and then transferred over the network from the source conversion host to the destination. The tooling inside the conversion host migrates one server by automating these actions on the source and destination clouds:

Source Cloud:

- Detach volumes from the target server to migrate
- Attach the volumes to the source conversion host
- Export the volumes as block devices and wait for destination conversion host to connect

Destination Cloud:

- Create new volumes on the destination conversion host, one for each source volume
- Attach the new volumes to the destination conversion host
- Connect to the block devices exported by source conversion host, and copy the data to the new attached volumes
- Detach the volumes from the destination conversion host
- Create a new server using the new volumes

This method keeps broad compatibility with the various flavors and configurations of OpenStack using as much of an API-only approach as possible, while allowing the use of libguestfs-based tooling to minimize total data transfer.

Preparation

We'll put additional parameters into `os-migrate-vars.yml`:

```
os_migrate_conversion_external_network_name: public
os_migrate_conversion_flavor_name: m1.large
```

These define the flavor and external network we want to use for our conversion hosts.

By default the migration will use an image named `os_migrate_conv` for conversion hosts. Make sure this image exists in Glance on both clouds. Currently it should be a CentOS 9 Cloud Image or RHEL 8 KVM Guest Image

When using RHEL as conversion host, make sure to set the necessary RHEL Variables

Conversion host deployment

The conversion host deployment playbook creates the servers, installs additional required packages, and authorizes the destination conversion host to connect to the source conversion host for the actual data transfer.

```
$OSM_CMD $OSM_DIR/playbooks/deploy_conversion_hosts.yml
```

Export

Before migrating workloads, the destination cloud must have imported all other resources (networks, security groups, etc.) or the migration will fail. Matching named resources (including flavor names) must exist on the destination before the servers are created.

Export workload information with the export_workloads playbook. Each server listed in the resulting workloads.yml will be migrated, except for the one matching the name given to the source conversion host server.

```
$OSM_CMD $OSM_DIR/playbooks/export_workloads.yml
```

The resulting workloads.yml file will look similar to:

```
os_migrate_version: 0.17.0
resources:
- _info:
    created_at: '2020-11-12T17:55:40Z'
    flavor_id: cd6258f9-c34b-4a9c-a1e2-8cb81826781e
    id: af615f8c-378a-4a2e-be6a-b4d38a954242
    launched_at: '2020-11-12T17:56:00.000000'
    security_group_ids:
    - 1359ec88-4873-40d2-aa0b-18ad0588f107
    status: SHUTOFF
```

```
        updated_at: '2020-11-12T17:56:30Z'
        user_id: 48be0a2e86a84682b8e4992a65d39e3e
      _migration_params:
        boot_disk_copy: false
      params:
        availability_zone: nova
        config_drive: null
        description: osm_server
        disk_config: MANUAL
        flavor_ref:
          domain_name: null
          name: m1.xtiny
          project_name: null
        image_ref:
          domain_name: null
          name: cirros-0.4.0-x86_64-disk.img
          project_name: null
        key_name: osm_key
        metadata: {}
        name: osm_server
        ports:
        - _info:
            device_id: af615f8c-378a-4a2e-be6a-b4d38a954242
            device_owner: compute:nova
            id: cf5d73c3-089b-456b-abb9-dc5da988844e
          _migration_params: {}
          params:
            fixed_ips_refs:
            - ip_address: 192.168.20.7
              subnet_ref:
                domain_name: '%auth%'
                name: osm_subnet
                project_name: '%auth%'
            network_ref:
              domain_name: '%auth%'
              name: osm_net
              project_name: '%auth%'
          type: openstack.network.ServerPort
        scheduler_hints: null
        security_group_refs:
        - domain_name: '%auth%'
          name: osm_security_group
          project_name: '%auth%'
        tags: []
        user_data: null
      type: openstack.compute.Server
```

Migration parameters

///TODO next chapter, inc. link You can edit the exported `workloads.yml` to adjust desired properties for the servers which will be created in the destination cloud during migration. You can also edit migration parameters to control how a workload should be migrated. Refer to `Migration Parameters Guide <migration-params-guide.html>`_ for more information.

Ansible Variables

///TODO next chapter, inc. link In addition to the migration parameters in the resource YAML files, you can alter the behavior of OS Migrate via Ansible variables, e.g. to specify a subset of resources/workloads that will be exported or imported. Refer to the `Variables Guide <variables-guide.html>`_ for details.

Migration

Then run the import_workloads playbook to migrate the workloads:

```
$OSM_CMD $OSM_DIR/playbooks/import_workloads.yml
```

Any server marked "changed" should be successfully migrated to the destination cloud. Servers are "skipped" if they match the name or ID of the specified conversion host. If there is already an server on the destination matching the name of the current server, it will be marked "ok" and no extra work will be performed.

Cleanup of conversion hosts

When you are done migrating workloads in given tenants, delete their conversion hosts via the delete_conversion_hosts playbook:

```
$OSM_CMD $OSM_DIR/playbooks/delete_conversion_hosts.yml
```

Demo

Workload migration recorded demo

ifdev::context[:parent-context: planning]

4.4. How it works: Workload migration

The information described in this page is not necessary knowledge for using OS Migrate workload migration. However, knowing how the migration works under the hood may be useful for troubleshooting or forming deeper understanding of OS Migrate.

4.4.1. Data flow

The workload migration utilizes so called **conversion hosts** to transfer data from the source cloud to the destination cloud.

The conversion hosts are temporarily deployed in the source & destination projects. Being in the same project as the source (and destination, respectively) VMs ensures that the conversion hosts will have access to the data that needs to be migrated (snapshots and volumes).

a. figure:: ../images/render/workload-migration-data-flow.png :alt: Workload migration (data flow) :width: 80%

```
Workload migration (data flow)
```

4.4.2. Migration sequence

The `export_workloads.yml` playbook simply exports workload metadata into `workloads.yml`.

The actual main migration sequence happens inside `import_workloads.yml` playbook and the `import_workloads` role. The initial common steps are:

- The resources loaded from `workloads.yml` are validated.
- Resources are filtered according to `os_migrate_workloads_filter`.
- Reachability of source & destination conversion hosts is verified.

If you're defaulting to the default storage migration mode `data_copy` then the role starts iterating over all workloads that passed the filter. The steps performed for each workload (Nova Server) are:

- The `import_workload_prelim` module creates log and state files under `{{ os_migrate_data_dir }}/workload_logs`. It also takes care of skipping migration of VMs that already exist in the destination, and skipping of conversion hosts, should such migration be attempted.
- The `import_workload_dst_check` module checks whether migration prerequisites are satisfied in the destination cloud/project. This means verifying that resources which are referenced by name from the workload serialization can be de-referenced in the destination cloud. In other words, this verifies the networks, subnets etc., that the destination VM should be attached to, indeed exist in the destination cloud.
- If `os_migrate_workload_stop_before_migration` is `true`, the VM in the source cloud is stopped.
- The `import_workload_src_check` checks whether the source workload is ready to be migrated. This means verifying that the Nova Server is `SHUTOFF`.
- The `import_workload_export_volumes` module prepares data for transfer to the destination cloud:
    - If `boot_disk_copy` is `true`, a snapshot of the source VM is created, converted to a Cinder volume and attached to the source conversion host.
    - Additional Cinder volumes attached to the source VM are detached from it and attached to the source conversion host.
    - All VM's volumes (boot & additional) on the conversion host are exported as NBD drives, listening on localhost only.
    - The `import_workload_transfer_volumes` copies data from source to destination:
    - SSH port forwarding is created for the NBD drives of the source conversion host, so that they are accessible on the destination conversion host, again on localhost only. (The data transfer mechanism could be described as "NBD over SSH".)

    - Cinder volumes are created in the destination project for both the boot disk and additional volumes (as applicable). The destination volume sizes match the volume sizes in the source cloud. The volumes are attached to the destination conversion host.
    - Sparsification of the NBDs is performed, only for recognizable filesystems that the `virt-sparsify` tool supports. This significantly speeds up copying of empty space on supported filesystems.
    - Data is copied from the NBDs to the respective destination Cinder volumes.
    - SSH port forwarding for the NBDs are closed, and volumes are detached from the destination conversion host.
- The `import_workload_create_instance` creates new Nova server in the destination cloud according to the data from the resource serialization, and using the copied Cinder volumes as applicable.
- The `import_workload_src_cleanup` cleans up after the migration in the source cloud. It closes the NBD exports, detaches volumes from the conversion host, deletes the temporary boot disk snapshot volume and re-attaches any additional volumes back onto the source VM (as applicable).
- In case of failure during the migration, the `import_workload_src_cleanup` module is executed too, and an extra `import_workload_dst_failure_cleanup` module is executed, which aims to clean up failed partial migration from the destination cloud. (In case of successful migration, no further clean up is necessary in the destination cloud.)
    a. figure:: ../images/render/workload-migration-sequence.png :alt: Workload migration (sequence) :width: 80%

    ```
    Workload migration (sequence)
    ```

4.4.3. Migration Parameters Guide

What are migration parameters

Resource YAML files generated by OS Migrate when exporting are editable by the user. The parameters in `params` section of a resource generally specify direct resource properties, i.e. **what** the resource is like. Sometimes, a resource can be copied from source to destination in multiple different ways, each suitable for different use cases. To control **how** an individual resource should be migrated, there is another editable section in resource serializations, called `_migration_params`. The descriptions of the most important ones are in this guide.

Workload migration parameters

- `boot_disk_copy` controls how if the boot disk of the destination server is copied or re-imaged:
    - `false`: The destination server will be booted from a Glance image of the same name as the source server. (This is the default for servers which were booted from an image in the source cloud.)
    - `true`: The source server's boot disk will be copied into the destination as a volume, and the destination server will be created as boot-from-volume. (For servers which are already boot-

from-volume in the source cloud, this is the default and the only possible path.)

- `data_copy` controls storage migration modes for workloads:
  - `false`: The copying of data using os-migrate is skipped.
  - `true`: (Default) We result to using os-migrate for data copying.
- `boot_volume_params` controls new boot disk creation parameters in the destination, in case the source VM has no boot disk, but `boot_disk_copy` is `true`.

  There are a few pre-filled parameters, defaulted to `None`. Then `None` value means those parameters will not be specified when creating the boot volume, or in case of the `name` parameter, the default for workload migration will be used (prefix + VM name).

  When the source VM **does** have a boot volume already, do not use `boot_volume_params` to edit the destination creation parameters. Instead, edit the serialized volume in the `volumes` section of the workload's `params`.

- `boot_volume` controls new boot disk creation parameters for workload migrations with storage mode (data_copy) set to false.

  There are a few pre-filled parameters, defaulted to `None`. Then `None` value means those parameters will not be specified when creating the boot volume, or in case of the `name` parameter, the default for workload migration will be used (prefix + VM name).

  When the source VM **does** have a boot volume already, do not use `boot_volume_params` to edit the destination creation parameters. Instead, edit the serialized volume in the `volumes` section of the workload's `params`.

- `additional_volumes` any additonal volumes to be configured for workload migrations with storage mode (data_copy) set to false.
- `floating_ip_mode` controls whether and how floating IPs should be created for workloads:
  - `skip`: Do not create any floating IPs on the destination server.
  - `new`: Create a new floating IP (auto-assigned address).
  - `existing`: Assume the floating IP address as specified in the workload serialization is already assigned to the destination project, but not attached. Attach this floating IP. If this is not possible for some reason, fail.
  - `auto` (default): Attempt the `existing` method of floating IP assignment first, but should it fail, fall back to the `new` method instead.

  General remarks regarding floating IPs:

  In the `workloads.yml` export, each serialized floating IP contains a `fixed_ip_address` property, so a floating IP will be created on the port with this address. (When editing ports/fixed addresses of a workload, make sure to also edit the `fixed_ip_address` properties of its floating IPs accordingly.)

  It is important to note that the address of a newly created floating IP will be automatically selected by the cloud, it will not match the floating IP address of the source server. (In most cases, the floating IP ranges of src/dst clouds don't overlap anyway.)

4.5. OS Migrate Glossary

A comprehensive glossary of terms for operators learning about the OS Migrate project.

4.5.1. Core Concepts

OS Migrate Collection

An Ansible collection (`os_migrate.os_migrate`) that provides modules, roles, and playbooks for migrating OpenStack resources between clouds.

Resource

An OpenStack entity that can be migrated (networks, instances, flavors, images, etc.). Each resource type has its own export/import workflow.

Export Phase

The process of extracting resource definitions from a source OpenStack cloud and serializing them to YAML files.

Import Phase

The process of reading YAML resource files and creating corresponding resources in a destination OpenStack cloud.

Parallel Migration

A modernization strategy where a new OpenStack deployment runs alongside an existing one, with tenant resources migrated between them.

Idempotent Operations

All playbooks can be re-run safely; they won’t duplicate resources or cause conflicts.

4.5.2. Resource Types

Workloads

Running instances/VMs that are migrated from source to destination cloud, including their attached storage and network configuration.

Detached Volumes

Storage volumes not currently attached to any instance.

Flavors

Virtual machine templates that define CPU, memory, and disk specifications.

Images

Disk images used as templates for creating instances.

Networks

Virtual networks that provide connectivity between instances.

Subnets

IP address ranges within networks that define available IP addresses.

Routers

Virtual routers that provide connectivity between networks and external networks.

Router Interfaces

Connections between routers and subnets.

Security Groups

Firewall rule sets that control network traffic to instances.

Security Group Rules

Individual firewall rules within security groups.

Projects

OpenStack tenants that contain and isolate resources.

Users

OpenStack user accounts with authentication credentials.

Keypairs

SSH key pairs used for secure instance access.

User Project Role Assignments

Mappings that grant users specific roles within projects.

4.5.3. Migration Infrastructure

Conversion Host

A temporary VM created in the destination cloud to facilitate workload migration, particularly for disk image transfer.

Conversion Host Content

Software and configuration deployed on conversion hosts to enable migration functionality.

Migrator Host

The system where OS Migrate playbooks are executed, coordinating the migration process.

4.5.4. Data Management

Data Directory (`os_migrate_data_dir`)

Local filesystem location where exported YAML resource files are stored during migration.

Resource Filter (`os_migrate_<resource>_filter`)

Name-based filtering to selectively export/import specific resources rather than all resources of a type.

Serialization

The process of converting OpenStack SDK objects into OS Migrate's standardized YAML format.

Resource Validation

Checking that imported YAML files contain valid resource definitions before attempting import.

File Validation (`import_<resource>_validate_file`)

Optional validation step that verifies resource data integrity before import operations.

4.5.5. Resource Structure

params_from_sdk

Resource properties that are copied to the destination cloud (configuration, settings).

info_from_sdk

Resource properties that are NOT copied (UUIDs, timestamps, read-only data).

Migration Parameters

OS Migrate-specific settings that control migration behavior for each resource.

SDK Parameters

Parameters used when making OpenStack API calls to create or update resources.

Readonly SDK Parameters

Parameters that are allowed during resource creation but not during updates.

4.5.6. Authentication & Configuration

Source Auth (`os_migrate_src_auth`)

OpenStack authentication credentials for the source cloud.

Destination Auth (`os_migrate_dst_auth`)

OpenStack authentication credentials for the destination cloud.

clouds.yaml

OpenStack client configuration file containing cloud authentication details.

Source Cloud (`SRC_CLOUD`)

Environment variable identifying the source cloud configuration in clouds.yaml.

Destination Cloud (`DST_CLOUD`)

Environment variable identifying the destination cloud configuration in clouds.yaml.

4.5.7. Ansible Components

Export Roles

Ansible roles that call export modules and handle resource filtering (e.g., `export_networks`, `export_workloads`).

Import Roles

Ansible roles that validate data files and call import modules (e.g., `import_networks`, `import_workloads`).

Export Modules

Ansible modules that retrieve resources from source cloud and serialize to YAML (e.g., `export_flavor.py`).

Import Modules

Ansible modules that read YAML files and create resources in destination cloud (e.g., `import_flavor.py`).

Prelude Roles

Setup roles that prepare the environment:

- `prelude_src` - Source cloud preparation
- `prelude_dst` - Destination cloud preparation
- `prelude_common` - Common setup tasks

Resource Class

Python class that defines how OpenStack SDK objects are converted to/from OS Migrate format. All inherit from the base `Resource` class.

4.5.8. Development & Testing

Role Skeleton

Template structure for creating new export/import roles using the role-addition script.

Unit Tests

Tests that verify module logic and resource class behavior in isolation.

Functional Tests

Tests that verify roles and modules work correctly in integration scenarios.

End-to-End Tests (E2E)

Full migration workflow tests that validate complete export/import cycles.

Sanity Tests

Ansible collection sanity checks that verify code quality and standards compliance.

4.5.9. Container Environment

Container Engine

Configurable container runtime (Podman/Docker) used for development environment via `CONTAINER_ENGINE` variable.

Development Container

CentOS Stream 10 container with Python 3.12 where all OS Migrate commands execute.

Source Mount

The `/code` directory inside the container where the OS Migrate source code is mounted.

4.5.10. Build & Deployment

Collection Build

Process of packaging OS Migrate into an Ansible collection archive for distribution.

Galaxy Installation

Installing the OS Migrate collection via `ansible-galaxy collection install os_migrate.os_migrate`.

Dependencies

Required Ansible collections (`community.crypto`, `community.general`, `openstack.cloud`) and Python packages.

4.5.11. Migration Process

Three-Phase Migration

The standard workflow: Export → Transfer → Import.

Transfer Phase

Moving exported YAML data files from source environment to destination environment.

Validation Phase

Verifying that resources were created correctly and handling any migration errors.

Resource References

Cross-references between resources (e.g., instances referencing networks) that must be resolved during import.

5. OS Migrate Developer Documentation

6. OS Migrate Design

6.1. High-level development goals

- I/O-based. Fetch metadata and/or content from source cloud → write them as outputs → read them as inputs → push it to the destination cloud. This allows other tools to enter the phase between the "write output" and "read input".
  - E.g. arbitrary/custom yaml-parsing tools can be used as a smart filter to analyze the exported contents from source cloud and choose what (not) to import into the destination cloud.
  - Sometimes (e.g. in workload migration) only metadata may be written out and editable on the migrator machine, and the actual binary data is transferred in a direct path between the

clouds for performance reasons.

- Logging. When contributing code, don't forget about logging and how will information be presented to the user. Try to think about what can potentially fail and what kind of log output might help figuring out what went wrong, and on the contrary, what kind of log output would add clutter without much information value.
- Testing. Unit-test semantics of small parts wherever possible. Emphasize automated functional tests (end-to-end, talking to a real OpenStack backend). Functional tests are closest to the real use cases and provide at least basic level of comfort that the software does what it should.
    - Avoid excessive mocking in unit tests. Focus on writing unit tests for self-contained (pure) functions/methods, and structure the code so that as much as possible is written as pure functions/methods. When this is not possible, consider writing functional/end-to-end tests instead of unit tests.
- Always talking to OpenStack via API. The tool must be able to be deployed externally to both source and destination clouds. No looking at DBs or other hacks. If we hit a hurdle due to no-hacks approach, it could mean that OpenStack's capabilities for tenant-to-tenant content migration are lacking, and an RFE for OpenStack might be required.
- Idempotency where possible. When a command fails, it should be possible to retry with the same command.
- Whenever simplicity / understandability / clarity gets into conflict with convenience / ease-of-use, we prefer simplicity / understandability / clarity.
    - Prefer running several CLI commands to do something where each command is simple and human can enter the process by amending inputs/outputs e.g. with additional tools or a text editor, rather than one magical command which aims to satisfy all use cases and eventually turns out to satisfy very few, and tends to fail in mysterious ways with partial completion and limited re-runnability.
- OS Migrate is intended to be a building block for tenant migrations rather than a push-button solution. The assumption is that to cover needs of a particular tenant migration, a knowledgeable human is running OS Migrate manually and/or has tweaked it to their needs.

6.2. Challenges of the problem domain

- Generally, admins can view resources of all projects, but cannot create resources in those projects (unless they have a role in the project). To create a resource under a project, either credentials of the target non-admin user have to be used for importing, or the admin user has to be added into the project for the duration of the import.
    - This is not true for **some** resources. As of January 2020, e.g. networking resources can be created under arbitrary domain/project by admin using Networking API v2, but it's not the case for e.g. volumes, servers, keypairs. At least initially, we will assume the general scenario that we're running the migration as the tenant who owns the resources on both src/dst sides.
- OpenStack doesn't have strict requirements on resource naming (doesn't require non-empty, unique names). The migration tool will enforce names which are non-empty, and unique per resource type. This is to allow idempotent imports, which is necessary for retrying imports on failure. The tool should allow to export/serialize resources that are not properly named, but the

exported data should fail validation and be refused when fed into the import command.

  - In the future, if we implement running all migrations as admin (might need OpenStack RFEs) instead of the tenant, we'd perhaps require uniqueness per resource type per tenant, rather than just per resource type.

6.3. Basic Ansible workflow design

- The challenge and a goal here is to give meaningful Ansible log output for debugging. This means, for example, that we shouldn't have a single Ansible task (one module call) to export/import the whole tenant, and we should also consider not having a single task to export/import all resources of some type. Ideally, each resource export/import would have its own module call (own task in Ansible playbook), so if the export or import fails, we can easily tell which resource was the one that caused a failure.
  - We have a YAML file per resource type. Export modules are able to add a resource (idempotently) to an existing YAML file without affecting the rest. There is reusable code for this idempotence.
- Example workflow: export networks. Provided playbook.
  - Ansible task, provided: fetch metadata (at least name and ID, but perhaps the more the better for advanced filtering) of all networks that are visible for the tenant, register a list variable.
  - Ansible task(s) optionally added by user into our playbook on per-environment basis: filter the metadata according to custom needs, re-register the list var.
    - Eventually we may want to provide some hooks here, but initially we'd be fine with users simply editing the provided playbook.
  - Ansible task, provided: filter networks by names, either via exact matches or via regexes.
  - Anisble task, provided, calling our custom module: Iterate (`loop`) over the list of metadata, and call our module which will fetch the data and write a YAML with our defined format for Network resources. If necessary, do any data mangling here to satisfy the format requirements.
- Example workflow: transform networks.
  - Initially we will not provide any tooling here, but at this point the user should have a YAML file (or a bunch of them) with the serialized resources. They can use any automation (Ansible, yq, sed, …) to go through them and edit as they please before importing.
- Example workflow: import networks. Provided playbook.
  - Ansible task, provided: read the serialized YAML networks into memory, register a list variable.
  - Anisble task, provided: Iterate (`loop`) over the list of networks, and call an Ansible module to create each network. We'll want to create our own module here to deal with our file format specifics, but underneath we may be calling the community OpenStack Ansible module if that proves helpful.
- We may want to consider using `tags` on our tasks to allow some sub-executions. However, it may be that chunking up the code and using `import_playbook` might be more clear and safe to

use than `tags` in many cases. If we do happen to add `tags`, they shouldn't be added wildly, use cases should be thought through and documented for both users and developers. Tags need clarity and disciplined use in order to be helpful.

   - The above talks about tags in end-user code. We do use tags in our test suites, and it's much less sensitive there as we can change those tags any time without breaking user expectations.

6.4. Misc

- Naming conventions - underscores rather than hyphens in Ansible.
   - Github does not support underscores in organization URLs though. So we have repo named os-migrate/os-migrate, and inside we have os_migrate Ansible collection.
- Distribution - the preference is to distribute os-migrate via Ansible Galaxy.

7. Development Environment Setup

7.1. Prerequisites

- Local clone of the os-migrate git repo from https://github.com/os-migrate/os-migrate
- Podman and Buildah for running rootless development environment containers.

7.2. Running e2e tests

OS Migrate also has a suite of end to end, e2e, tests which tests a migration from one existing openstack deployment to another existing openstack deployment.

You can also test with a single existing openstack deployment migrating from one project to another.

These docs cover the testing of a single existing openstack deployment migrating from one project to another scenario.

The concepts and prerequisites are the same for other deployments.

7.2.1. Prerequisites for e2e test

- Source environment credentials
- Destination environment credentials
- Existing images in both source and destination environments
- Flavors
- Public network
- Space requirements
   - 2 images totalling 1.25 GB

◦ 1 volume totalling 1 GB in source environment
◦ 2 volumes totalling 6 GB in destination environment
◦ 2 VMs totalling 35 GB disk usage in each environment

Below are the steps required to satisfy the above requirements and run e2e tests in a test environment, migrating resources from one project to another.

7.2.2. Create source environment and destination environment projects and users

```
# Create the src user in the default domain with password 'redhat'
openstack user create --domain default --password redhat src

# Create the src project
openstack project create --domain default src

# Assign src user a 'member' role in the src project
openstack role add \
--user src --user-domain default \
--project src --project-domain default member

# Confirm role assignment was successful
openstack role assignment list --project src

# Create the dst user in the default domain with password 'redhat'
openstack user create --domain default --password redhat dst

# Create the dst project
openstack project create --domain default dst

# Assign dst user a 'member' role in the src project
openstack role add \
--user dst --user-domain default \
--project dst --project-domain default member

# Confirm role assignment was successful
openstack role assignment list --project dst
```

7.2.3. Create images

```
# Download images
wget https://cloud.centos.org/centos/9-stream/x86_64/images/CentOS-Stream-
GenericCloud-9-20230704.1.x86_64.qcow2
wget http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img

# Create images in glance from these downloads
openstack image create --public --disk-format qcow2 --file \
    CentOS-Stream-GenericCloud-9-20230704.1.x86_64.qcow2 CentOS-Stream-GenericCloud-9-
20230704.1.x86_64.qcow2
openstack image create --public --disk-format raw --file cirros-0.4.0-x86_64-disk.img
cirros-0.4.0-x86_64-disk.img
```

7.2.4. Create flavors

```
openstack flavor create --public \
--ram 256 --disk 5 --vcpus 1 --rxtx-factor 1 m1.xtiny

openstack flavor create --public \
--ram 2048 --disk 30 --vcpus 2 --rxtx-factor 1 m1.large
```

7.2.5. Create public network

If your OpenStack environment doesn't have a public network created yet, you'll need to create one. The parameters below should work if you're deploying your OpenStack environment with Infrared Virsh plugin. If you deployed using something else, you may need to adjust the parameters.

```
openstack network create \
     --mtu 1500 \
     --external \
     --provider-network-type flat \
     --provider-physical-network datacentre \
     public

openstack subnet create \
    --network public \
    --gateway 10.0.0.1 \
    --subnet-range 10.0.0.0/24 \
    --allocation-pool start=10.0.0.150,end=10.0.0.190 \
    public
```

7.2.6. Sample e2e config yaml using the above prerequisites

Also see https://github.com/os-migrate/os-migrate/blob/main/tests/e2e/tenant/scenario_variables.yml

Auth URLs and network names will change based on your environment.

```
os_migrate_src_auth:
  auth_url: http://10.0.0.131:5000/v3
  password: redhat
  project_domain_name: Default
  project_name: src
  user_domain_name: Default
  username: src
os_migrate_src_region_name: regionOne
os_migrate_dst_auth:
  auth_url: http://10.0.0.131:5000/v3
  password: redhat
  project_domain_name: Default
  project_name: dst
  user_domain_name: Default
  username: dst
os_migrate_dst_region_name: regionOne

os_migrate_data_dir: /root/os_migrate/local/migrate-data

os_migrate_conversion_host_ssh_user: cloud-user
os_migrate_src_conversion_external_network_name: nova
os_migrate_dst_conversion_external_network_name: nova
os_migrate_conversion_flavor_name: m1.large
os_migrate_conversion_image_name: CentOS-Stream-GenericCloud-8-20220913.0.x86_64.qcow2

os_migrate_src_osm_server_flavor: m1.xtiny
os_migrate_src_osm_server_image: cirros-0.4.0-x86_64-disk.img
os_migrate_src_osm_router_external_network: nova

os_migrate_src_validate_certs: False
os_migrate_dst_validate_certs: False

os_migrate_src_release: 16
os_migrate_dst_release: 16

os_migrate_src_conversion_net_mtu: 1400
os_migrate_dst_conversion_net_mtu: 1400
```

7.2.7. Run e2e test using the OS Migrate toolbox and the above config

Copy the above config to file `custom-config.yaml` in the `local` directory of your local `os-migrate` source.

Run the full test suite using the above config.

```
OS_MIGRATE_E2E_TEST_ARGS='-e @/root/os_migrate/local/custom-config.yaml' ./toolbox/run
make test-e2e-tenant
```

7.2.8. Expected output from successful e2e test run

```
PLAY RECAP
*************************************************************************************
******************
```

```
localhost                  : ok=318  changed=110  unreachable=0    failed=0
skipped=27   rescued=0    ignored=0
os_migrate_conv_dst        : ok=12   changed=5    unreachable=0    failed=0
skipped=3    rescued=0    ignored=0
os_migrate_conv_src        : ok=12   changed=5    unreachable=0    failed=0
skipped=3    rescued=0    ignored=0

Wednesday 21 July 2021  09:59:17 +0000 (0:00:03.419)       0:29:17.016 ********
===============================================================================
os_migrate.os_migrate.conversion_host_content : update all packages
--------------------------------------- 435.56s
os_migrate.os_migrate.import_workloads : transfer volumes to destination
---------------------------------- 101.72s
os_migrate.os_migrate.import_workloads : expose source volumes
--------------------------------------------- 66.67s
os_migrate.os_migrate.conversion_host_content : install content
-------------------------------------------- 62.35s
os_migrate.os_migrate.import_workloads : transfer volumes to destination
----------------------------------- 58.75s
os_migrate.os_migrate.import_workloads : clean up in the source cloud after migration
---------------------- 27.40s
os_migrate.os_migrate.import_workloads : expose source volumes
--------------------------------------------- 27.30s
Create osm_server
--------------------------------------------------------------------------------------
---- 24.71s
create osm_image
--------------------------------------------------------------------------------------
----- 23.86s
os_migrate.os_migrate.export_images : export image blobs
--------------------------------------------------- 23.80s
os_migrate.os_migrate.import_images : import images
-------------------------------------------------------- 23.69s
os_migrate.os_migrate.import_workloads : create destination instance
--------------------------------------- 23.30s
os_migrate.os_migrate.import_workloads : create destination instance
--------------------------------------- 21.93s
Create osm_server
--------------------------------------------------------------------------------------
---- 21.61s
os_migrate.os_migrate.import_workloads : clean up in the source cloud after migration
---------------------- 21.16s
os_migrate.os_migrate.conversion_host : create os_migrate conversion host
---------------------------------- 20.03s
Remove osm_server
--------------------------------------------------------------------------------------
---- 19.01s
os_migrate.os_migrate.conversion_host : create os_migrate conversion host
---------------------------------- 18.23s
Shutdown osm_server
--------------------------------------------------------------------------------------
```

```
-- 18.14s
Shutdown osm_server
-----------------------------------------------------------------------------------------
-- 17.88s
```

7.2.9. Optional tags to pass to e2e tests

There are a set of tags that can be used to filter which tasks to run during test.

- test_clean_before
- test_workload
- test_image_workload_boot_copy
- test_image_workload_boot_nocopy
- test_image_workload_boot_copy_clean
- test_clean_before
- test_pre_workload

7.2.10. Optional playbook variable

There is also an optional variable `test_clean_conversion_hosts_after` which can be set to `false` if you do not wish to clean up conversion hosts after test is complete.

7.2.11. Environment variables

The following environment variables can be used when running e2e tests.

- `OS_MIGRATE_E2E_TEST_ARGS`: All of the above tags and playbook variables can be set using the `OS_MIGRATE_E2E_TEST_ARGS` environment variable. This variable is also used to pass in the playbook custom config file. eg:

  ```
  `OS_MIGRATE_E2E_TEST_ARGS='-e @/root/os_migrate/local/custom-config.yaml \
  --tags test_clean_before,test_workload --skip-tags test_clean_after -e
  test_clean_conversion_hosts_after=false'`
  ```

- `ROOT_DIR`: Absolute directory path to OS Migrate source. When not set the default when run using OS Migrate developer toolbox this is set to `/root/os_migrate`.
- `OS_MIGRATE`: Absolute directory path to the OS Migrate ansible collection. When not set the default when run using os-migrate developer toolbox this is set to `/root/.ansible/collections/ansible_collections/os_migrate/os_migrate`.

8. General Contribution Guidelines

8.1. Commit Messages for Changelog

OS Migrate uses commit messages for automated generation of project changelog. For every pull request we request contributors to be compliant with the following commit message notation.

8.1.1. Format

```
<type>: <summary>

<body>
```

Accepted `<type>` values:

- `new` - newly implemented user-facing features
- `chg` - changes in existing user-facing features
- `fix` - user-facing bugfixes
- `oth` - other changes which users should know about
- `dev` - any developer-facing changes, regardless of new/chg/fix status

8.1.2. Keeping Changelog Clean

If the commit message subject starts with `dev:`, it will be omitted when rendering the changelog.

Using this convention is important to keep the changelog document concise and focused on user-facing changes. Any developer-facing changes (developer environment, CI, developer-only docs) or miniature "cosmetic" edits should be tagged as `dev`.

8.1.3. Summary (First Line)

The first line should not be longer than 75 characters, the second line is always blank and other lines should be wrapped at 80 characters.

8.1.4. Message Body

Uses the imperative, present tense: "change" not "changed" nor "changes" and includes motivation for the change and contrasts with previous behavior. Think how the commit message will appear in the project changelog.

9. Contributing Code

As an open source project, OS Migrate welcomes contributions from the community at large. The following guide provides information on how to add a new role to the project and where additional testing or documentation artifacts should be added. This isn't an exhaustive reference and is a living document subject to change as needed when the project formalizes any practice or pattern.

9.1. Adding a New Role

The most common contribution to OS Migrate is adding a new role to support the export or import of resources. The construction of the roles will likely follow a similar pattern. Each role also has specific unit and functional test requirements. The most common pattern for adding a new role will follow these steps:

- Add a resource class to `os_migrate/plugins/module_utils` that inherits from `ansible_collections.os_migrate.os_migrate.plugins.module_utils.resource.Resource`
- Add a unit test to `os_migrate/tests/unit` to test serializing and de-serializing data, and any other functionality your class may have.
- Create a new module in `os_migrate/plugins/modules` to facilitate the import or export of the resource.
- Add a new playbook to `os_migrate/playbooks`.
- Add a new role to `os_migrate/roles`.
- Add functional tests to `tests/func` that test primary use, idempotency, and updates as a minimum. Additional tests as necessary.
- Add documentation within classes and roles/playbooks as required. Update developer or user documentation as needed.

9.1.1. Creating the role skeleton automatically

Adding roles into os-migrate can be easily achieved by creating the role structure skeleton automatically.

From the repository root directory execute:

```
ansible-playbook \
   -i 'localhost,' \
   ./scripts/role-addition.yml \
   -e ansible_connection=local \
   -e role_name=import_example_resource
```

This command will generate the role, the default variables file, and the documentation stub.

9.1.2. Resources

Resources are the primary data structures that are exported from and imported to clouds. In `os_migrate/plugins/module_utils/resource.py`, there is a `Resource` class provided for your new resource to inherit from. It provides a way to quickly build an organized class that uses openstacksdk to retrieve, create and update data within a connected OpenStack tenant. See how other classes in the `module_utils` directory inherit from `Resource` for inspiration.

Two very important properties are the 'params' and '_info', as these are the primary data fields that will be exported and imported. At a minimum, the name property of any resource should be in params_from_sdk, and most likely addresses should be there too. The difference between info and params is:

- A property that would be an '_info' type is something that we don’t want or cannot "make the same" in the destination cloud. For example, typically UUIDs can’t be the same between two tenants so an 'id' property should remain in info.
- A property that would be in a 'params' type are things that we do want to copy to the destination cloud. These typically also get looked at when we’re making sure the import behavior is idempotent. I.e. when we re-run import playbooks, things that already got imported before are not attempted to be imported again.

9.1.3. Export Roles

In the `defaults/main.yml` file, at a minimum you will need to define the `os_migrate_[resource]_filter` variable to support filtering of resources by the name property.

In the `meta/main.yml` file you will likely just need to add the following default content:

```
---
galaxy_info:
  author: os-migrate
  description: os-migrate resource
  company: Red Hat
  license: Apache-2.0
  min_ansible_version: "2.9"
  platforms:
    - name: Fedora
      versions:
        - "34"
  galaxy_tags: ["osmigrate"]
dependencies:
  - role: os_migrate.os_migrate.prelude_src
```

In the `tasks/main.yml` file, add your Ansible tasks for exporting the data. A common pattern is retrieving data from an OpenStack tenant via a cloud module, creating a collection of name/id pairs for export, filtering the names for specific resources, and then calling the module you created for export.

9.1.4. Import Roles

In the `defaults/main.yml` file, at a minimum you will need to define the same `os_migrate_[resource]filter variable as with export, and import[resource]_validate_file: true` variable to set whether or not the import data file should be validated for this role. In most cases, it should be set to `true`.

In the `meta/main.yml` file you will likely just need to add the following default content:

```
---
```

```yaml
galaxy_info:
  author: os-migrate
  description: os-migrate resource
  company: Red Hat
  license: Apache-2.0
  min_ansible_version: "2.9"
  platforms:
    - name: Fedora
      versions:
        - "34"
  galaxy_tags: ["osmigrate"]
dependencies:
  - role: os_migrate.os_migrate.prelude_dst
```

In the `tasks/main.yml` file, add your Ansible tasks for importing the data. A common pattern is validating the data file created by the associated export role, reading the data file and then calling the module you created for import.

9.2. Writing Tests

For newly implemented resources, ensure comprehensive test coverage by following this checklist:

9.2.1. Functional Tests

- Ensure both import and export functionalities are tested, not just idempotency.
- Include tests for admin-only resources, ensuring they are renamed before import.
- Test resources as a tenant whenever possible to ensure broader coverage.
- For resources with special properties like `links` or `extra_specs`, write detailed tests inspecting these properties closely.

9.2.2. Unit Tests

- Write unit tests for each new module created, focusing on the logic within the module.
- Ensure that edge cases and error handling paths are covered in unit tests.

9.2.3. Integration Tests

- Add integration tests that cover the entire process of exporting and then importing the resource, verifying the integrity and consistency of the data.
- Verify that the resource behaves as expected in the context of the os-migrate ecosystem.

9.2.4. Documentation and Examples

- In the `DOCUMENTATION` constant of each module, include examples of how to use the module in a playbook.
- Ensure that the `README.md` file for the new role is comprehensive, covering the role's purpose,

usage, and any dependencies.

9.2.5. Test Execution in CI

- Confirm that all tests are executed in the Continuous Integration (CI) environment before merging.
- If a feature is merged without proper tests due to specific circumstances, create a technical debt tracking card to follow up.

9.2.6. Review and Inspection

- Conduct thorough reviews, especially when introducing new resources, to catch potential issues early.
- Implement policies or checklists in development documentation to ensure test soundness and coverage.

9.2.7. Location of Tests

- Place functional and integration tests in the `tests/e2e` or `tests/func` directory respectively, following the existing structure for similar resources.
- Unit tests should reside alongside the modules they are testing, in the `os_migrate/tests/` directory.

9.2.8. Special Considerations

- For resources that are only accessible by admin users, ensure tests reflect this by running them with appropriate permissions.
- Address any known issues from previous retrospectives, such as fixing the handling of Nova keypairs or ensuring resources are tested in tenant context.

9.2.9. Necessary Documentation

If this is your first time adding a pull request to the os-migrate repository, add your author information to `galaxy.yml`.

In each Ansible module in `os_migrate/plugins/modules`, there is a `DOCUMENTATION` constant where you must provide standard documentation on what the module does and an example of how you would use it in a playbook.

Each new role must have a `README.md` file as a requirement for Ansible Galaxy publishing.

10. Contributing Documentation

If you notice missing or errorneous documentation, you can either create an issue on Github, or you can create a pull request with the desired documentation changes.

Documentation sources in the repository:

- The majority of documentation is located under `docs/src` folder.
- Individual roles have a readme file under `os_migrate/roles/<ROLE>/README.md` (Ansible Galaxy requirement).
- Modules are documented directly in their Python file `os_migrate/plugins/modules/<MODULE>.py` (Ansible convention).

10.1. Rendering the Documentation

After you make your change and before you submit a pull request, it's good to verify that the changes you made are getting rendered into HTML correctly.

If you haven't yet built a toolbox container for OS Migrate development, do so now:

```
make toolbox-build
```

To build the documentation, run:

```
./toolbox/run make docs
```

You can inspect the rendered documentation by opening `docs/src/_build/html/index.html` in your web browser.

11. 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.

11.1. 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.

11.2. Available AEE Images

11.2.1. 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

11.2.2. 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

11.3. Building AEE Images

11.3.1. 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:

```bash
# 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
```

11.3.2. 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
```

11.3.3. 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
```

11.3.4. 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)

11.3.5. 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:

```yaml
# 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:

```yaml
# 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:

```yaml
# 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
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 }}
```

11.4. Using AEE Images

11.4.1. 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
```

11.4.2. 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
```

11.4.3. 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

11.5. AEE Configuration

11.5.1. 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:

```yaml
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
```

11.5.2. 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
```

11.6. Troubleshooting

11.6.1. 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.

```yaml
- 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
```

11.6.2. 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>
```

11.6.3. 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

11.7. Maintenance

11.7.1. 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

11.7.2. 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

11.7.3. 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

11.8. Best Practices

11.8.1. 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

11.8.2. 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

11.8.3. 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

11.9. TODO

11.9.1. 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.

12. Releasing the OS Migrate collection

Set env vars with old and new versions:

```
OLD_VERSION=$(./toolbox/run bash -c 'cat os_migrate/galaxy.yml | shyaml get-value
version')
echo "OLD_VERSION=$OLD_VERSION"

NEW_VERSION="SOME.NEW.VERSION"
```

Edit the version in galaxy.yml and any hardcoded values in functional tests:

```
sed -i -e "s/^version: $OLD_VERSION/version: $NEW_VERSION/" ./os_migrate/galaxy.yml
grep -lr "os_migrate_version: $OLD_VERSION" ./tests | xargs sed -i -e
"s/^os_migrate_version: $OLD_VERSION/os_migrate_version: $NEW_VERSION/"
```

A build is required to update const.py:

```
./toolbox/run make
```

Create a pull request with these changes. Once it’s merged, check out the merge commit and release to galaxy:

```
git checkout $MERGED_COMMIT
./toolbox/run ./scripts/publish.sh -k <YOUR_GALAXY_TOKEN>
```

After a successful release, create a tag on the commit you built the release from, and push the tag to the upstream repo:

```
git tag -m "$NEW_VERSION" "$NEW_VERSION"
# assuming the os-migrate upstream repo is named 'upstream' in your repo clone
git push upstream --tags
```

If you’ve incremented "X" or "Y" in "X.Y.Z" version scheme, create also a stable branch to allow us to create ".Z" releases:

```
STABLE_VERSION=$(awk -F. '{ print $1 "." $2; }' <<<"$NEW_VERSION")
git checkout -b stable/$STABLE_VERSION
git push upstream stable/$STABLE_VERSION
```

13. Installing Vagrant directly on host (alternative path)

Normally you should run Vagrant containerized as the developer environment setup doc suggests. If you have a reason to install on bare metal instead, here are relevant instructions.

```
yum install https://releases.hashicorp.com/vagrant/2.4.1/vagrant-2.4.1-1.x86_64.rpm -y
vagrant --version
```

```
virsh pool-define-as vagrant dir - - - - "/var/lib/libvirt/images/vagrant"
virsh pool-list --all
virsh pool-build vagrant
virsh pool-start vagrant
virsh pool-autostart vagrant
virsh pool-info vagrant
```

We need to install the libvirt provider

```
yum groupinstall "Virtualization Host" -y
yum install libvirt-devel -y
vagrant plugin install vagrant-libvirt
systemctl restart libvirtd
```

Go to the toolbox/vagrant folder

```
cd toolbox/vagrant
```

```
vagrant box add --name fedora37 https://app.vagrantup.com/fedora/boxes/37-cloud-
base/versions/37.20221105.0/providers/libvirt/unknown/vagrant.box
```

Start the environment using the libvirt provider

```
vagrant up --provider libvirt
```

14. Documented modules in os-migrate

14.1. Contents

15. Module - auth_info

This module provides for the following ansible plugin:

- auth_info

```
:module: os_migrate/plugins/modules/auth_info.py
:documentation: true
:examples: true
```

16. Module - export_detached_volume

This module provides for the following ansible plugin:

- export_detached_volume

```
:module: os_migrate/plugins/modules/export_detached_volume.py
:documentation: true
:examples: true
```

17. Module - export_flavor

This module provides for the following ansible plugin:

- export_flavor

```
:module: os_migrate/plugins/modules/export_flavor.py
:documentation: true
:examples: true
```

18. Module - export_image_blob

This module provides for the following ansible plugin:

- export_image_blob

```
:module: os_migrate/plugins/modules/export_image_blob.py
:documentation: true
:examples: true
```

19. Module - export_image_meta

This module provides for the following ansible plugin:

- export_image_meta

```
:module: os_migrate/plugins/modules/export_image_meta.py
:documentation: true
:examples: true
```

20. Module - export_keypair

This module provides for the following ansible plugin:

- export_keypair

```
:module: os_migrate/plugins/modules/export_keypair.py
:documentation: true
:examples: true
```

21. Module - export_network

This module provides for the following ansible plugin:

- export_network

```
:module: os_migrate/plugins/modules/export_network.py
:documentation: true
:examples: true
```

22. Module - export_project

This module provides for the following ansible plugin:

- export_project

```
:module: os_migrate/plugins/modules/export_project.py
:documentation: true
:examples: true
```

23. Module - export_router

This module provides for the following ansible plugin:

- export_router

```
:module: os_migrate/plugins/modules/export_router.py
:documentation: true
:examples: true
```

24. Module - export_router_interfaces

This module provides for the following ansible plugin:

- export_router_interfaces

```
:module: os_migrate/plugins/modules/export_router_interfaces.py
:documentation: true
:examples: true
```

25. Module - export_security_group

This module provides for the following ansible plugin:

- export_security_group

```
:module: os_migrate/plugins/modules/export_security_group.py
:documentation: true
:examples: true
```

26. Module - export_security_group_rules

This module provides for the following ansible plugin:

- export_security_group_rules

```
:module: os_migrate/plugins/modules/export_security_group_rules.py
:documentation: true
:examples: true
```

27. Module - export_subnet

This module provides for the following ansible plugin:

- export_subnet

```
:module: os_migrate/plugins/modules/export_subnet.py
:documentation: true
:examples: true
```

28. Module - export_user

This module provides for the following ansible plugin:

- export_user

```
:module: os_migrate/plugins/modules/export_user.py
:documentation: true
:examples: true
```

29. Module - export_user_project_role_assignment

This module provides for the following ansible plugin:

- export_user_project_role_assignment

```
:module: os_migrate/plugins/modules/export_user_project_role_assignment.py
:documentation: true
:examples: true
```

30. Module - export_workload

This module provides for the following ansible plugin:

- export_workload

```
:module: os_migrate/plugins/modules/export_workload.py
:documentation: true
:examples: true
```

31. Module - import_flavor

This module provides for the following ansible plugin:

- import_flavor

```
:module: os_migrate/plugins/modules/import_flavor.py
:documentation: true
:examples: true
```

32. Module - import_image

This module provides for the following ansible plugin:

- import_image

```
:module: os_migrate/plugins/modules/import_image.py
:documentation: true
:examples: true
```

33. Module - import_keypair

This module provides for the following ansible plugin:

- import_keypair

```
:module: os_migrate/plugins/modules/import_keypair.py
:documentation: true
:examples: true
```

34. Module - import_network

This module provides for the following ansible plugin:

- import_network

```
:module: os_migrate/plugins/modules/import_network.py
:documentation: true
:examples: true
```

35. Module - import_project

This module provides for the following ansible plugin:

- import_project

```
:module: os_migrate/plugins/modules/import_project.py
:documentation: true
:examples: true
```

36. Module - import_router

This module provides for the following ansible plugin:

- import_router

```
:module: os_migrate/plugins/modules/import_router.py
:documentation: true
:examples: true
```

37. Module - import_router_interface

This module provides for the following ansible plugin:

- import_router_interface

```
:module: os_migrate/plugins/modules/import_router_interface.py
:documentation: true
:examples: true
```

38. Module - import_security_group

This module provides for the following ansible plugin:

- import_security_group

```
:module: os_migrate/plugins/modules/import_security_group.py
:documentation: true
:examples: true
```

39. Module - import_security_group_rule

This module provides for the following ansible plugin:

- import_security_group_rule

```
:module: os_migrate/plugins/modules/import_security_group_rule.py
:documentation: true
:examples: true
```

40. Module - import_subnet

This module provides for the following ansible plugin:

- import_subnet

```
:module: os_migrate/plugins/modules/import_subnet.py
:documentation: true
:examples: true
```

41. Module - import_user

This module provides for the following ansible plugin:

- import_user

```
:module: os_migrate/plugins/modules/import_user.py
:documentation: true
:examples: true
```

42. Module - import_user_project_role_assignment

This module provides for the following ansible plugin:

- import_user_project_role_assignment

```
:module: os_migrate/plugins/modules/import_user_project_role_assignment.py
:documentation: true
:examples: true
```

43. Module - import_volumes_export

This module provides for the following ansible plugin:

- import_volumes_export

```
:module: os_migrate/plugins/modules/import_volumes_export.py
:documentation: true
:examples: true
```

44. Module - import_volumes_src_cleanup

This module provides for the following ansible plugin:

- import_volumes_src_cleanup

```
:module: os_migrate/plugins/modules/import_volumes_src_cleanup.py
:documentation: true
:examples: true
```

45. Module - import_volumes_transfer

This module provides for the following ansible plugin:

- import_volumes_transfer

```
:module: os_migrate/plugins/modules/import_volumes_transfer.py
:documentation: true
:examples: true
```

46. Module - import_workload_create_instance

This module provides for the following ansible plugin:

- import_workload_create_instance

```
:module: os_migrate/plugins/modules/import_workload_create_instance.py
:documentation: true
:examples: true
```

47. Module - import_workload_dst_check

This module provides for the following ansible plugin:

- import_workload_dst_check

```
:module: os_migrate/plugins/modules/import_workload_dst_check.py
:documentation: true
:examples: true
```

48. Module - import_workload_dst_failure_cleanup

This module provides for the following ansible plugin:

- import_workload_dst_failure_cleanup

```
:module: os_migrate/plugins/modules/import_workload_dst_failure_cleanup.py
:documentation: true
:examples: true
```

49. Module - import_workload_export_volumes

This module provides for the following ansible plugin:

- import_workload_export_volumes

```
:module: os_migrate/plugins/modules/import_workload_export_volumes.py
:documentation: true
:examples: true
```

50. Module - import_workload_prelim

This module provides for the following ansible plugin:

- import_workload_prelim

```
:module: os_migrate/plugins/modules/import_workload_prelim.py
:documentation: true
```

```
:examples: true
```

51. Module - import_workload_src_check

This module provides for the following ansible plugin:

- import_workload_src_check

```
:module: os_migrate/plugins/modules/import_workload_src_check.py
:documentation: true
:examples: true
```

52. Module - import_workload_src_cleanup

This module provides for the following ansible plugin:

- import_workload_src_cleanup

```
:module: os_migrate/plugins/modules/import_workload_src_cleanup.py
:documentation: true
:examples: true
```

53. Module - import_workload_transfer_volumes

This module provides for the following ansible plugin:

- import_workload_transfer_volumes

```
:module: os_migrate/plugins/modules/import_workload_transfer_volumes.py
:documentation: true
:examples: true
```

54. Module - os_conversion_host_info

This module provides for the following ansible plugin:

- os_conversion_host_info

```
:module: os_migrate/plugins/modules/os_conversion_host_info.py
:documentation: true
```

```
:examples: true
```

55. Module - os_keypairs_info

This module provides for the following ansible plugin:

- os_keypairs_info

```
:module: os_migrate/plugins/modules/os_keypairs_info.py
:documentation: true
:examples: true
```

56. Module - os_role_assignments_info

This module provides for the following ansible plugin:

- os_role_assignments_info

```
:module: os_migrate/plugins/modules/os_role_assignments_info.py
:documentation: true
:examples: true
```

57. Module - os_routers_info

This module provides for the following ansible plugin:

- os_routers_info

```
:module: os_migrate/plugins/modules/os_routers_info.py
:documentation: true
:examples: true
```

58. Module - os_security_groups_info

This module provides for the following ansible plugin:

- os_security_groups_info

```
:module: os_migrate/plugins/modules/os_security_groups_info.py
:documentation: true
:examples: true
```

59. Module - read_resources

This module provides for the following ansible plugin:

- read_resources

```
:module: os_migrate/plugins/modules/read_resources.py
:documentation: true
:examples: true
```

60. Module - validate_resource_files

This module provides for the following ansible plugin:

- validate_resource_files

```
:module: os_migrate/plugins/modules/validate_resource_files.py
:documentation: true
:examples: true
```

61. Documented roles in os-migrate

61.1. Contents

62. Role - conversion_host

a. ansibleautoplugin:: :role: os_migrate/roles/conversion_host

63. Role - conversion_host_content

a. ansibleautoplugin:: :role: os_migrate/roles/conversion_host_content

64. Role - export_detached_volumes

a. ansibleautoplugin:: :role: os_migrate/roles/export_detached_volumes

65. Role - export_flavors

a. ansibleautoplugin:: :role: os_migrate/roles/export_flavors

66. Role - export_images

a. ansibleautoplugin:: :role: os_migrate/roles/export_images

67. Role - export_keypairs

a. ansibleautoplugin:: :role: os_migrate/roles/export_keypairs

68. Role - export_networks

a. ansibleautoplugin:: :role: os_migrate/roles/export_networks

69. Role - export_projects

a. ansibleautoplugin:: :role: os_migrate/roles/export_projects

70. Role - export_router_interfaces

a. ansibleautoplugin:: :role: os_migrate/roles/export_router_interfaces

71. Role - export_routers

a. ansibleautoplugin:: :role: os_migrate/roles/export_routers

72. Role - export_security_group_rules

a. ansibleautoplugin:: :role: os_migrate/roles/export_security_group_rules

73. Role - export_security_groups

a. ansibleautoplugin:: :role: os_migrate/roles/export_security_groups

74. Role - export_subnets

a. ansibleautoplugin:: :role: os_migrate/roles/export_subnets

75. Role - export_user_project_role_assignments

a. ansibleautoplugin:: :role: os_migrate/roles/export_user_project_role_assignments

76. Role - export_users

a. ansibleautoplugin:: :role: os_migrate/roles/export_users

77. Role - export_users_keypairs

This role is meant to be run with admin privileges. It imports Nova keypairs matching `os_migrate_keypairs_filter` for all users matching `os_migrate_users_filter`.

When using this role, make sure you have recent enough OpenStack SDK (0.57+).

a. ansibleautoplugin:: :role: os_migrate/roles/export_users_keypairs

78. Role - export_workloads

a. ansibleautoplugin:: :role: os_migrate/roles/export_workloads

79. Role - import_detached_volumes

a. ansibleautoplugin:: :role: os_migrate/roles/import_detached_volumes

80. Role - import_flavors

a. ansibleautoplugin:: :role: os_migrate/roles/import_flavors

81. Role - import_images

a. ansibleautoplugin:: :role: os_migrate/roles/import_images

82. Role - import_keypairs

a. ansibleautoplugin:: :role: os_migrate/roles/import_keypairs

83. Role - import_networks

a. ansibleautoplugin:: :role: os_migrate/roles/import_networks

84. Role - import_projects

a. ansibleautoplugin:: :role: os_migrate/roles/import_projects

85. Role - import_router_interfaces

a. ansibleautoplugin:: :role: os_migrate/roles/import_router_interfaces

86. Role - import_routers

a. ansibleautoplugin:: :role: os_migrate/roles/import_routers

87. Role - import_security_group_rules

a. ansibleautoplugin:: :role: os_migrate/roles/import_security_group_rules

88. Role - import_security_groups

a. ansibleautoplugin:: :role: os_migrate/roles/import_security_groups

89. Role - import_subnets

a. ansibleautoplugin:: :role: os_migrate/roles/import_subnets

90. Role - import_user_project_role_assignments

a. ansibleautoplugin:: :role: os_migrate/roles/import_user_project_role_assignments

91. Role - import_users

a. ansibleautoplugin:: :role: os_migrate/roles/import_users

92. Role - import_users_keypairs

This role is meant to be run with admin privileges. It exports Nova keypairs matching `os_migrate_keypairs_filter` for all users matching `os_migrate_users_filter`.

When using this role, make sure you have recent enough OpenStack SDK (0.57+).

a. ansibleautoplugin:: :role: os_migrate/roles/import_users_keypairs

93. Role - import_workloads

a. ansibleautoplugin:: :role: os_migrate/roles/import_workloads

94. Role - prelude_common

a. ansibleautoplugin:: :role: os_migrate/roles/prelude_common

95. Role - prelude_dst

a. ansibleautoplugin:: :role: os_migrate/roles/prelude_dst

96. Role - prelude_src

a. ansibleautoplugin:: :role: os_migrate/roles/prelude_src

97. Role - validate_data_dir

a. ansibleautoplugin:: :role: os_migrate/roles/validate_data_dir

98. Role - validate_resource_files

a. ansibleautoplugin:: :role: os_migrate/roles/validate_resource_files