



Openshift 4 Disconnected Collection

Overview

The why

- The goal of this repository is to provide a standardized workflow for deploying OpenShift 4 in a disconnected environment using an agent-based installer.

The how

- The idea is to have all variables set in `group_vars/all`, with some defaults set in roles. The defaults in roles are only applied in case they are not set in `group_vars`.

Variables

- `cluster_version`: This variable controls the version of the cluster and allows for the pulling of operators and additional images (for the cluster) via `oc-mirror`. This requires an active subscription with a pull secret in `~/.docker/config.json`. It will pull all the required binaries along with some additional nice-to-haves (such as Butane).
- `cluster_deployment`: This variable controls how the cluster is deployed. It will generate the `install-config` and `agent-config` based on configured variables.
- `static` this is used to set things that very rarely change or set variables defined in other vars, that are built on top of other variables

Some assumptions include:

- SSH keys will not exist.
 - SSL certificates for the hosted registry will be generated, which will run on the disconnected host.
- Post-Install Configuration

Goals

- The current goal for post-install configuration is to deploy ArgoCD and have it manage the configuration and enforcement.
- To assist with this, a local Git repository will be configured as described in this article.

Status

- This is a work in progress, and we will continue to make enhancements over time.

Requirements (disconnected system)

- Packages:
 - `nmstate`
 - `openssl`
 - `httpd-tools`
- Ansible-collections
 - `dellemc.openmanage`

Additional Info:

View notes on recommended procedures for vm migrations

Thanks:

This started as as a fork from hyperkineticnerd/ansible-ocp4-agent, but we have since diverged quite a bit away, but still would like to give credit.



Red Hat OpenShift Installation Guide

This guide covers the process for installing Red Hat OpenShift 4.15+ on Red Hat Enterprise Linux 9 with options for FIPS and STIG configurations. The instructions support deployment on both connected and disconnected environments.

Prerequisites

RHEL Install

RHEL 9 Installation: Install Red Hat Enterprise Linux 9 and register it with your Red Hat account (if connected to the internet).

NOTE: If the goal is to have a FIPS-enabled cluster, the bastion host has to be FIPS-enabled as well. If you don't need FIPS, you can ignore the following configurations.

Environment Configurations:

NOTE: These configurations can be done post install. Changes to `usbguard/sysctl.conf` will require a reboot, while `fapolicyd` will only require restart on the service.

1. Ensure **FIPS** mode is enabled for RHEL 9.

- Enable FIPS during installation; or
- Enable FIPS post-installation

2. Configure **STIG** compliance as needed

3. Configure **fapolicyd** for Ansible Playbooks:

- Allow regular users to run Ansible playbooks by creating a new file at `/etc/fapolicyd/rules.d/22-ansible.rules` with the following contents:

```
allow perm=any uid=1000 : dir=/home/user/.ansible
allow perm=any uid=1000 : dir=/home/user/.cache/agent
allow perm=any uid=1000 : dir=/usr/share/git-core/templates/hooks
allow perm=any uid=1000 : dir=/pods
allow perm=any uid=1000 : dir=/usr/bin
allow perm=any uid=0,1000 : dir=/tmp
```

NOTE: This was what was used for testing, you can use Group Identifier (GID) or User Identifier (ID). I used UID because it more narrow scoped than GID. Policy and best judgement should dictate which route you do

4. Adjust User Namespace Limits for Registry Pod:

- Increase the `user.max_user_namespaces` setting to enable the registry pod to run as a non-root user. Update `/etc/sysctl.conf` as follows:

```
# Per CCE-83956-3: Set user.max_user_namespaces = 0 in /etc/sysctl.conf
user.max_user_namespaces = 5
```

5. Enable Access to External USB Devices (for Disconnected Environments):

- Add the following commands to the `%post` section in your kickstart file:

```
systemctl disable usbguard
sed -i 's/black/\#black/g' /etc/modprobe.d/usb-storage.conf
sed -i 's/install/\#install/g' /etc/modprobe.d/usb-storage.conf
```

6. Install Ansible/Podman:

```
sudo dnf install -y ansible-core container-tools
```

Running the Automation

Getting Collection installed (connected side or connected cluster)

1. Clone this Repository:

```
git clone https://github.com/cjnovak98/ocp4-disconnected-config
```

2. Navigate to the Playbooks Directory: `bash` `cd ocp4-disconnected-config/playbooks`

3. Install Required Ansible Collections(1st time/connected system):

NOTE: This will install disconnected-collection and should be run on a fresh system, before anything else has been run.

```
ansible-playbook initial-ansible-collection.yml
```

or

```
./initial-ansible-collection.yml
```

Directory Structure Assumptions

Ensure that the content resides in `/pods/content` (definable in `playbooks/group_vars/all/cluster-deployment.yml`). It is assumed this is the transferable media when running connected side for a disconnected cluster and disconnected config is `/pods/content/ansible`

Run Content Gathering Playbook to Prepare Disconnected Environments:

NOTE: Ensure a valid pull secret exists. You can get your pull secret from <https://console.redhat.com/openshift/create/local> and store it in `~/.docker/config` on the host where you're running the automation.

Update Environment Variables Modify `group_vars/all/cluster-version.yml` to customize it for your environment. Key variables include:

- `common_openshift_release`: Target cluster version
- `common_previous_openshift_release`: Previous version, used for updates
- `common_trident_version`: trident target version
- `content_generation_disconnected_operators`: Array of all Catalog sources Openshift will use
 - `redhat-operator-index`: Name of this catalog source
 - * `name`: Name of any operator to pull within this catalog source
 - `channel`: specific version of operator to pull on
 - `redhat-marketplace-index`:
 - `community-operator-index`:
 - `certified-operator-index`:
- `content_generation_additionalImages`: an array of additional images to pull

Example group_vars/all/cluster-version.yml

```
common_openshift_release: 4.16.4
common_previous_openshift_release: 4.16.3
common_trident_version: v24.06.0
```

```

# Disconnected Images

#content_generation_pull_mirror: true

## Operators To be mirrored, based on index they are in
content_generation_disconnected_operators:
  - redhat-operator-index:
    - name: kubernetes-nmstate-operator
    - name: kubevirt-hyperconverged
  - redhat-marketplace-index:
  - community-operator-index:
  - certified-operator-index:

## additionalImages for oc-mirror to pull. Include version
content_generation_additionalImages:
  - name: registry.redhat.io/ubi8/ubi:latest
  - name: registry.redhat.io/ubi9/ubi:latest
  - name: quay.io/noseka1/toolbox-container:full

```

NOTE: for the operator portion of cluster-versions, if nothing is defined as an operator for a catalog source, the entire source is omitted, it is the same design for channel, however the latest is pulled vs being omitted

Run the automation If you are deploying on a disconnected system then you will first need to gather all of the openshift content on a machine that has internet connection and transfer it over. There is a playbook that you can run which will gather the appropriate content:

```
ansible-playbook -K gather-content.yml
```

or

```
./gather-content.yml
```

Once you have the content downloaded, transfer it to your disconnected machine and put in the content directory (i.e. /pods/content), you will also need the ocp4-disconnected-config folder used to run this playbook

Install the Collection on a Disconnected Machine

You can either transfer the content from the media used to get it to the disconnected side, or update playbooks/group_vars/all/cluster-deployment.yml to point to the mount point of the media

Example: For testing, im running from a 1TB external drive, mounted in /pods/content and the directory structure was:

```
ls /pods/content
bin  downloads  images  ocp4-disconnected-config
```

Install required Ansible Collection from the previous step:

NOTE: This will install need the media to be mounted, the group vars set, and be in the ocp4-disconnected-config/playbooks directory

```
ansible-playbook disconnected-ansible-collection.yml
```

or

```
./disconnected-ansible-collection.yml
```

Run the Deployment Playbook

Update Environment Variables

- Modify `group_vars/all/cluster-deployment.yml` to customize it for your environment. Key variables include:
 - `common_openshift_dir`: Directory for pulled content to live.
 - `common_connected_cluster`: Set to `true` if the cluster itself is connected.
 - `mirror_content_pull_mirror`: Set to `true` to pull content (set to `false` for disconnected environments).
 - `common_fips_enabled`: `true` if the host is stigged and FIPS is enabled.
 - `common_cluster_domain`: Top-Level Domain (TLD) for the cluster.
 - `common_ip_space`: First three octets of the IP address range for both the bastion and cluster.
 - `common_nodes`: Details of nodes, including name, last octet of the node IP, and MAC addresses.
 - `idrac_user` and `idrac_password`: iDRAC credentials.
 - `idrac`: Node name and IP of the node's iDRAC.

Example playbooks/group_vars/all/cluster-deployment.yml

`common_git_repos:`

- `"https://github.com/cjnovak98/ocp4-disconnected-collection.git"`
- `"https://github.com/cjnovak98/ocp4-disconnected-config"`

`common_openshift_dir: /pods/content`

`common_connected_cluster: false`

`mirror_content_pull_mirror: false`

`common_openshift_interface: ens1`

`common_fips_enabled: true`

`common_cluster_domain: example.com`

`common_ip_space: 192.168.122`

`common_nodes:`

- `name: master-0`
`ip: '42'`
`mac: 52:54:00:8d:13:77`
- `name: worker-0`
`ip: '43'`
`mac: 52:54:00:8d:13:78`

iDRAC

#idrac_user: test

#idrac_password: tester

#media_share_idracs:

- # - name: master-0*
ip: '{{ ip_space }}.5'
- # - name: master-1*
ip: '{{ ip_space }}.6'
- # - name: master-2*
ip: '{{ ip_space }}.7'
- # - name: worker-0*
ip: '{{ ip_space }}.8'

NOTE: if cluster hardware supports idrac, uncomment to idrac vars.

For both connected and disconnected clusters, deploy the cluster by running:

```
ansible-playbook -K deploy-cluster.yml
```

or

```
./deploy-cluster.yml
```

Fix DNS for disconnected cluster without a DNS Server

If you are running a disconnected server (or at least on a network with no DNS), We recommend that you now point your bastion at master-0 node to be able to resolve cluster resources (ingress/api)

1. Open nmtui Interface:

Run the following command in your terminal:

```
sudo nmtui
```

2. Select “Edit a Connection”:

Use the arrow keys to navigate to Edit a connection and press Enter.

3. Choose Your Interface:

Highlight the interface that lives in the same IP space and VLAN as your soon to be deployed cluster (e.g. eth0, or eth0.670).

Press Enter to edit the selected connection.

4. Edit IPv4:

Use the arrow keys to navigate to IPv4 CONFIGURATION.

5. Add DNS Server:

Navigate to the DNS servers field.

Enter the IP of master-0.

6. Save Changes:

Use the arrow keys to navigate to OK and press Enter to save the settings.

7. Activate the Connection:

Navigate back to the main nmtui menu and select Activate a connection.

Highlight the interface you just edited, and press Enter to deactivate and reactivate the connection, applying the changes.

8. Verify the DNS Settings:

Run the following command to ensure the DNS server is correctly applied:

```
nmcli dev show <interface_name> | grep IP4.DNS
```

Replace <interface_name> with the name of your network interface.

9. Test with oc:

You can test resolution with oc by running the following:

```
oc whoami
```

It should return:

```
sysadmin
```

Add an additional node

Check Variables Validate the variables defined from the **Run the Deployment Playbook** portion above

NOTE: Double check the `add_worker_disk_type: sda` playbooks/group_vars/all/cluster-deployment.yml matches the actual disk type of the target nodes; i.e. vda (virtual drives), sda (sata drives), or nvme drives.

Run the playbook

```
ansible-playbook -K create-worker-for-single-node.yml
```

or

```
./create-worker-for-single-node.yml
```

NOTE: DNS needs to be functional as defined in the previous section, or it will fail when pulling the worker ignition file from the “cluster”.

OCP GitOps Day 1

This helm chart is in charge of some basic setup after the initial deployment of OpenShift.

The idea behind this is that without OpenShift GitOps, working with an easily observable and maintainable infrastructure as code setup becomes hard. This template essentially just sets up that GitOps installation to where is useful for the next steps (day2) where the bulk of the cluster configuration is performed.

This helm chart currently is responsible for the initial deployment and configuration of the OpenShift GitOps operator, including the automated setup of the ArgoCD application for the ‘Day 2’ operations.

Values

In order to properly run this chart, you will need to either modify the existing values.yaml file or create your own values file specifying the values relevant to your setup.

This can be built by copying the existing values.yaml, and adjusting accordingly.

Mainly, you will need the IP of the system you ran the Ansible from, which is where resources are hosted for the disconnected install, as well as for continued GitOps work. You will also need the ssh private key of the user that was used to run the Ansible, as that is in turn used to access the Git repository for the aforementioned GitOps. Example values file:

gitOps:

```
#the namespace where the argocd instance will reside, defaults to openshift-gitops since that is the
namespace: openshift-gitops
#This source variable represents the catalog source where the operator lives. For disconnected instal
source: cs-redhat-operator-index
```

gitRepoTemplate:

```
# The URL of the repository setup in the Ansible step. replace the IP with that of your syscon
url: git@<ip of syscon>:repos/ocp4-disconnected-config
# The SSH key of the user that ran the Ansible step, used to connect to the git repo. Be sure to incl
sshPrivateKey: |
  -----BEGIN OPENSSH PRIVATE KEY-----
  -----END OPENSSH PRIVATE KEY-----
```

Usage

To apply this template to your cluster, simply run the helm install command:


```
helm install gitops-day1 . --values=./your/values.yaml
```

This applies the yaml to the cluster, and will perform the day 1 operations. Note that it might take a few minutes for everything to come up and be ready, so be patient. You can tell the installation has finished and was successful by navigating to ArgoCD (top right apps menu in OCP web ui -> ArgoCD), clicking ‘Log in via OpenShift’, and seeing the ArgoCD dashboard. You should see the cluster-day2 app already loaded and attempting to sync. If not you can create the day 2 application manually. You should also see your git repository that’s hosted on the syscon by navigating to Settings (on the left) -> Repositories. You should see your gitOps git repo in the list, with a green checkmark indicating success.

You can also use the following command to output the finished product without actually applying it to OpenShift, in order to debug and check:

```
helm template test . --debug | less
```

OCP 4 Disconnected Config Day2 Helm Chart

This chart applies the cluster settings and resources necessary for day to day operations in the target environment. It is intended to be deployed via OpenShift GitOps as part of a infrastructure as code process.

In this case, comments have been provided in both the default `values.yaml` and each of the `.yaml` files in the `templates/` directory to help explain each component.

Deployment

Configuring the Template

To use, you will simply add an application in ArgoCD (Openshift GitOps), and supply this repo/directory, as well as the values file customized for your environment.

Since the intent is to have ArgoCD automatically sync your configurations, it is important to first ensure you have prepared the appropriate values for your environment. When creating the application manually in argoCD you have the option to override the values but it is recommended to maintain the values in a file that can be managed and updated as needed. This method also allows for a more automated method of creating the ArgoCD application. To do this, it is recommended that you copy the default values file found at: `./helm/openshift-gitops-day2/values.yaml` to a separate file, like `./helm/openshift-gitops-day2/day2-values-prod.yaml`.

This template expects the following cluster specific configurations for the following in your helm values:

1. Cluster-wide certificates
2. Networking configurations for each node in your cluster
3. LDAP sync configuration
4. Chrony configuration
5. Storage configuration

The example values file structure can be seen here

Configuring ArgoCD (Openshift Gitops)

To do this you can run the following command:

```
oc apply -f ./cluster-day2-app.yaml
```

To create the ArgoCD application manually you can use similar values to the below:

- Application Name: `cluster-day2`
- Project Name: `default`
- Sync Policy: `Automatic`
- Source:

- Repository URL: Select the `ocp4-disconnected-config` repo running on the bastion host
- Revision: `HEAD` (or override to your desired branch)
- Path: `./helm/openshift-gitops-day2`
- Destination:
 - Cluster URL: `https://kubernetes.default.svc` # Use this value to target the same cluster that Argo is running on.
 - Namespace: leave blank or set to default if required
- Values:
 - Override the values as needed. If Argo was able to successfully connect to the source repo configuration, then you should see a list of optional values to override. You can also upload your own `values.yaml` file.

To get to Argo, simply select it from the top right dropdown on the OpenShift web ui.

You can also use the following command to see what will be applied before actually applying:

```
helm template test . --debug | less
```

The default `values.yaml` is designed to be give you the basic layout, as well as provide a template for your own custom values file. Simply make a copy of this default, and start filling in your real-world data. You can even run the above template command, adding `-f /path/to/yourValues.yaml` before the `|` to tell Helm to use your new file, in order to check your work.

Components overview

nmstate operator

Relevant templates:

- `nmstate-namespace.yaml`
- `nmstate-operatorgroup.yaml`
- `nmstate-operator.yaml`
- `nmstate-instance.yaml`
- `nmstate-nncp-bond.yaml`
- `nmstate-nncp-nodes.yaml`

The `nmstate` operator is what brings in the `NodeNetworkConfigurationPolicy` CRD's , allowing us to further tweak the network setup of the cluster.

Chrony

Relevant templates:

- `chrony-configuration.yaml`

LDAP

Relevant templates:

- `ldap-accounts,secret,and oauth provider`

Storage

Relevant templates:

- `trident-sc.yaml`
- NOTE Will trigger a reboot of the cluster: `trident-machineconfig-master.yaml`
- NOTE Will trigger a reboot of the cluster: `trident-machineconfig-worker.yaml`

Most Kubernetes distributions come with the packages and utilities to mount NFS backends installed by default, including Red Hat OpenShift.

However, for NFSv3, there is no mechanism to negotiate concurrency between the client and the server. Hence the maximum number of client-side sunrpc slot table entries must be manually synced with supported value on the server to ensure the best performance for the NFS connection without the server having to decrease the window size of the connection.

For ONTAP, the supported maximum number of sunrpc slot table entries is 128 i.e. ONTAP can serve 128 concurrent NFS requests at a time. However, by default, Red Hat CoreOS/Red Hat Enterprise Linux has maximum of 65,536 sunrpc slot table entries per connection. We need to set this value to 128 and this can be done using Machine Config Operator (MCO) in OpenShift.

For more information see <https://docs.netapp.com/us-en/netapp-solutions/containers/rh-os-n-overview-trident.html#nfs>

Other

Proxy Relevant templates:

- proxy-custom-ca-cm.yaml

Ingress Relevant templates:

- ingress-config.yaml

VM Migration from VMware vCenter to OpenShift Virtualization

This guide outlines the steps to migrate virtual machines (VMs) from VMware vCenter to OpenShift Virtualization. It includes setting up networking, managing VM configurations, executing the migration, and troubleshooting common issues.

Pre-Migration Preparation

1. Access and Permissions

- Ensure you have the necessary access permissions to both VMware vCenter and OpenShift Virtualization clusters.
- Create source provider(s) for vmware and/or vcenter:
 - Steps for creating a provider can be found here: https://docs.redhat.com/en/documentation/migration_toolkit_for_virtualization/2.7/html-single/installing_and_using_the_migration_toolkit_for_virtualization/index#mtv-overview-page_mtv
- Verify the following operators are properly configured in OpenShift:
 - Virtualization Operator
 - * for hosting the migrated virtual machines
 - Migration Toolkit for Virtualization Operator
 - * to support a more automated migration of virtual machines off of VMware
 - Kubernetes NMState Operator
 - * to manage the required networking configurations for the machines

2. Identify/Prepare Target VMs

Ensure the VM(s) you wish to migrate are properly identified in VMware.

WARNING: These VMs will need to be powered down for the migration and remain down for testing to avoid IP conflicts.

3. Networking Setup

- Create Bond: Create the Bond network using a Node Network Configuration Policy (NNCP). Example:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0
spec:
  desiredState:
    interfaces:
      - name: bond0
        type: bond
        link-aggregation:
          mode: 802.3ad
        port:
          - ens2f0
          - ens2f1
          - ens3f0
          - ens3f1
```

- Create VLANs: Create VLANs to map network traffic between VMware and OpenShift Virtualization.
 - Example VLAN IDs: 677 (primary), 58 (for public networks), 59 (for public networks). Example NodeNetworkConfigurationPolicy

- Network Attachment: Apply network attachment definitions (NAD) that point to the appropriate VLAN. For instance, the primary network for the migrated VMs will map to VLAN 677, while public-facing VMs might use VLAN 58.

NOTE: Unless communication is needed with the containers then it is recommended to set the container network as disabled in the network attachment definition.

#Example Network Attachment Definitions

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond0-bridge-vlan677
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "bond0-bridge-vlan677",
    "type": "bridge",
    "bridge": "bond0-br0",
    "macspoofchk": false,
    "vlan": 677
  }'
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond0-bridge-vlan58
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "bond0-bridge-vlan58",
    "type": "bridge",
    "bridge": "bond0-br0",
    "macspoofchk": false,
    "vlan": 58
  }'
```

Migration Process

Plan Setup

Migration plans are created in OpenShift using the Migration Toolkit. This plan selects the targeted virtual machines and specifies the source and destination environments.

- Choose Source Provider: Depending on your requirements, you can select the following:
 - ESXi Host (preferred): Directly migrate VMs from an ESXi host without going through vCenter. If you have access to the ESXi host directly this is the preferable source as it is faster and can handle more machines simultaneously. The limitation here is all of your specified VMs will need to be put onto this specific ESXi host in vCenter first or you will need to create a separate migration plan for each source ESXi host.
 - vCenter: Migrate from the full vCenter environment, but avoid pulling too many VMs simultaneously to prevent system crashes. When using this plan you will want to limit the plan to a single VM.
- Create Migration Plan:

- Select source provider (e.g., `esxi13-2` or `vcenter`).
- Enter a plan name using Kubernetes naming conventions:
 - * name must be unique among other migration plans
 - * at most 63 characters
 - * only lowercase alphanumeric characters or ‘-’
 - * start with alpha
 - * end with alphanumeric
 - * (e.g., `core-test-24oct-1449`).
- Choose target provider: OpenShift Virtualization.
- Set up network mapping:
 - * Map VMware networks to OpenShift VLANs (e.g., map `vlan-677` to `vlan-677` in OCP).
 - NOTE: if the source vlan numbers don’t line up, `vlan-677` is to be used for all internal communications and should likely be used by default. `vlan 58` or `59` are considered the ‘public’ communication network.
 - If the target vlan is not available then review the network attachment definitions are set up properly.
- Set up storage mapping:
 - * Select OpenShift’s temporary storage (e.g., `openshift-temp`) and map it to OpenShift’s CSI storage (e.g., `basic-csi`).
- Start Migration: Click Start Migration to initiate the migration of selected VMs.

VM Post-Migration Configuration

The virtio drivers that OpenShift Virtualization uses are not supported by default for the older versions of RHEL and Windows Server. Since OpenShift Virtualization defaults to virtio for the network and disk interfaces you will need to update that setting post migration. Once the migration process is complete, and the VM is created in OpenShift (e.g., `web-qa`), follow these steps:

- Modify Network Interfaces:
 - Ensure the network interfaces are set to `e1000e`.
 - Be sure it is set to attach the VM to the correct VLAN (e.g., `vlan-677` for internal network communication).
- Modify Disk Configuration:
 - Change the disk interface type from virtio to SATA
- Start the VM:
 - Once all configurations are verified, go to Actions → Start to power on the VM in OpenShift Virtualization.
 - If the VM is already running you must stop it and restart it in order for the changes to be applied.

Dealing with Windows-Specific Issues

For any Windows based VMs (e.g., Windows Server 2012), special steps are needed:

- Shutdown Windows Properly in vCenter:
 - Always shut down the Windows VM gracefully: `shutdown /s`.
 - Windows VMs can experience issues migrating due to hibernation or fast boot settings. When these are enabled it may cause the file system to be mounted as read-only which prevents the migration from taking place.
- NOTE: it appears fast boot is enabled by default for Windows Server 2012

Recommended Practices and Considerations

VMs Not Starting After Migration

Any VMs running an OS which is not in the list of supported guest operating systems, or operating systems that have difficulty with the virtio drivers may need to use these other drivers for their disk and network

interfaces:

- Disk Configuration: Ensure all disks are configured to SATA after migration.
- Network Configuration: Double-check that the VM is connected to the correct VLAN and that e1000e is used for network interfaces. Additionally, if e1000e is the interface used in the VMware environment, then using this setting may help to ensure previous network configurations remain on the migrated VM.

Migration Plan Recommended Practices

- Avoid migrating multiple VMs simultaneously from vCenter as this may overwhelm the system. Always perform migration of one VM at a time when migrating from vCenter as the source provider.
- It is also possible to throttle the number of simultaneous migrations in the MTV configurations. This will allow you to put more than one machine in a single migration plan (regardless of if it's from vCenter vs ESXi host) and not to overwhelm the source system. This can be done by modifying the `controller_max_vm_inflight` setting which is found in the forklift-controller CR created by the MTV operator. More info for additional configuration options

spec:

```
controller_max_vm_inflight: <number> #defaults to 20
```

Saving and Re-Importing VMs

- Consider creating backups of VM data and configurations before migration, especially when dealing with production VMs.
- Use tools like `virtctl` to upload images or configure persistent volumes. In case of failure, ensure you have a repeatable process for re-importing the VM images.

Migration Plan Failures

- API Gateway Errors: If the migration fails due to a bad API gateway, verify network connectivity and access permissions for the gateway. Restarting the gateway or re-initiating the plan may resolve the issue.
- High RAM Usage: Monitor node resource usage during the migration process. Nodes with high memory consumption may affect the migration speed or cause failures.

More information:

- More detailed information can be found in the migration toolkit user guide: https://docs.redhat.com/en/documentation/migration_toolkit_for_virtualization/2.7
- More detailed information about OpenShift Virtualization in general can be found here: https://docs.redhat.com/en/documentation/openshift_container_platform/4.17/html/virtualization/index

Automation Notes

Automatically update the disk and network interfaces for all virtual machines with a script. This script:

- gets the current configurations for all vms on openshift and then loops through each to
 - modify the storage and network interfaces to use SATA/e1000e to prevent any potential virtio/storage issues for unsupported guest operating systems
 - apply the updated config
 - start or restart the VM

```
#!/bin/bash
```

```
# Set namespace and project
```

```

NAMESPACE="your-namespace"

# Get all VirtualMachine configurations from OpenShift
echo "Fetching all VM configurations..."
oc get vm -n "$NAMESPACE" -o name | while read -r vm; do
    echo "Processing $vm..."

    # Export the VM's YAML config
    oc get "$vm" -n "$NAMESPACE" -o yaml > "/tmp/${vm//\//-}.yaml"

    # Check if the YAML file exists
    if [[ ! -f "/tmp/${vm//\//-}.yaml" ]]; then
        echo "Error: Could not fetch YAML for $vm. Skipping..."
        continue
    fi

    # Perform sed replacement on disk interface type (virtio -> sata)
    echo "Updating disk interface to 'sata'"
    sed -i '/disk:\/,\/interface:/s/virtio/sata/g' "/tmp/${vm//\//-}.yaml"

    # Perform sed replacement on network interface type (virtio -> e1000e)
    echo "Updating network interface to 'e1000e'"
    sed -i '/networkInterfaces:/s/virtio/e1000e/g' "/tmp/${vm//\//-}.yaml"

    # Apply the updated YAML to OpenShift
    echo "Applying updated configuration for $vm..."
    oc apply -f "/tmp/${vm//\//-}.yaml" -n "$NAMESPACE"

    # Check if the VM is running or stopped
    vm_status=$(virtctl status "$vm" -n "$NAMESPACE" | grep -oP "(?<=Status: )\w+")

    if [[ "$vm_status" == "Running" ]]; then
        echo "VM $vm is running. Stopping now..."
        virtctl stop "$vm"
    elif [[ "$vm_status" == "Stopped" ]]; then
        echo "VM $vm is already stopped."
    else
        echo "VM $vm is in an unknown state: $vm_status. Skipping restart..."
        continue
    fi

    # Start the VM
    virtctl start "$vm"

    # Clean up the temporary file
    rm -f "/tmp/${vm//\//-}.yaml"

    echo "Finished processing $vm."
done

echo "All VM configurations updated successfully!"

```