**vCenter To OpenShift PoC**

**vSphere to OpenShift Migration POC**

## **Introduction**

This POC demonstrates a **service transition** from traditional vSphere virtualization to Red Hat OpenShift Container Platform, focusing on workload migration and infrastructure modernization. We're essentially performing a **major change** to our compute platform while ensuring business continuity.

The engagement follows a structured approach with four key phases: **service assessment** (discovering and cataloging existing vSphere workloads), **infrastructure deployment** (building the target OCP environment), **service preparation** (configuring virtualization and migration tools), and **controlled release deployment** (executing phased VM migrations).

Our **configuration management** approach uses automated discovery tools to inventory source VMs, analyze application dependencies, and generate migration complexity matrices. The target OpenShift cluster provides both **service validation** through OpenShift Virtualization and a **service improvement** path toward containerization.

Key deliverables include comprehensive assessment reports, automated migration tooling, and **knowledge transfer** documentation covering the entire migration workflow. This POC establishes the foundation for **continual service improvement** by providing multiple migration strategies - lift-and-shift for immediate benefits, containerization for modern apps, and replatforming for databases.

The goal is proving we can migrate vSphere workloads to OpenShift reliably while maintaining service levels and creating a repeatable migration factory for production rollouts.

*Total migration timeline: 1-2 weeks for complete POC validation*

### **Stage 0: OpenShift Container Platform Setup** - Infrastructure Planning and Specifications

#### Cluster Architecture Overview

For this POC, we'll deploy a compact, yet functional OCP cluster that can handle VM migrations while keeping resource requirements reasonable.

#### Infrastructure Layout:

A diagram of a software system

AI-generated content may be incorrect.

##### Hardware Specifications

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Hostname** | **Role** | **CPU Cores** | **RAM (GB)** | **OS Disk (GB)** | **Data Disk (GB)** | **Network** | **Purpose** |
| bastion | Helper/Bastion | 4 | 16 | 120 | - | 1 NIC | Bootstrap, DNS, Registry |
| haproxy01 | Load Balancer | 2 | 8 | 60 | - | 1 NIC | API/Ingress LB |
| master01 | Control Plane | 4 | 16 | 120 | - | 1 NIC | etcd, API Server |
| master02 | Control Plane | 4 | 16 | 120 | - | 1 NIC | etcd, API Server |
| master03 | Control Plane | 4 | 16 | 120 | - | 1 NIC | etcd, API Server |
| worker01 | Compute | 8 | 32 | 120 | 500 | 1 NIC | Workloads + Storage |
| worker02 | Compute | 8 | 32 | 120 | 500 | 1 NIC | Workloads + Storage |
| worker03 | Compute | 8 | 32 | 120 | 500 | 1 NIC | Workloads + Storage |

**Total Resources Required:**

* **CPU**: 42 cores
* **RAM**: 168 GB
* **Storage**: 2,280 GB (OS + Data)
* **VMs**: 8 total

**Network Requirements:**

* 1 x /24 subnet (minimum 30 IPs)
* DNS resolution (internal)
* Internet access for image pulls
* NTP synchronization

#### Environment Preparation

###### Setup Bastion Host

The bastion host serves as your control center for the entire deployment.

Create bastion setup script

cat > setup\_bastion.sh <<'EOF'

#!/bin/bash

echo "Setting up OpenShift bastion host for POC deployment"

# Update system

dnf update -y

# Install essential packages

dnf install -y wget curl vim git bind-utils httpd-tools jq tar

# Install OpenShift installer and client

OCP\_VERSION="4.14.8"

wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/${OCP\_VERSION}/openshift-install-linux.tar.gz

wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/${OCP\_VERSION}/openshift-client-linux.tar.gz

tar -xzf openshift-install-linux.tar.gz -C /usr/local/bin/

tar -xzf openshift-client-linux.tar.gz -C /usr/local/bin/

chmod +x /usr/local/bin/{openshift-install,oc,kubectl}

# Install Helm

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Install podman and container tools

dnf install -y podman buildah skopeo

# Create working directories

mkdir -p /opt/ocp-install/{configs,ignition,images}

mkdir -p /var/www/html/{images,ignition}

# Install and configure nginx for serving boot images

dnf install -y nginx

systemctl enable nginx

# Configure firewall

firewall-cmd --permanent --add-service=http

firewall-cmd --permanent --add-service=https

firewall-cmd --permanent --add-service=dns

firewall-cmd --reload

# Create SSH key for cluster access

if [ ! -f ~/.ssh/id\_rsa ]; then

ssh-keygen -t rsa -b 4096 -f ~/.ssh/id\_rsa -N ""

fi

echo "Bastion host setup complete"

echo "SSH public key for cluster access:"

cat ~/.ssh/id\_rsa.pub

EOF

# chmod +x setup\_bastion.sh && ./setup\_bastion.sh

###### Configure DNS (Using DNSMasq on Bastion)

# Create DNS configuration script

cat > configure\_dns.sh <<'EOF'

#!/bin/bash

echo "Configuring DNS for OpenShift cluster"

# Install dnsmasq

dnf install -y dnsmasq

# Get network configuration

BASTION\_IP=$(hostname -I | awk '{print $1}')

NETWORK\_PREFIX=$(echo $BASTION\_IP | cut -d. -f1-3)

echo "Bastion IP: $BASTION\_IP"

echo "Network: $NETWORK\_PREFIX.0/24"

# Create dnsmasq configuration

cat > /etc/dnsmasq.conf <<DNSMASQ

# Basic configuration

domain-needed

bogus-priv

listen-address=127.0.0.1,$BASTION\_IP

expand-hosts

domain=ocp.local

local=/ocp.local/

# DNS records for OpenShift cluster

address=/api.ocp.local/$BASTION\_IP

address=/api-int.ocp.local/$BASTION\_IP

address=/.apps.ocp.local/$BASTION\_IP

# Bootstrap node

address=/bootstrap.ocp.local/${NETWORK\_PREFIX}.10

# Master nodes

address=/master01.ocp.local/${NETWORK\_PREFIX}.11

address=/master02.ocp.local/${NETWORK\_PREFIX}.12

address=/master03.ocp.local/${NETWORK\_PREFIX}.13

# Worker nodes

address=/worker01.ocp.local/${NETWORK\_PREFIX}.21

address=/worker02.ocp.local/${NETWORK\_PREFIX}.22

address=/worker03.ocp.local/${NETWORK\_PREFIX}.23

# Load balancer

address=/haproxy01.ocp.local/${NETWORK\_PREFIX}.5

# etcd SRV records

srv-host=\_etcd-server-ssl.\_tcp.ocp.local,master01.ocp.local,2380

srv-host=\_etcd-server-ssl.\_tcp.ocp.local,master02.ocp.local,2380

srv-host=\_etcd-server-ssl.\_tcp.ocp.local,master03.ocp.local,2380

DNSMASQ

# Enable and start dnsmasq

systemctl enable dnsmasq

systemctl start dnsmasq

# Update local DNS to use dnsmasq

echo "nameserver 127.0.0.1" > /etc/resolv.conf.new

cat /etc/resolv.conf >> /etc/resolv.conf.new

mv /etc/resolv.conf.new /etc/resolv.conf

# Test DNS resolution

echo "Testing DNS resolution:"

nslookup api.ocp.local

nslookup master01.ocp.local

echo "DNS configuration complete"

echo "Configure other nodes to use $BASTION\_IP as primary DNS"

EOF

# chmod +x configure\_dns.sh && ./configure\_dns.sh

###### Setup Load Balancer (HAProxy)

Create HAProxy setup script for **haproxy01** node

cat > setup\_haproxy.sh <<'EOF'

#!/bin/bash

echo "Setting up HAProxy for OpenShift API and Ingress"

# Install HAProxy

dnf install -y haproxy

# Create HAProxy configuration

cat > /etc/haproxy/haproxy.cfg <<'HAPROXY'

global

log stdout local0

chroot /var/lib/haproxy

stats socket /run/haproxy/admin.sock mode 660 level admin

stats timeout 30s

user haproxy

group haproxy

daemon

defaults

mode http

log global

option httplog

option dontlognull

option log-health-checks

option forwardfor except 127.0.0.0/8

option redispatch

retries 3

timeout http-request 10s

timeout queue 1m

timeout connect 10s

timeout client 300s

timeout server 300s

timeout http-keep-alive 10s

timeout check 10s

maxconn 20000

# Stats page

listen stats

bind \*:8404

stats enable

stats uri /stats

stats refresh 30s

# OpenShift API Server

frontend openshift\_api\_frontend

bind \*:6443

default\_backend openshift\_api\_backend

mode tcp

option tcplog

backend openshift\_api\_backend

balance source

mode tcp

server bootstrap bootstrap.ocp.local:6443 check

server master01 master01.ocp.local:6443 check

server master02 master02.ocp.local:6443 check

server master03 master03.ocp.local:6443 check

# Machine Config Server

frontend machine\_config\_server\_frontend

bind \*:22623

default\_backend machine\_config\_server\_backend

mode tcp

option tcplog

backend machine\_config\_server\_backend

balance source

mode tcp

server bootstrap bootstrap.ocp.local:22623 check

server master01 master01.ocp.local:22623 check

server master02 master02.ocp.local:22623 check

server master03 master03.ocp.local:22623 check

# OpenShift Router HTTP

frontend openshift\_router\_http\_frontend

bind \*:80

default\_backend openshift\_router\_http\_backend

mode tcp

option tcplog

backend openshift\_router\_http\_backend

balance source

mode tcp

server worker01 worker01.ocp.local:80 check

server worker02 worker02.ocp.local:80 check

server worker03 worker03.ocp.local:80 check

# OpenShift Router HTTPS

frontend openshift\_router\_https\_frontend

bind \*:443

default\_backend openshift\_router\_https\_backend

mode tcp

option tcplog

backend openshift\_router\_https\_backend

balance source

mode tcp

server worker01 worker01.ocp.local:443 check

server worker02 worker02.ocp.local:443 check

server worker03 worker03.ocp.local:443 check

HAPROXY

# Configure firewall

firewall-cmd --permanent --add-port=6443/tcp

firewall-cmd --permanent --add-port=22623/tcp

firewall-cmd --permanent --add-port=80/tcp

firewall-cmd --permanent --add-port=443/tcp

firewall-cmd --permanent --add-port=8404/tcp

firewall-cmd --reload

# Enable and start HAProxy

systemctl enable haproxy

systemctl start haproxy

# Test HAProxy status

systemctl status haproxy

echo "HAProxy setup complete"

echo "Stats available at: http://haproxy01.ocp.local:8404/stats"

EOF

Copy this script to haproxy01 node and execute

# scp setup\_haproxy.sh root@haproxy01.ocp.local:/root/

# ssh root@haproxy01.ocp.local "chmod +x setup\_haproxy.sh && ./setup\_haproxy.sh"’

#### OpenShift Installation

###### Generate Installation Configuration

Create installation configuration script

cat > create\_install\_config.sh <<'EOF'

#!/bin/bash

echo "Creating OpenShift installation configuration"

cd /opt/ocp-install/configs

# Get your Red Hat pull secret

echo "You need your Red Hat pull secret from:"

echo "https://console.redhat.com/openshift/install/pull-secret"

echo

read -p "Paste your pull secret here: " PULL\_SECRET

# Create install-config.yaml

cat > install-config.yaml <<YAML

apiVersion: v1

baseDomain: ocp.local

compute:

- hyperthreading: Enabled

name: worker

replicas: 0

controlPlane:

hyperthreading: Enabled

name: master

replicas: 3

metadata:

name: cluster

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

networkType: OVNKubernetes

serviceNetwork:

- 172.30.0.0/16

platform:

none: {}

fips: false

pullSecret: '$PULL\_SECRET'

sshKey: '$(cat ~/.ssh/id\_rsa.pub)'

YAML

# Create backup of install config

cp install-config.yaml install-config.yaml.backup

echo "Installation configuration created"

echo "Review and modify install-config.yaml if needed"

cat install-config.yaml

EOF

# chmod +x create\_install\_config.sh && ./create\_install\_config.sh

###### Generate Ignition Configs

Create ignition generation script

cat > generate\_ignition.sh <<'EOF'

#!/bin/bash

echo "Generating OpenShift ignition configurations"

cd /opt/ocp-install/configs

# Generate ignition configs

openshift-install create ignition-configs --dir=.

# Copy ignition files to web server

cp \*.ign /var/www/html/ignition/

chmod 644 /var/www/html/ignition/\*.ign

# Start nginx to serve ignition files

systemctl start nginx

systemctl enable nginx

echo "Ignition files generated and available at:"

echo "http://$(hostname -I | awk '{print $1}')/ignition/"

ls -la /var/www/html/ignition/

# Display ignition URLs for reference

BASTION\_IP=$(hostname -I | awk '{print $1}')

echo ""

echo "Ignition file URLs for boot parameters:"

echo "Bootstrap: http://$BASTION\_IP/ignition/bootstrap.ign"

echo "Master: http://$BASTION\_IP/ignition/master.ign"

echo "Worker: http://$BASTION\_IP/ignition/worker.ign"

EOF

# chmod +x generate\_ignition.sh && ./generate\_ignition.sh

###### Download and Prepare Boot Images

Create RHCOS image download script

cat > download\_rhcos.sh <<'EOF'

#!/bin/bash

echo "Downloading RHCOS images for OpenShift installation"

cd /opt/ocp-install/images

# Get OpenShift version from installer

OCP\_VERSION=$(openshift-install version | grep "openshift-install" | awk '{print $2}')

echo "OpenShift version: $OCP\_VERSION"

# Download RHCOS images

RHCOS\_VERSION="4.14.3" # Adjust based on your OCP version

# Download ISO for manual installations

wget -O rhcos-live.x86\_64.iso \

"https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/${RHCOS\_VERSION}/rhcos-${RHCOS\_VERSION}-x86\_64-live.x86\_64.iso"

# Download kernel and initramfs for PXE (if needed)

wget -O rhcos-live-kernel-x86\_64 \

"https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/${RHCOS\_VERSION}/rhcos-${RHCOS\_VERSION}-x86\_64-live-kernel-x86\_64"

wget -O rhcos-live-initramfs.x86\_64.img \

"https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/${RHCOS\_VERSION}/rhcos-${RHCOS\_VERSION}-x86\_64-live-initramfs.x86\_64.img"

# Copy to web server for access

cp rhcos-live.x86\_64.iso /var/www/html/images/

cp rhcos-live-\* /var/www/html/images/

echo "RHCOS images downloaded and available at:"

echo "http://$(hostname -I | awk '{print $1}')/images/"

ls -la /var/www/html/images/

EOF

# chmod +x download\_rhcos.sh && ./download\_rhcos.sh

##### Node Installation

###### Bootstrap Node Installation

Create bootstrap installation guide

cat > install\_bootstrap.sh <<'EOF'

#!/bin/bash

echo "Bootstrap Node Installation Instructions"

echo "========================================"

BASTION\_IP=$(hostname -I | awk '{print $1}')

echo ""

echo "1. Boot bootstrap.ocp.local from RHCOS ISO"

echo "2. At the boot prompt, append these kernel parameters:"

echo ""

echo "coreos.inst.install\_dev=/dev/sda \\"

echo "coreos.inst.ignition\_url=http://$BASTION\_IP/ignition/bootstrap.ign \\"

echo "ip=dhcp"

echo ""

echo "Full boot command example:"

echo "vmlinuz ... coreos.inst.install\_dev=/dev/sda coreos.inst.ignition\_url=http://$BASTION\_IP/ignition/bootstrap.ign ip=dhcp"

echo ""

echo "3. Wait for installation to complete and node to reboot"

echo "4. Bootstrap node will be available at: bootstrap.ocp.local"

echo ""

echo "Monitor bootstrap process with:"

echo "openshift-install wait-for bootstrap-complete --dir=/opt/ocp-install/configs --log-level=debug"

EOF

# chmod +x install\_bootstrap.sh && ./install\_bootstrap.sh

###### Master Nodes Installation Script

Create master nodes installation script

cat > install\_masters.sh <<'EOF'

#!/bin/bash

echo "Master Nodes Installation Instructions"

echo "======================================"

BASTION\_IP=$(hostname -I | awk '{print $1}')

echo ""

echo "Install each master node (master01, master02, master03) with:"

echo ""

echo "1. Boot from RHCOS ISO"

echo "2. Use these kernel parameters:"

echo ""

echo "coreos.inst.install\_dev=/dev/sda \\"

echo "coreos.inst.ignition\_url=http://$BASTION\_IP/ignition/master.ign \\"

echo "ip=dhcp"

echo ""

echo "3. Installation order:"

echo " - Start all three masters simultaneously"

echo " - They will form etcd cluster automatically"

echo ""

echo "4. Wait for all masters to complete installation"

echo ""

echo "Monitor progress:"

echo "openshift-install wait-for bootstrap-complete --dir=/opt/ocp-install/configs"

echo ""

echo "Check master nodes status:"

echo "oc --kubeconfig=/opt/ocp-install/configs/auth/kubeconfig get nodes"

EOF

# chmod +x install\_masters.sh && ./install\_masters.sh

###### Worker Nodes Installation Script

Create worker nodes installation script

cat > install\_workers.sh <<'EOF'

#!/bin/bash

echo "Worker Nodes Installation Instructions"

echo "======================================"

BASTION\_IP=$(hostname -I | awk '{print $1}')

echo ""

echo "Install each worker node (worker01, worker02, worker03) with:"

echo ""

echo "1. Boot from RHCOS ISO"

echo "2. Use these kernel parameters:"

echo ""

echo "coreos.inst.install\_dev=/dev/sda \\"

echo "coreos.inst.ignition\_url=http://$BASTION\_IP/ignition/worker.ign \\"

echo "ip=dhcp"

echo ""

echo "3. After installation, approve pending CSRs:"

echo ""

echo "# Check pending CSRs"

echo "oc --kubeconfig=/opt/ocp-install/configs/auth/kubeconfig get csr"

echo ""

echo "# Approve all pending CSRs"

echo "oc --kubeconfig=/opt/ocp-install/configs/auth/kubeconfig get csr -o name | xargs oc --kubeconfig=/opt/ocp-install/configs/auth/kubeconfig adm certificate approve"

echo ""

echo "4. Wait for worker nodes to join cluster:"

echo "oc --kubeconfig=/opt/ocp-install/configs/auth/kubeconfig get nodes"

EOF

# chmod +x install\_workers.sh && ./install\_workers.sh

###### Complete Installation Monitoring Script

Create installation monitoring and completion script

cat > complete\_installation.sh <<'EOF'

#!/bin/bash

echo "OpenShift Installation Monitoring and Completion"

echo "==============================================="

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

cd /opt/ocp-install/configs

# Wait for bootstrap to complete

echo "Step 1: Waiting for bootstrap completion..."

openshift-install wait-for bootstrap-complete --log-level=debug

if [ $? -eq 0 ]; then

echo "Bootstrap completed successfully!"

echo "You can now shut down the bootstrap node"

# Remove bootstrap from haproxy

echo "Removing bootstrap from HAProxy configuration..."

ssh root@haproxy01.ocp.local "sed -i '/server bootstrap/d' /etc/haproxy/haproxy.cfg && systemctl reload haproxy"

else

echo "Bootstrap failed. Check logs for issues."

exit 1

fi

# Check cluster nodes

echo "Step 2: Checking cluster nodes..."

oc get nodes

# Approve any pending CSRs

echo "Step 3: Approving pending certificate signing requests..."

while true; do

pending\_csrs=$(oc get csr --no-headers | grep Pending | wc -l)

if [ $pending\_csrs -gt 0 ]; then

echo "Found $pending\_csrs pending CSRs, approving..."

oc get csr -o name | xargs oc adm certificate approve

sleep 10

else

break

fi

done

# Wait for installation to complete

echo "Step 4: Waiting for installation to complete..."

openshift-install wait-for install-complete --log-level=debug

if [ $? -eq 0 ]; then

echo ""

echo "OpenShift installation completed successfully!"

echo ""

echo "Cluster access information:"

echo "=========================="

echo "Console URL: https://console-openshift-console.apps.ocp.local"

echo "API URL: https://api.ocp.local:6443"

echo ""

echo "Admin credentials:"

cat auth/kubeconfig | grep server

echo ""

echo "kubeadmin password:"

cat auth/kubeadmin-password

echo ""

echo "To access cluster:"

echo "export KUBECONFIG=/opt/ocp-install/configs/auth/kubeconfig"

echo "oc whoami"

echo ""

echo "Cluster status:"

oc get co | grep -v AVAILABLE.\*True || echo "Some operators still progressing..."

else

echo "Installation failed. Check logs for issues."

exit 1

fi

EOF

# chmod +x complete\_installation.sh

##### Post-Installation Configuration

###### Configure Storage for VM Migration

Create storage configuration script for VM workloads

cat > configure\_storage.sh <<'EOF'

#!/bin/bash

echo "Configuring storage for OpenShift virtualization workloads"

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

# Label worker nodes for storage

echo "Labeling worker nodes for local storage..."

oc label node worker01.ocp.local node-role.kubernetes.io/storage=""

oc label node worker02.ocp.local node-role.kubernetes.io/storage=""

oc label node worker03.ocp.local node-role.kubernetes.io/storage=""

# Install Local Storage Operator

echo "Installing Local Storage Operator..."

cat <<YAML | oc apply -f -

apiVersion: v1

kind: Namespace

metadata:

name: openshift-local-storage

---

apiVersion: operators.coreos.com/v1alpha2

kind: OperatorGroup

metadata:

name: local-operator-group

namespace: openshift-local-storage

spec:

targetNamespaces:

- openshift-local-storage

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: local-storage-operator

namespace: openshift-local-storage

spec:

channel: stable

name: local-storage-operator

source: redhat-operators

sourceNamespace: openshift-marketplace

YAML

# Wait for operator to be ready

echo "Waiting for Local Storage Operator..."

sleep 30

oc wait --for=condition=Ready csv -l operators.coreos.com/local-storage-operator.openshift-local-storage -n openshift-local-storage --timeout=300s

# Create LocalVolume for VM storage

echo "Creating local volume configuration..."

cat <<YAML | oc apply -f -

apiVersion: local.storage.openshift.io/v1

kind: LocalVolume

metadata:

name: local-vm-storage

namespace: openshift-local-storage

spec:

nodeSelector:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- worker01.ocp.local

- worker02.ocp.local

- worker03.ocp.local

storageClassDevices:

- storageClassName: local-vm-storage

volumeMode: Filesystem

fsType: xfs

devicePaths:

- /dev/sdb

YAML

echo "Storage configuration applied"

echo "Checking for local storage PVs..."

sleep 60

oc get pv | grep local-vm-storage

echo "Storage configuration complete"

EOF

# chmod +x configure\_storage.sh && ./configure\_storage.sh

###### Install OpenShift Virtualization

Create OpenShift Virtualization installation script

cat > install\_virtualization.sh <<'EOF'

#!/bin/bash

echo "Installing OpenShift Virtualization for VM migration"

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

# Create namespace and install CNV operator

echo "Installing OpenShift Virtualization operator..."

cat <<YAML | oc apply -f -

apiVersion: v1

kind: Namespace

metadata:

name: openshift-cnv

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: kubevirt-hyperconverged-group

namespace: openshift-cnv

spec:

targetNamespaces:

- openshift-cnv

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: hco-operatorhub

namespace: openshift-cnv

spec:

source: redhat-operators

sourceNamespace: openshift-marketplace

name: kubevirt-hyperconverged

channel: stable

YAML

# Wait for operator installation

echo "Waiting for CNV operator to be ready..."

sleep 60

oc wait --for=condition=Ready csv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv -n openshift-cnv --timeout=600s

# Create HyperConverged instance

echo "Creating HyperConverged instance..."

cat <<YAML | oc apply -f -

apiVersion: hco.kubevirt.io/v1beta1

kind: HyperConverged

metadata:

name: kubevirt-hyperconverged

namespace: openshift-cnv

spec:

storageImport:

insecureRegistries: []

YAML

# Wait for virtualization to be ready

echo "Waiting for OpenShift Virtualization to be ready..."

sleep 120

oc wait --for=condition=Available hyperconverged kubevirt-hyperconverged -n openshift-cnv --timeout=900s

echo "Checking virtualization status..."

oc get hyperconverged -n openshift-cnv

oc get pods -n openshift-cnv

echo "OpenShift Virtualization installation complete"

echo "You can now migrate VMs to this cluster"

EOF

# chmod +x install\_virtualization.sh && ./install\_virtualization.sh

##### Complete Installation Workflow

Create master installation workflow script

cat > deploy\_ocp\_cluster.sh <<'EOF'

#!/bin/bash

echo "OpenShift Container Platform Deployment for vSphere Migration POC"

echo "================================================================="

# Check if running on bastion

if ! command -v openshift-install &> /dev/null; then

echo "Run this script on the bastion host after setting it up"

exit 1

fi

echo "Pre-installation checklist:"

echo "1. All nodes powered on and accessible"

echo "2. DNS configured and tested"

echo "3. HAProxy running and configured"

echo "4. Pull secret available"

echo ""

read -p "Are all prerequisites met? (y/n): " PREREQS\_MET

if [[ "$PREREQS\_MET" != "y" ]]; then

echo "Complete prerequisites first:"

echo "- Run setup\_bastion.sh on this host"

echo "- Run configure\_dns.sh on this host"

echo "- Run setup\_haproxy.sh on haproxy01"

exit 1

fi

# Phase 1: Generate configurations

echo ""

echo "Phase 1: Generating installation configurations"

./create\_install\_config.sh

./generate\_ignition.sh

./download\_rhcos.sh

# Phase 2: Manual node installation

echo ""

echo "Phase 2: Manual node installation required"

echo "Install nodes in this order:"

echo "1. Bootstrap node first"

echo "2. All master nodes simultaneously"

echo "3. All worker nodes"

echo ""

echo "Installation guides:"

./install\_bootstrap.sh

echo ""

read -p "Press Enter after bootstrap node is installed and running..."

./install\_masters.sh

echo ""

read -p "Press Enter after all master nodes are installed and running..."

./install\_workers.sh

echo ""

read -p "Press Enter after all worker nodes are installed and running..."

# Phase 3: Complete installation

echo ""

echo "Phase 3: Completing OpenShift installation"

./complete\_installation.sh

# Phase 4: Configure for VM migration

echo ""

echo "Phase 4: Configuring cluster for VM migration"

./configure\_storage.sh

./install\_virtualization.sh

echo ""

echo "OpenShift deployment complete!"

echo "Next steps:"

echo "1. Access console: https://console-openshift-console.apps.ocp.local"

echo "2. Configure vSphere migration with previous migration scripts"

echo "3. Start POC VM migrations"

# Generate summary

cat > /opt/ocp-install/cluster\_summary.txt <<SUMMARY

OpenShift Cluster Summary

========================

Cluster Name: cluster.ocp.local

Console: https://console-openshift-console.apps.ocp.local

API: https://api.ocp.local:6443

Admin Access:

kubeconfig: /opt/ocp-install/configs/auth/kubeconfig

password: $(cat /opt/ocp-install/configs/auth/kubeadmin-password)

Node Status:

$(export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig" && oc get nodes)

Cluster Operators:

$(export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig" && oc get co)

Installation completed: $(date)

SUMMARY

echo "Cluster summary saved to: /opt/ocp-install/cluster\_summary.txt"

EOF

# chmod +x deploy\_ocp\_cluster.sh

The cluster will be ready for vSphere VM migration testing once all components are installed and healthy. The setup provides a solid foundation for evaluating OpenShift Virtualization capabilities while maintaining reasonable resource requirements for a POC environment.

### **Stage 1: Assessment and Discovery**

#### **Setup Assessment Environment**

First, spin up a RHEL 9 VM in your vSphere environment to run the assessment tools. This becomes your migration control plane.

# Create assessment VM setup script

cat > setup\_assessment\_node.sh <<'EOF'

#!/bin/bash

echo "Setting up vSphere to OCP migration assessment node"

# Update system

dnf update -y

# Install required packages

dnf install -y git curl wget jq python3-pip podman skopeo buildah

# Install govc for vSphere API interactions

curl -L -o - "https://github.com/vmware/govmomi/releases/latest/download/govc\_$(uname -s)\_$(uname -m).tar.gz" | tar -C /usr/local/bin -xvzf - govc

chmod +x /usr/local/bin/govc

# Install oc client

curl -L https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable/openshift-client-linux.tar.gz | tar -C /usr/local/bin -xzf - oc kubectl

# Install helm

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Create working directories

mkdir -p /opt/migration/{scripts,data,reports}

cd /opt/migration

echo "Assessment node ready"

EOF

# chmod +x setup\_assessment\_node.sh && ./setup\_assessment\_node.sh

#### **vSphere Discovery and Inventory**

Create vSphere discovery script

cat > /opt/migration/scripts/vsphere\_discovery.sh <<'EOF'

#!/bin/bash

# vSphere connection details

export GOVC\_URL='https://vsphere.example.com/sdk'

export GOVC\_USERNAME='your-username'

export GOVC\_PASSWORD='your-password'

export GOVC\_INSECURE=1

echo "Starting vSphere infrastructure discovery"

# Test connectivity

if ! govc about &>/dev/null; then

echo "Failed to connect to vSphere. Check credentials and connectivity."

exit 1

fi

echo "Connected to vSphere successfully"

# Create output directory

OUTPUT\_DIR="/opt/migration/data/$(date +%Y%m%d\_%H%M%S)"

mkdir -p $OUTPUT\_DIR

# Discover all VMs with detailed information

echo "Discovering virtual machines..."

govc find . -type m | while read vm\_path; do

vm\_name=$(basename "$vm\_path")

echo "Processing VM: $vm\_name"

govc vm.info -json "$vm\_path" | jq -r '

.VirtualMachines[0] | {

name: .Name,

path: .Config.Name,

os: .Config.GuestFullName,

cpu\_count: .Config.Hardware.NumCPU,

memory\_mb: .Config.Hardware.MemoryMB,

power\_state: .Runtime.PowerState,

tools\_status: .Guest.ToolsStatus,

ip\_address: .Guest.IpAddress,

hostname: .Guest.HostName,

disks: [.Config.Hardware.Device[] | select(.DeviceInfo.Label | startswith("Hard disk")) | {

label: .DeviceInfo.Label,

size\_gb: (.CapacityInKB / 1024 / 1024 | floor)

}],

networks: [.Config.Hardware.Device[] | select(.DeviceInfo.Summary | contains("Network")) | .DeviceInfo.Summary]

}' > "$OUTPUT\_DIR/${vm\_name}\_details.json"

done

# Generate consolidated inventory

echo "Generating consolidated inventory..."

cat > "$OUTPUT\_DIR/generate\_inventory.py" <<'PYTHON'

import json

import glob

import csv

import os

output\_dir = os.environ.get('OUTPUT\_DIR')

json\_files = glob.glob(f"{output\_dir}/\*\_details.json")

inventory = []

for file\_path in json\_files:

with open(file\_path, 'r') as f:

vm\_data = json.load(f)

total\_disk\_gb = sum([disk['size\_gb'] for disk in vm\_data.get('disks', [])])

inventory.append({

'vm\_name': vm\_data.get('name', 'Unknown'),

'os': vm\_data.get('os', 'Unknown'),

'cpu\_count': vm\_data.get('cpu\_count', 0),

'memory\_gb': round(vm\_data.get('memory\_mb', 0) / 1024, 2),

'total\_disk\_gb': total\_disk\_gb,

'power\_state': vm\_data.get('power\_state', 'Unknown'),

'ip\_address': vm\_data.get('ip\_address', 'Unknown'),

'hostname': vm\_data.get('hostname', 'Unknown'),

'tools\_status': vm\_data.get('tools\_status', 'Unknown')

})

# Write CSV inventory

csv\_file = f"{output\_dir}/vm\_inventory.csv"

with open(csv\_file, 'w', newline='') as csvfile:

fieldnames = ['vm\_name', 'os', 'cpu\_count', 'memory\_gb', 'total\_disk\_gb', 'power\_state', 'ip\_address', 'hostname', 'tools\_status']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(inventory)

print(f"Inventory written to: {csv\_file}")

print(f"Total VMs discovered: {len(inventory)}")

# Generate resource summary

total\_cpu = sum([vm['cpu\_count'] for vm in inventory])

total\_memory = sum([vm['memory\_gb'] for vm in inventory])

total\_storage = sum([vm['total\_disk\_gb'] for vm in inventory])

with open(f"{output\_dir}/resource\_summary.txt", 'w') as f:

f.write(f"vSphere Infrastructure Summary\n")

f.write(f"============================\n")

f.write(f"Total VMs: {len(inventory)}\n")

f.write(f"Total CPU cores: {total\_cpu}\n")

f.write(f"Total Memory (GB): {total\_memory}\n")

f.write(f"Total Storage (GB): {total\_storage}\n\n")

f.write(f"Recommended OCP Requirements (with 50% buffer):\n")

f.write(f"CPU cores: {int(total\_cpu \* 1.5)}\n")

f.write(f"Memory (GB): {int(total\_memory \* 1.5)}\n")

f.write(f"Storage (GB): {int(total\_storage \* 1.5)}\n")

PYTHON

# Run Python inventory script

export OUTPUT\_DIR="$OUTPUT\_DIR"

python3 "$OUTPUT\_DIR/generate\_inventory.py"

# Network and storage discovery

echo "Discovering networks..."

govc ls network/ > "$OUTPUT\_DIR/networks.txt"

echo "Discovering datastores..."

govc datastore.info -json | jq -r '.Datastores[] | "\(.Name) \(.Summary.Capacity/1024/1024/1024|floor)GB \(.Summary.Type)"' > "$OUTPUT\_DIR/datastores.txt"

echo "Discovery complete. Results in: $OUTPUT\_DIR"

echo "Review vm\_inventory.csv and resource\_summary.txt for planning"

EOF

# chmod +x /opt/migration/scripts/vsphere\_discovery.sh

#### **Application Assessment and Complexity Analysis**

Create application discovery script

cat > /opt/migration/scripts/app\_assessment.sh <<'EOF'

#!/bin/bash

INVENTORY\_CSV="/opt/migration/data/$(ls -t /opt/migration/data/ | head -1)/vm\_inventory.csv"

if [ ! -f "$INVENTORY\_CSV" ]; then

echo "Run vsphere\_discovery.sh first to generate VM inventory"

exit 1

fi

echo "Starting application assessment"

OUTPUT\_DIR=$(dirname "$INVENTORY\_CSV")

ASSESSMENT\_DIR="$OUTPUT\_DIR/application\_assessment"

mkdir -p "$ASSESSMENT\_DIR"

# Create application discovery script for remote execution

cat > "$ASSESSMENT\_DIR/remote\_app\_discovery.sh" <<'REMOTE\_SCRIPT'

#!/bin/bash

HOSTNAME=$(hostname)

echo "=== Application Discovery for $HOSTNAME ==="

# Operating system details

echo "OS\_INFO:"

cat /etc/os-release 2>/dev/null || cat /etc/redhat-release 2>/dev/null

# Running services

echo -e "\nSERVICES:"

systemctl list-units --type=service --state=running --no-pager | grep -v '@' | head -20

# Listening ports and processes

echo -e "\nLISTENING\_PORTS:"

netstat -tlnp 2>/dev/null | grep LISTEN | head -20

# Database detection

echo -e "\nDATABASES:"

ps aux | grep -E "(mysqld|postgres|oracle|mongo)" | grep -v grep

systemctl status mysqld mariadb postgresql\* oracle\* mongod 2>/dev/null | grep -E "(Active|Loaded)"

# Application servers

echo -e "\nAPP\_SERVERS:"

ps aux | grep -E "(tomcat|jboss|weblogic|websphere|wildfly)" | grep -v grep

find /opt /usr/local /home -name "\*.war" -o -name "catalina.sh" -o -name "standalone.sh" 2>/dev/null | head -10

# Web servers

echo -e "\nWEB\_SERVERS:"

systemctl status httpd nginx apache2 2>/dev/null | grep -E "(Active|Loaded)"

ps aux | grep -E "(httpd|nginx|apache)" | grep -v grep

# Java applications

echo -e "\nJAVA\_APPS:"

ps aux | grep java | grep -v grep | head -10

find / -name "\*.jar" -type f 2>/dev/null | head -20

# Container runtime

echo -e "\nCONTAINERS:"

systemctl status docker podman containerd 2>/dev/null | grep -E "(Active|Loaded)"

docker ps 2>/dev/null || podman ps 2>/dev/null

# Cron jobs

echo -e "\nCRON\_JOBS:"

crontab -l 2>/dev/null

ls -la /etc/cron.\* 2>/dev/null

# Mounted filesystems

echo -e "\nFILESYSTEMS:"

df -h

cat /etc/fstab | grep -v "^#"

REMOTE\_SCRIPT

chmod +x "$ASSESSMENT\_DIR/remote\_app\_discovery.sh"

# Process each VM from inventory

tail -n +2 "$INVENTORY\_CSV" | while IFS=',' read vm\_name os cpu memory disk power ip hostname tools\_status; do

if [[ "$power" == "poweredOn" && "$ip" != "Unknown" && "$ip" != "" ]]; then

echo "Assessing applications on: $vm\_name ($ip)"

# Try SSH connection with common methods

if ssh -o ConnectTimeout=10 -o StrictHostKeyChecking=no root@$ip "echo 'SSH successful'" &>/dev/null; then

scp -o ConnectTimeout=10 -o StrictHostKeyChecking=no "$ASSESSMENT\_DIR/remote\_app\_discovery.sh" root@$ip:/tmp/

ssh -o ConnectTimeout=10 -o StrictHostKeyChecking=no root@$ip "chmod +x /tmp/remote\_app\_discovery.sh && /tmp/remote\_app\_discovery.sh" > "$ASSESSMENT\_DIR/${vm\_name}\_apps.txt" 2>&1

echo "Completed assessment for $vm\_name"

else

echo "Cannot SSH to $vm\_name ($ip) - skipping application assessment"

echo "SSH\_FAILED: Cannot connect to $ip" > "$ASSESSMENT\_DIR/${vm\_name}\_apps.txt"

fi

else

echo "Skipping $vm\_name - VM not powered on or no IP"

fi

done

# Generate migration complexity matrix

cat > "$ASSESSMENT\_DIR/generate\_complexity.py" <<'PYTHON'

import csv

import glob

import os

assessment\_dir = os.environ.get('ASSESSMENT\_DIR')

inventory\_csv = os.environ.get('INVENTORY\_CSV')

# Read VM inventory

vms = {}

with open(inventory\_csv, 'r') as f:

reader = csv.DictReader(f)

for row in reader:

vms[row['vm\_name']] = row

# Analyze application files

app\_files = glob.glob(f"{assessment\_dir}/\*\_apps.txt")

migration\_plan = []

for app\_file in app\_files:

vm\_name = os.path.basename(app\_file).replace('\_apps.txt', '')

if vm\_name not in vms:

continue

vm\_info = vms[vm\_name]

with open(app\_file, 'r') as f:

content = f.read().lower()

# Determine complexity and migration strategy

complexity = "LOW"

migration\_type = "LIFT\_AND\_SHIFT"

notes = []

# Database detection increases complexity

if any(db in content for db in ['mysqld', 'postgres', 'oracle', 'mongo']):

complexity = "HIGH"

migration\_type = "REPLATFORM"

notes.append("Database detected")

# Application servers suggest containerization opportunity

elif any(app in content for app in ['tomcat', 'jboss', 'wildfly', 'java']):

complexity = "MEDIUM"

migration\_type = "CONTAINERIZE"

notes.append("Java application server")

# Container runtime suggests easy containerization

elif any(container in content for container in ['docker', 'podman']):

complexity = "LOW"

migration\_type = "CONTAINERIZE"

notes.append("Already containerized")

# Web servers can often be containerized

elif any(web in content for web in ['httpd', 'nginx', 'apache']):

complexity = "LOW"

migration\_type = "CONTAINERIZE"

notes.append("Web server")

# SSH failures need manual assessment

if 'ssh\_failed' in content:

complexity = "UNKNOWN"

migration\_type = "MANUAL\_ASSESSMENT"

notes.append("SSH access failed")

migration\_plan.append({

'vm\_name': vm\_name,

'os': vm\_info['os'],

'cpu': vm\_info['cpu\_count'],

'memory\_gb': vm\_info['memory\_gb'],

'disk\_gb': vm\_info['total\_disk\_gb'],

'complexity': complexity,

'migration\_type': migration\_type,

'priority': 'MEDIUM',

'notes': '; '.join(notes)

})

# Write migration plan

plan\_file = f"{assessment\_dir}/migration\_plan.csv"

with open(plan\_file, 'w', newline='') as csvfile:

fieldnames = ['vm\_name', 'os', 'cpu', 'memory\_gb', 'disk\_gb', 'complexity', 'migration\_type', 'priority', 'notes']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(migration\_plan)

print(f"Migration plan written to: {plan\_file}")

# Summary stats

complexity\_counts = {}

migration\_type\_counts = {}

for vm in migration\_plan:

complexity\_counts[vm['complexity']] = complexity\_counts.get(vm['complexity'], 0) + 1

migration\_type\_counts[vm['migration\_type']] = migration\_type\_counts.get(vm['migration\_type'], 0) + 1

with open(f"{assessment\_dir}/assessment\_summary.txt", 'w') as f:

f.write("Application Assessment Summary\n")

f.write("=============================\n\n")

f.write("Complexity Breakdown:\n")

for complexity, count in complexity\_counts.items():

f.write(f" {complexity}: {count} VMs\n")

f.write("\nMigration Strategy Breakdown:\n")

for migration\_type, count in migration\_type\_counts.items():

f.write(f" {migration\_type}: {count} VMs\n")

print("Assessment summary written to assessment\_summary.txt")

PYTHON

export ASSESSMENT\_DIR="$ASSESSMENT\_DIR"

export INVENTORY\_CSV="$INVENTORY\_CSV"

python3 "$ASSESSMENT\_DIR/generate\_complexity.py"

echo "Application assessment complete"

echo "Review migration\_plan.csv for detailed migration strategy"

EOF

# chmod +x /opt/migration/scripts/app\_assessment.sh

#### **Run Complete Assessment**

Execute the complete assessment workflow

# cd /opt/migration/scripts

echo "Starting complete vSphere to OCP migration assessment"

echo "Stage 1: vSphere infrastructure discovery"

./vsphere\_discovery.sh

echo "Stage 2: Application assessment"

./app\_assessment.sh

echo "Assessment complete. Check /opt/migration/data/ for results"

ls -la /opt/migration/data/$(ls -t /opt/migration/data/ | head -1)/

### **Stage 2: OpenShift Container Platform Prep**

#### **OCP Cluster Preparation**

Assuming you already have an OCP cluster, let's prepare it for VM migration.

# Create OCP migration setup script

cat > /opt/migration/scripts/setup\_ocp\_migration.sh <<'EOF'

#!/bin/bash

echo "Setting up OpenShift for VM migration"

# Login to OCP cluster

read -p "Enter OCP cluster API URL: " OCP\_API\_URL

read -p "Enter OCP admin username: " OCP\_USER

read -s -p "Enter OCP admin password: " OCP\_PASS

echo

oc login $OCP\_API\_URL -u $OCP\_USER -p $OCP\_PASS

# Verify cluster access

if ! oc whoami &>/dev/null; then

echo "Failed to login to OCP cluster"

exit 1

fi

echo "Connected to OCP cluster: $(oc cluster-info | head -1)"

# Create migration namespace

oc new-project openshift-mtv || oc project openshift-mtv

# Install OpenShift Virtualization operator

echo "Installing OpenShift Virtualization operator"

cat <<YAML | oc apply -f -

apiVersion: v1

kind: Namespace

metadata:

name: openshift-cnv

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: kubevirt-hyperconverged-group

namespace: openshift-cnv

spec:

targetNamespaces:

- openshift-cnv

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: hco-operatorhub

namespace: openshift-cnv

spec:

source: redhat-operators

sourceNamespace: openshift-marketplace

name: kubevirt-hyperconverged

startingCSV: kubevirt-hyperconverged-operator.v4.14.0

channel: "stable"

YAML

# Wait for operator installation

echo "Waiting for OpenShift Virtualization operator to install"

sleep 30

oc wait --for=condition=Ready csv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv -n openshift-cnv --timeout=600s

# Create HyperConverged instance to enable virtualization

echo "Enabling OpenShift Virtualization"

cat <<YAML | oc apply -f -

apiVersion: hco.kubevirt.io/v1beta1

kind: HyperConverged

metadata:

name: kubevirt-hyperconverged

namespace: openshift-cnv

spec: {}

YAML

# Install Migration Toolkit for Virtualization

echo "Installing Migration Toolkit for Virtualization"

cat <<YAML | oc apply -f -

apiVersion: v1

kind: Namespace

metadata:

name: openshift-mtv

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: migration-operator

namespace: openshift-mtv

spec:

targetNamespaces:

- openshift-mtv

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: mtv-operator

namespace: openshift-mtv

spec:

channel: release-v2.5

name: mtv-operator

source: redhat-operators

sourceNamespace: openshift-marketplace

YAML

# Wait for MTV operator

echo "Waiting for MTV operator to install"

sleep 30

oc wait --for=condition=Ready csv -l operators.coreos.com/mtv-operator.openshift-mtv -n openshift-mtv --timeout=600s

# Create ForkliftController instance

echo "Creating MTV ForkliftController"

cat <<YAML | oc apply -f -

apiVersion: forklift.konveyor.io/v1beta1

kind: ForkliftController

metadata:

name: forklift-controller

namespace: openshift-mtv

spec:

olm\_managed: true

YAML

# Verify installations

echo "Verifying installations..."

sleep 60

echo "OpenShift Virtualization status:"

oc get csv -n openshift-cnv

oc get hyperconverged -n openshift-cnv

echo "MTV status:"

oc get csv -n openshift-mtv

oc get forkliftcontroller -n openshift-mtv

# Get MTV UI URL

MTV\_ROUTE=$(oc get route forklift-ui -n openshift-mtv -o jsonpath='{.spec.host}' 2>/dev/null)

if [ ! -z "$MTV\_ROUTE" ]; then

echo "MTV UI available at: https://$MTV\_ROUTE"

else

echo "MTV UI route not ready yet. Check again in a few minutes."

fi

echo "OCP migration setup complete"

EOF

# chmod +x /opt/migration/scripts/setup\_ocp\_migration.sh

# ./setup\_ocp\_migration.sh

#### **Configure Storage and Network for Migration**

Create storage and network configuration script

cat > /opt/migration/scripts/configure\_migration\_infrastructure.sh <<'EOF'

#!/bin/bash

echo "Configuring OCP infrastructure for VM migration"

# Check available storage classes

echo "Available storage classes:"

oc get storageclass

# Create storage mapping configuration

read -p "Enter primary storage class name for VM disks: " STORAGE\_CLASS

cat > /opt/migration/storage-map.yaml <<YAML

apiVersion: forklift.konveyor.io/v1beta1

kind: StorageMap

metadata:

name: vsphere-to-ocp-storage

namespace: openshift-mtv

spec:

map:

- source:

name: datastore1

destination:

storageClass: $STORAGE\_CLASS

- source:

name: datastore2

destination:

storageClass: $STORAGE\_CLASS

YAML

oc apply -f /opt/migration/storage-map.yaml

# Create network mapping configuration

echo "Available networks:"

oc get networks.config.openshift.io cluster -o yaml | grep -A 10 "clusterNetwork"

cat > /opt/migration/network-map.yaml <<YAML

apiVersion: forklift.konveyor.io/v1beta1

kind: NetworkMap

metadata:

name: vsphere-to-ocp-network

namespace: openshift-mtv

spec:

map:

- source:

name: "VM Network"

destination:

type: pod

- source:

name: "Production Network"

destination:

type: pod

YAML

oc apply -f /opt/migration/network-map.yaml

echo "Storage and network mapping configured"

oc get storagemap,networkmap -n openshift-mtv

EOF

# chmod +x /opt/migration/scripts/configure\_migration\_infrastructure.sh

# ./configure\_migration\_infrastructure.sh

### **Stage 3: Migration Execution**

#### **Create vSphere Provider and Migration Plans**

Create migration execution script

cat > /opt/migration/scripts/execute\_migration.sh <<'EOF'

#!/bin/bash

echo "Setting up migration from vSphere to OpenShift"

# Get the latest assessment data

LATEST\_DATA\_DIR="/opt/migration/data/$(ls -t /opt/migration/data/ | head -1)"

MIGRATION\_PLAN\_CSV="$LATEST\_DATA\_DIR/application\_assessment/migration\_plan.csv"

if [ ! -f "$MIGRATION\_PLAN\_CSV" ]; then

echo "Migration plan not found. Run assessment first."

exit 1

fi

# Create vSphere provider credentials secret

read -p "Enter vSphere username: " VSPHERE\_USER

read -s -p "Enter vSphere password: " VSPHERE\_PASS

echo

oc create secret generic vsphere-credentials -n openshift-mtv \

--from-literal=user="$VSPHERE\_USER" \

--from-literal=password="$VSPHERE\_PASS" \

--dry-run=client -o yaml | oc apply -f -

# Create vSphere provider

cat > /opt/migration/vsphere-provider.yaml <<YAML

apiVersion: forklift.konveyor.io/v1beta1

kind: Provider

metadata:

name: vsphere-source

namespace: openshift-mtv

spec:

type: vsphere

url: https://vsphere.example.com/sdk

secret:

name: vsphere-credentials

namespace: openshift-mtv

YAML

oc apply -f /opt/migration/vsphere-provider.yaml

# Wait for provider to be ready

echo "Waiting for vSphere provider to be ready"

oc wait --for=condition=Ready provider/vsphere-source -n openshift-mtv --timeout=300s

# Create host provider for OpenShift destination

cat > /opt/migration/host-provider.yaml <<YAML

apiVersion: forklift.konveyor.io/v1beta1

kind: Provider

metadata:

name: ocp-destination

namespace: openshift-mtv

spec:

type: openshift

YAML

oc apply -f /opt/migration/host-provider.yaml

echo "Providers created and ready"

oc get providers -n openshift-mtv

# Generate migration plans based on assessment

echo "Generating migration plans"

# Create Python script to generate migration plans from CSV

cat > /opt/migration/generate\_migration\_plans.py <<'PYTHON'

import csv

import yaml

import sys

import os

plan\_csv = sys.argv[1]

output\_dir = sys.argv[2]

# Read migration plan CSV

vms\_by\_type = {

'LIFT\_AND\_SHIFT': [],

'CONTAINERIZE': [],

'REPLATFORM': []

}

with open(plan\_csv, 'r') as f:

reader = csv.DictReader(f)

for row in reader:

if row['migration\_type'] in vms\_by\_type:

vms\_by\_type[row['migration\_type']].append(row)

# Generate migration plans for each type

for migration\_type, vms in vms\_by\_type.items():

if not vms:

continue

plan\_name = f"migration-plan-{migration\_type.lower().replace('\_', '-')}"

vm\_list = []

for vm in vms[:5]: # Limit to 5 VMs per plan for POC

vm\_list.append({'name': vm['vm\_name']})

if not vm\_list:

continue

plan = {

'apiVersion': 'forklift.konveyor.io/v1beta1',

'kind': 'Plan',

'metadata': {

'name': plan\_name,

'namespace': 'openshift-mtv'

},

'spec': {

'provider': {

'source': {'name': 'vsphere-source'},

'destination': {'name': 'ocp-destination'}

},

'map': {

'network': {'name': 'vsphere-to-ocp-network'},

'storage': {'name': 'vsphere-to-ocp-storage'}

},

'targetNamespace': f'migrated-{migration\_type.lower().replace("\_", "-")}',

'vms': vm\_list

}

}

plan\_file = f"{output\_dir}/{plan\_name}.yaml"

with open(plan\_file, 'w') as f:

yaml.dump(plan, f, default\_flow\_style=False)

print(f"Generated migration plan: {plan\_file}")

print(f"VMs in plan: {len(vm\_list)}")

print("Migration plan generation complete")

PYTHON

# Run plan generation

python3 /opt/migration/generate\_migration\_plans.py "$MIGRATION\_PLAN\_CSV" "/opt/migration"

# Apply migration plans

echo "Creating migration plans in OpenShift"

for plan\_file in /opt/migration/migration-plan-\*.yaml; do

if [ -f "$plan\_file" ]; then

echo "Applying $(basename $plan\_file)"

oc apply -f "$plan\_file"

# Create namespace for migrated VMs

plan\_name=$(basename "$plan\_file" .yaml)

namespace\_name="migrated-${plan\_name#migration-plan-}"

oc new-project "$namespace\_name" 2>/dev/null || echo "Namespace $namespace\_name already exists"

fi

done

echo "Migration plans created:"

oc get plans -n openshift-mtv

echo "Ready to start migrations. Review plans and execute when ready."

EOF

# chmod +x /opt/migration/scripts/execute\_migration.sh

# ./execute\_migration.sh

## **Execute Actual Migration**

#### **Create migration execution script**

cat > /opt/migration/scripts/start\_migration.sh <<'EOF'

#!/bin/bash

echo "Starting VM migrations"

# List available plans

echo "Available migration plans:"

oc get plans -n openshift-mtv -o custom-columns="NAME:.metadata.name,VMS:.spec.vms[\*].name"

# Select plan for execution

read -p "Enter migration plan name to execute: " PLAN\_NAME

if ! oc get plan "$PLAN\_NAME" -n openshift-mtv &>/dev/null; then

echo "Migration plan '$PLAN\_NAME' not found"

exit 1

fi

# Create migration resource

MIGRATION\_NAME="migration-$(date +%Y%m%d-%H%M%S)"

cat > /opt/migration/${MIGRATION\_NAME}.yaml <<YAML

apiVersion: forklift.konveyor.io/v1beta1

kind: Migration

metadata:

name: $MIGRATION\_NAME

namespace: openshift-mtv

spec:

plan:

name: $PLAN\_NAME

namespace: openshift-mtv

YAML

oc apply -f /opt/migration/${MIGRATION\_NAME}.yaml

echo "Migration '$MIGRATION\_NAME' started"

echo "Monitor progress with:"

echo " oc get migration $MIGRATION\_NAME -n openshift-mtv -w"

echo " oc describe migration $MIGRATION\_NAME -n openshift-mtv"

# Monitor migration progress

echo "Monitoring migration progress (Ctrl+C to stop monitoring):"

while true; do

clear

echo "Migration Status:"

oc get migration $MIGRATION\_NAME -n openshift-mtv -o custom-columns="NAME:.metadata.name,PHASE:.status.phase,STARTED:.status.started,COMPLETED:.status.completed"

echo

# Get VM status

echo "VM Migration Status:"

oc get migration $MIGRATION\_NAME -n openshift-mtv -o jsonpath='{.status.vms[\*].name}' | tr ' ' '\n' | while read vm\_name; do

vm\_phase=$(oc get migration $MIGRATION\_NAME -n openshift-mtv -o jsonpath="{.status.vms[?(@.name=='$vm\_name')].phase}")

echo " $vm\_name: $vm\_phase"

done

# Check if migration completed

migration\_phase=$(oc get migration $MIGRATION\_NAME -n openshift-mtv -o jsonpath='{.status.phase}')

if [[ "$migration\_phase" == "Succeeded" || "$migration\_phase" == "Failed" ]]; then

echo "Migration completed with status: $migration\_phase"

break

fi

sleep 30

done

echo "Migration execution complete"

EOF

# chmod +x /opt/migration/scripts/start\_migration.sh

#### **Post-Migration Validation**

Create validation script

cat > /opt/migration/scripts/validate\_migration.sh <<'EOF'

#!/bin/bash

echo "Validating migrated VMs"

# Get all migrated VM namespaces

MIGRATED\_NAMESPACES=$(oc get namespaces -o name | grep "migrated-" | cut -d'/' -f2)

if [ -z "$MIGRATED\_NAMESPACES" ]; then

echo "No migrated VM namespaces found"

exit 1

fi

echo "Found migrated namespaces: $MIGRATED\_NAMESPACES"

# Validate each namespace

for ns in $MIGRATED\_NAMESPACES; do

echo "=== Validating namespace: $ns ==="

# Check VMs

echo "Virtual Machines:"

oc get vm -n $ns -o custom-columns="NAME:.metadata.name,STATUS:.status.printableStatus,RUNNING:.status.ready"

echo "VM Instances:"

oc get vmi -n $ns -o custom-columns="NAME:.metadata.name,PHASE:.status.phase,NODE:.status.nodeName,IP:.status.interfaces[0].ipAddress"

# Check PVCs

echo "Persistent Volume Claims:"

oc get pvc -n $ns -o custom-columns="NAME:.metadata.name,STATUS:.status.phase,CAPACITY:.status.capacity.storage"

# Check events for issues

echo "Recent Events:"

oc get events -n $ns --sort-by=.metadata.creationTimestamp | tail -10

echo "----------------------------------------"

done

# Generate validation report

REPORT\_FILE="/opt/migration/validation\_report\_$(date +%Y%m%d\_%H%M%S).txt"

echo "Migration Validation Report" > $REPORT\_FILE

echo "==========================" >> $REPORT\_FILE

echo "Date: $(date)" >> $REPORT\_FILE

echo "" >> $REPORT\_FILE

for ns in $MIGRATED\_NAMESPACES; do

echo "Namespace: $ns" >> $REPORT\_FILE

echo "VMs:" >> $REPORT\_FILE

oc get vm -n $ns -o custom-columns="NAME:.metadata.name,STATUS:.status.printableStatus" --no-headers >> $REPORT\_FILE

echo "" >> $REPORT\_FILE

done

echo "Validation report saved to: $REPORT\_FILE"

# Check resource utilization

echo "Resource Utilization:"

echo "Nodes:"

oc adm top nodes

echo "Pods in migrated namespaces:"

for ns in $MIGRATED\_NAMESPACES; do

echo "Namespace $ns:"

oc adm top pods -n $ns 2>/dev/null || echo " No running pods"

done

echo "Migration validation complete"

EOF

chmod +x /opt/migration/scripts/validate\_migration.sh

**Complete Migration Workflow**

# Create master execution script

cat > /opt/migration/run\_complete\_migration.sh <<'EOF'

#!/bin/bash

echo "vSphere to OpenShift Migration POC - Complete Workflow"

echo "====================================================="

cd /opt/migration/scripts

# Phase 1: Assessment

echo "Phase 1: Running assessment and discovery"

if [ ! -f "/opt/migration/data/$(ls -t /opt/migration/data/ 2>/dev/null | head -1 2>/dev/null)/vm\_inventory.csv" 2>/dev/null ]; then

echo "Running vSphere discovery..."

./vsphere\_discovery.sh

echo "Running application assessment..."

./app\_assessment.sh

else

echo "Assessment data found, skipping discovery"

fi

# Phase 2: OCP Setup

echo "Phase 2: Setting up OpenShift for migration"

read -p "Is OpenShift migration infrastructure already set up? (y/n): " SETUP\_DONE

if [[ "$SETUP\_DONE" != "y" ]]; then

./setup\_ocp\_migration.sh

./configure\_migration\_infrastructure.sh

fi

# Phase 3: Migration

echo "Phase 3: Executing migration"

read -p "Ready to create migration plans? (y/n): " CREATE\_PLANS

if [[ "$CREATE\_PLANS" == "y" ]]; then

./execute\_migration.sh

fi

echo "Migration setup complete. Next steps:"

echo "1. Review migration plans: oc get plans -n openshift-mtv"

echo "2. Start migration: ./start\_migration.sh"

echo "3. Validate results: ./validate\_migration.sh"

echo "Migration Toolkit UI:"

oc get route forklift-ui -n openshift-mtv -o jsonpath='https://{.spec.host}'

EOF

# chmod +x /opt/migration/run\_complete\_migration.sh

**Summary**

This POC migration setup provides:

**Assessment Phase**: Complete vSphere infrastructure discovery and application complexity analysis.

**Setup Phase**: OpenShift Virtualization and Migration Toolkit for Virtualization configuration

**Migration Phase**: Automated migration plan creation and execution

**Key Files Generated**:

* **vm\_inventory.csv -** Complete VM inventory with resources
* **migration\_plan.csv** - Migration strategy per VM
* **assessment\_summary.txt** - High-level migration recommendations
* Migration plans and validation reports

**Next Steps After POC**:

1. Review assessment results and refine migration strategies
2. Execute pilot migrations with non-critical workloads
3. Develop runbooks for production migration phases
4. Plan network and security configurations for production

The scripts are designed to work with minimal changes - just update the vSphere credentials and OCP connection details. Each phase can be run independently, making it easy to iterate and refine the migration approach.

# Appendix:

### Infrastructure validation script

cat > validate\_infrastructure.sh <<'EOF'

#!/bin/bash

echo "Validating OCP infrastructure readiness"

# Check bastion connectivity to all nodes

NODES="haproxy01 master01 master02 master03 worker01 worker02 worker03"

for node in $NODES; do

if ping -c 2 ${node}.ocp.local &>/dev/null; then

echo "✓ ${node}.ocp.local - Reachable"

else

echo "✗ ${node}.ocp.local - Unreachable"

fi

done

# Check DNS resolution

echo "DNS Resolution Test:"

nslookup api.ocp.local

nslookup \*.apps.ocp.local

# Check HAProxy status

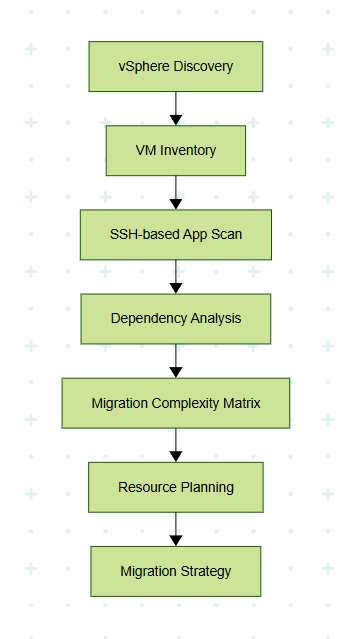
echo "HAProxy Status:"

curl -s http://haproxy01.ocp.local:8404/stats | grep -E "(master|worker|bootstrap)" || echo "HAProxy stats not available"

echo "Infrastructure validation complete"

EOF

### Assessment Workflow:



#### Assessment Data Processing:

##### Create assessment data aggregation script

cat > aggregate\_assessment\_data.sh <<'EOF'

#!/bin/bash

echo "Aggregating assessment data for migration planning"

ASSESSMENT\_DIR="/opt/migration/data/$(ls -t /opt/migration/data/ | head -1)"

# Generate migration wave planning

cat > ${ASSESSMENT\_DIR}/migration\_waves.py <<'PYTHON'

import pandas as pd

import matplotlib.pyplot as plt

# Read migration plan

df = pd.read\_csv('application\_assessment/migration\_plan.csv')

# Generate migration waves based on complexity

waves = {

'Wave 1 - Quick Wins': df[df['complexity'] == 'LOW'],

'Wave 2 - Medium Complexity': df[df['complexity'] == 'MEDIUM'],

'Wave 3 - High Complexity': df[df['complexity'] == 'HIGH'],

'Wave 4 - Manual Review': df[df['complexity'] == 'UNKNOWN']

}

# Create wave summary

wave\_summary = []

for wave\_name, wave\_vms in waves.items():

total\_cpu = wave\_vms['cpu'].sum()

total\_memory = wave\_vms['memory\_gb'].sum()

total\_storage = wave\_vms['disk\_gb'].sum()

vm\_count = len(wave\_vms)

wave\_summary.append({

'wave': wave\_name,

'vm\_count': vm\_count,

'total\_cpu': total\_cpu,

'total\_memory\_gb': total\_memory,

'total\_storage\_gb': total\_storage,

'estimated\_duration\_hours': vm\_count \* 2 # 2 hours per VM average

})

wave\_df = pd.DataFrame(wave\_summary)

wave\_df.to\_csv('migration\_waves\_summary.csv', index=False)

print("Migration waves analysis complete")

print(wave\_df.to\_string(index=False))

PYTHON

cd $ASSESSMENT\_DIR && python3 migration\_waves.py

echo "Assessment aggregation complete"

EOF

### Service Dependencies:

A diagram of a process

AI-generated content may be incorrect.

#### Cluster Readiness Validation Script:

cat > **validate\_cluster\_readiness.sh** <<'EOF'

#!/bin/bash

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

echo "Validating OpenShift cluster readiness for VM migration"

# Check cluster operators

echo "=== Cluster Operators Status ==="

oc get co | grep -E "(AVAILABLE|VERSION)" | head -1

oc get co | grep -v "True.\*False.\*False"

# Check OpenShift Virtualization

echo "=== OpenShift Virtualization Status ==="

oc get hyperconverged -n openshift-cnv -o custom-columns="NAME:.metadata.name,AVAILABLE:.status.conditions[?(@.type=='Available')].status"

# Check Migration Toolkit

echo "=== Migration Toolkit Status ==="

oc get forkliftcontroller -n openshift-mtv -o custom-columns="NAME:.metadata.name,READY:.status.conditions[?(@.type=='Successful')].status"

# Check storage classes

echo "=== Storage Classes ==="

oc get sc | grep -E "(local-vm-storage|default)"

# Check worker node resources

echo "=== Worker Node Resources ==="

oc get nodes -l node-role.kubernetes.io/worker="" -o custom-columns="NODE:.metadata.name,CPU:.status.allocatable.cpu,MEMORY:.status.allocatable.memory,STORAGE:.status.allocatable.ephemeral-storage"

# Check CNV feature gates

echo "=== CNV Feature Status ==="

oc get kubevirt -n openshift-cnv -o yaml | grep -A 10 "featureGates"

# Generate readiness report

cat > cluster\_readiness\_report.txt <<REPORT

OpenShift Cluster Readiness Report

=================================

Date: $(date)

Cluster: $(oc whoami --show-server)

Operators Status:

$(oc get co --no-headers | awk '{print $1 ": " $3 "/" $4 "/" $5}')

Virtualization Ready: $(oc get hyperconverged -n openshift-cnv --no-headers | wc -l)

Migration Toolkit Ready: $(oc get forkliftcontroller -n openshift-mtv --no-headers | wc -l)

Storage Classes Available:

$(oc get sc --no-headers | awk '{print $1}')

Available Worker Resources:

$(oc describe nodes -l node-role.kubernetes.io/worker="" | grep -E "Allocatable|cpu:|memory:|ephemeral-storage:")

REPORT

echo "Cluster readiness validation complete"

echo "Report saved: cluster\_readiness\_report.txt"

EOF

### Migration Process Flow:

A diagram with colorful squares and text

AI-generated content may be incorrect.

#### Migration Execution Orchestration:

Create comprehensive migration orchestration script

cat > orchestrate\_migration.sh <<'EOF'

#!/bin/bash

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

echo "Migration Orchestration - vSphere to OpenShift"

# Get latest assessment data

LATEST\_ASSESSMENT="/opt/migration/data/$(ls -t /opt/migration/data/ | head -1)/application\_assessment"

# Create migration execution plan

cat > migration\_execution\_plan.sh <<'EXEC\_PLAN'

#!/bin/bash

echo "Executing phased migration based on assessment"

# Phase 1: Lift and Shift (Low Complexity)

echo "=== Phase 1: Lift and Shift Migration ==="

PHASE1\_VMS=$(awk -F',' '$6=="LOW" && $7=="LIFT\_AND\_SHIFT" {print $1}' migration\_plan.csv | head -3)

for vm in $PHASE1\_VMS; do

echo "Migrating $vm (Lift and Shift)"

# Create migration plan for this VM

cat > ${vm}-migration.yaml <<VMPLAN

apiVersion: forklift.konveyor.io/v1beta1

kind: Plan

metadata:

name: ${vm}-plan

namespace: openshift-mtv

spec:

provider:

source: {name: vsphere-source}

destination: {name: ocp-destination}

map:

network: {name: vsphere-to-ocp-network}

storage: {name: vsphere-to-ocp-storage}

targetNamespace: migrated-lift-and-shift

vms:

- name: ${vm}

VMPLAN

oc apply -f ${vm}-migration.yaml

# Start migration

cat > ${vm}-migration-start.yaml <<MIGRATION

apiVersion: forklift.konveyor.io/v1beta1

kind: Migration

metadata:

name: ${vm}-migration

namespace: openshift-mtv

spec:

plan:

name: ${vm}-plan

namespace: openshift-mtv

MIGRATION

oc apply -f ${vm}-migration-start.yaml

# Wait for completion

echo "Waiting for $vm migration to complete..."

oc wait --for=condition=Succeeded migration/${vm}-migration -n openshift-mtv --timeout=3600s

if [ $? -eq 0 ]; then

echo "✓ $vm migrated successfully"

else

echo "✗ $vm migration failed"

fi

done

# Phase 2: Containerization Candidates (Medium Complexity)

echo "=== Phase 2: Containerization Candidates ==="

PHASE2\_VMS=$(awk -F',' '$6=="MEDIUM" && $7=="CONTAINERIZE" {print $1}' migration\_plan.csv | head -3)

for vm in $PHASE2\_VMS; do

echo "Migrating $vm (Containerize)"

# Similar process but with containerization namespace

# Implementation follows same pattern as Phase 1

done

# Phase 3: Database Replatforming (High Complexity)

echo "=== Phase 3: Database Replatforming ==="

PHASE3\_VMS=$(awk -F',' '$6=="HIGH" && $7=="REPLATFORM" {print $1}' migration\_plan.csv | head -2)

for vm in $PHASE3\_VMS; do

echo "Migrating $vm (Replatform)"

# Database-specific migration handling

# May require additional pre/post migration steps

done

echo "All migration phases completed"

EXEC\_PLAN

# Execute migration plan

cd $LATEST\_ASSESSMENT

chmod +x migration\_execution\_plan.sh

./migration\_execution\_plan.sh

echo "Migration orchestration complete"

EOF

### **Complete Migration Monitoring Dashboard:**

cat > migration\_dashboard.sh <<'EOF'

#!/bin/bash

export KUBECONFIG="/opt/ocp-install/configs/auth/kubeconfig"

echo "Migration Status Dashboard"

echo "========================="

# Real-time migration status

while true; do

clear

echo "OpenShift VM Migration Dashboard - $(date)"

echo "==========================================="

# Migration status

echo "Active Migrations:"

oc get migrations -n openshift-mtv -o custom-columns="NAME:.metadata.name,PLAN:.spec.plan.name,PHASE:.status.phase,STARTED:.status.started,VMS:.status.vms[\*].name" 2>/dev/null || echo "No active migrations"

echo ""

echo "Migrated VMs by Namespace:"

for ns in migrated-lift-and-shift migrated-containerize migrated-replatform; do

if oc get ns $ns &>/dev/null; then

vm\_count=$(oc get vm -n $ns --no-headers 2>/dev/null | wc -l)

running\_count=$(oc get vmi -n $ns --no-headers 2>/dev/null | wc -l)

echo " $ns: $vm\_count VMs ($running\_count running)"

fi

done

echo ""

echo "Resource Utilization:"

echo "Nodes:"

oc adm top nodes 2>/dev/null | head -5

echo ""

echo "Storage Usage:"

oc get pvc --all-namespaces | grep -E "(migrated-|Bound)" | wc -l | xargs echo "Total PVCs:"

echo ""

echo "Press Ctrl+C to exit monitoring..."

sleep 30

done

EOF

# chmod +x migration\_dashboard.sh

## **Diagrams:**