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.

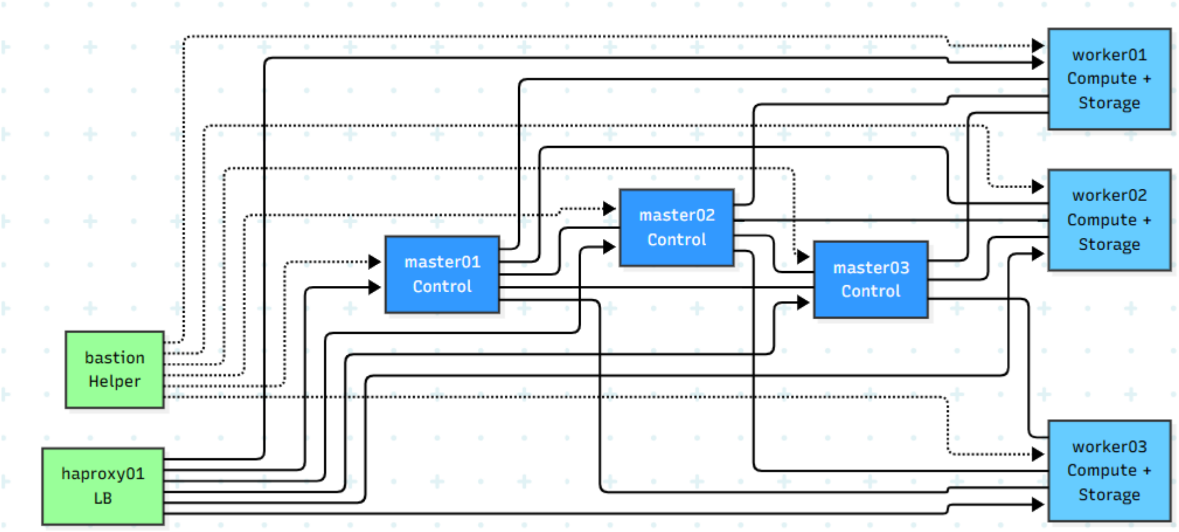
The goal is proving we can migrate vSphere workloads to OpenShift reliably while maintaining service levels and creating a repeatable migration factory for production rollouts.
Total migration timeline: 1-2 weeks for complete POC validation

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:



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_$(una
me -s)_$(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.start
ed,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:.s
tatus.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.cap
acity.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.

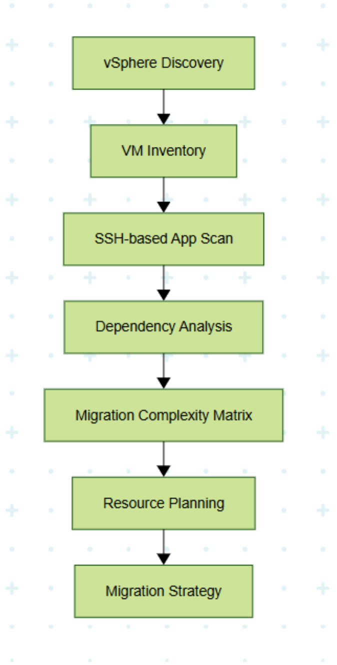
Appendix:

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 "X ${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:



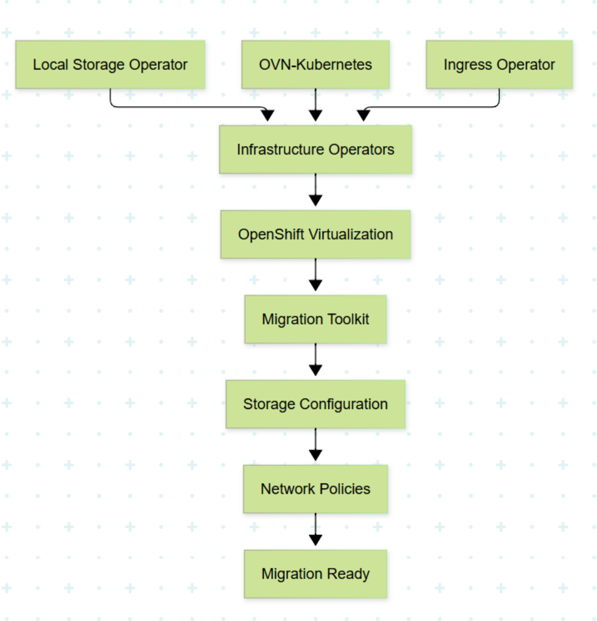
Assessment Data Processing:

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:

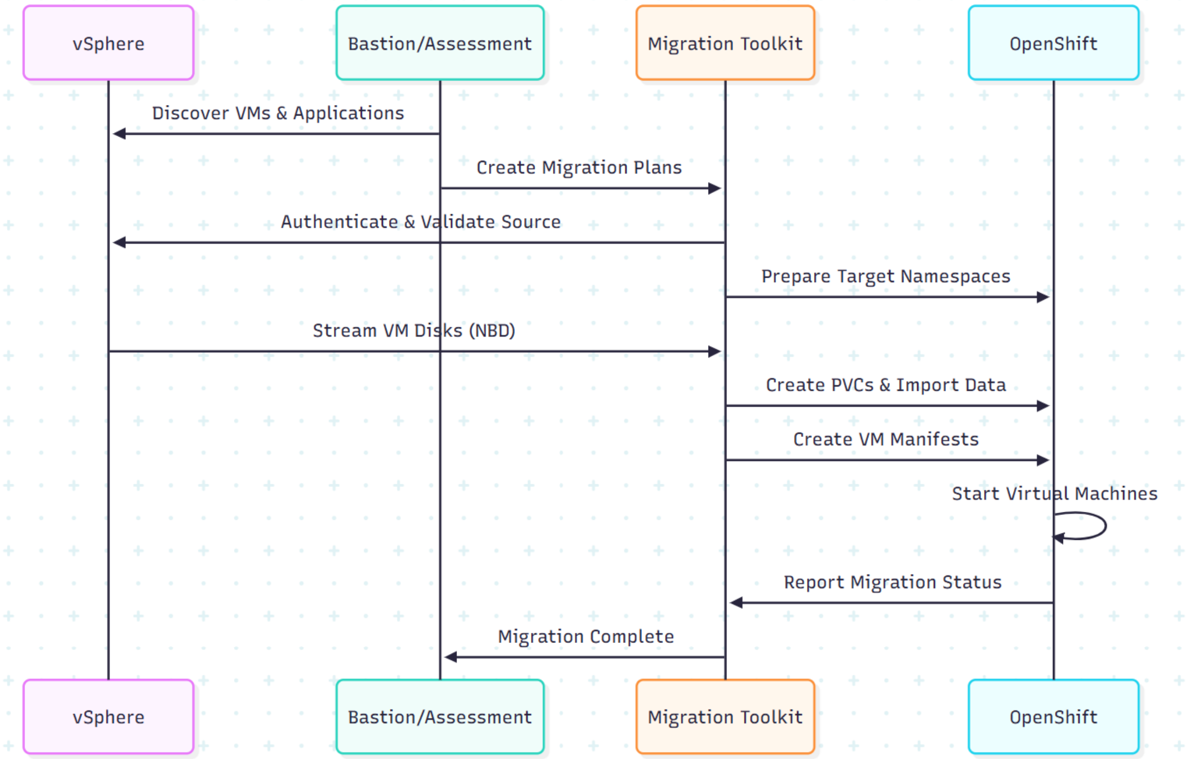


Cluster Readiness Validation Script:

```
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.alloca
table.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:



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:

VC-OCP Infra Workflow.svg

VC-OCP Migration Infrastructure Layou

Browser file - Setup workflow.svg

mermaid code - Setup workflow.mm

Mermaid code - Infra layout.mmd