Environment Overview

Deployment Architecture

This guide deploys a FIPS-compliant OpenShift 4.18 cluster with full Platform Plus capabilities in a completely disconnected air-gapped environment. The deployment includes OpenShift Virtualization for running VMs alongside containers.

Cluster Specifications:

- **Cluster Name**: ovetest
- **Base Domain**: test.com
- **OpenShift Version**: 4.18
- **Network**: 192.168.1.0/24
- **Deployment Type**: Fully disconnected air-gapped with FIPS compliance
- **Features**: OpenShift Virtualization, Advanced Cluster Management, OpenShift Data Foundation

Infrastructure Components:

- 3x Bare metal OpenShift master nodes (host01-03)
- 1x NFS storage server (dedicated bare metal)
- 6x Supporting VMs on vSphere cluster
- Single 1GbE network infrastructure

IP Address Allocation

```
# Supporting Infrastructure
192.168.1.10    tmpregistry.test.com     (Mirror Registry)
192.168.1.11    octools.test.com         (OpenShift Tools - FIPS enabled)
192.168.1.13    dns.test.com             (DNS/NTP Server)
192.168.1.13    content.test.com         (Content Server)
192.168.1.15    nfs.test.com             (NFS Storage Server)

# OpenShift Infrastructure
192.168.1.21    host01.ove.test.com      (Master Node 1)
192.168.1.22    host02.ove.test.com      (Master Node 2)
192.168.1.23    host03.ove.test.com      (Master Node 3)
192.168.1.25    bootstrap.ove.test.com   (Bootstrap Node)

# OpenShift Services
192.168.1.30    api.ove.test.com         (API VIP)
192.168.1.30    api-int.ove.test.com     (Internal API VIP)
192.168.1.31    *.apps.ove.test.com      (Applications VIP)
```

Prerequisites and Planning

Required Expertise

- OpenShift 4.18 administration and troubleshooting
- RHEL system administration and FIPS compliance
- DNS/NTP service configuration
- Container registry management and mirroring
- Network infrastructure management

Critical FIPS Requirements

⚠ **MANDATORY**: FIPS compliance must be configured from the beginning - it cannot be enabled after installation.

- **Installation Host**: The octools VM must be FIPS-enabled before running any OpenShift installation commands
- **All Systems**: RHEL systems should be FIPS-compliant for consistency
- **Validation**: FIPS mode verification is required at each phase

Hardware Requirements

<u>Bare Metal Servers (4 total):</u>

- 3x OpenShift Master Nodes: 500GB OS drive, 1TB data drive, single 1GbE NIC
- 1x NFS Storage Server: 3TB SSD storage, single 1GbE NIC

vSphere Cluster Requirements:

- Sufficient resources for 6 VMs (total: 28 vCPU, 68GB RAM, 2.3TB storage)
- Network connectivity to 192.168.1.0/24

Pre-Deployment Validation Checklist

- ✓ All bare metal servers have been wiped clean
- ✓ Network infrastructure (192.168.1.0/24) is operational
- ✓ vSphere cluster has sufficient resources allocated
- ✓ External connectivity available for initial image pulling
- ✓ DNS domain (test.com) is available for use
- ✓ Red Hat pull secret downloaded and accessible

PHASE 1: EXTERNAL INFRASTRUCTURE SETUP (INTERNET CONNECTED)

Context: This phase is performed on systems WITH internet access to download and prepare all required OpenShift images and content for transfer to the air-gapped environment.

Step 1.1: Create External Mirror Host (OUTSIDE AIR GAP)

Purpose: Download OpenShift images and operator content from Red Hat registries

Create VM with: 4c, 16GB, 1TB storage, RHEL9, WITH INTERNET ACCESS

```
# Install required packages for image mirroring
sudo dnf install -y podman curl wget jq

# Download oc-mirror v2 tool
curl -L https://console.redhat.com/openshift/downloads/tool-mirror-registry -o oc-
mirror.tar.gz
tar xvf oc-mirror.tar.gz
sudo mv oc-mirror /usr/local/bin/
chmod +x /usr/local/bin/oc-mirror

# Verify oc-mirror installation
oc-mirror --v2 --help

# Create working directory for mirror operations
mkdir -p ~/mirror-workspace
cd ~/mirror-workspace
```

Step 1.2: Configure Image Mirroring Content

Purpose: Define what OpenShift content to mirror for the disconnected installation

```
# Download your Red Hat pull secret from https://console.redhat.com/openshift/install/pull-secret
# Save it as pull-secret.json in current directory

# Create comprehensive ImageSetConfiguration for Platform Plus
cat << 'EOF' > imageset-config.yaml
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v2alpha1
mirror:
  platform:
    channels:
      - name: stable-4.18
        minVersion: 4.18.0
        maxVersion: 4.18.20
        type: ocp
    graph: true
    kubeVirtContainer: true
  additionalImages:
    # Base Container Images
    - name: registry.redhat.io/ubi8/ubi:latest
    - name: registry.redhat.io/ubi9/ubi:latest

    # RHEL Guest Images for Virtualization
    - name: registry.redhat.io/rhel9/rhel-guest-image:latest
    - name: registry.redhat.io/rhel8/rhel-guest-image:latest

    # Support and Troubleshooting Tools
    - name: registry.redhat.io/rhel8/support-tools:latest
    - name: registry.redhat.io/rhel9/support-tools:latest

    # Must-Gather Images for Debugging
    - name: registry.redhat.io/openshift4/ose-must-gather:latest
    - name: registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.18
    - name: registry.redhat.io/odf4/odf-must-gather-rhel9:v4.18

    # NFS CSI Driver Dependencies
    - name: registry.k8s.io/sig-storage/csi-provisioner:v5.2.0
    - name: registry.k8s.io/sig-storage/csi-attacher:v4.8.0
    - name: registry.k8s.io/sig-storage/csi-resizer:v1.13.2
    - name: registry.k8s.io/sig-storage/csi-snapshotter:v8.2.0
    - name: registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.13.0

  operators:
```

```
# Red Hat Operator Catalog
- catalog: registry.redhat.io/redhat/redhat-operator-index:v4.18
  packages:
    # Core Platform Services
    - name: advanced-cluster-management
    - name: multicluster-engine
    - name: multicluster-global-hub-operator-rh
    - name: openshift-cert-manager-operator
    - name: nfd
    - name: windows-machine-config-operator

    # Storage and Data Management
    - name: odf-operator
    - name: odf-csi-addons-operator
    - name: odf-dependencies
    - name: odf-prometheus-operator
    - name: odf-multicluster-orchestrator
    - name: rook-ceph-operator
    - name: redhat-oadp-operator

    # Virtualization Components
    - name: kubevirt-hyperconverged
    - name: kubernetes-nmstate-operator
    - name: quay-operator
    - name: quay-bridge-operator

    # Observability and Logging
    - name: cluster-logging
    - name: openshift-logging
    - name: cluster-observability-operator
    - name: loki-operator

    # CI/CD and Developer Tools
    - name: openshift-pipelines-operator-rh
    - name: openshift-gitops-operator
    - name: web-terminal
    - name: ansible-automation-platform-operator

    # Security and Compliance
    - name: compliance-operator
    - name: container-security-operator
    - name: security-profiles-operator

# Certified Operator Catalog
- catalog: registry.redhat.io/redhat/certified-operator-index:v4.18
  packages:
    # Storage Solutions
```

```
        - name: trident-operator
EOF
```

Step 1.3: Mirror Images to Disk

Purpose: Download all required images to local storage for transfer to air-gapped environment

```
# Authenticate with Red Hat registries
podman login registry.redhat.io
podman login quay.io

# Create mirror directory
mkdir -p /tmp/openshift-mirror

# Mirror images to disk using oc-mirror v2
echo "🔄 Starting image mirroring to disk (this may take several hours)..."
oc-mirror -c imageset-config.yaml file:///tmp/openshift-mirror --v2

# Verify mirror completed successfully
if [ $? -eq 0 ]; then
    echo "✅ Image mirroring completed successfully"
else
    echo "❌ Image mirroring failed - check logs and retry"
    exit 1
fi

# Create compressed archive for transfer
echo "📦 Creating transfer archive..."
cd /tmp/openshift-mirror
tar -czf openshift-4.18-fips-mirror.tar.gz working-dir/

# Verify archive created
ls -lh openshift-4.18-fips-mirror.tar.gz
echo "✅ Mirror archive ready for transfer: openshift-4.18-fips-mirror.tar.gz"
echo "📋 Archive size: $(du -h openshift-4.18-fips-mirror.tar.gz | cut -f1)"
```

Step 1.4: Transfer to Air-Gapped Environment

Purpose: Move the mirrored content to the disconnected environment

```
# Transfer methods (choose based on your environment):

# Option 1: Physical media transfer
echo "💿 Copy archive to removable media for physical transfer"
cp openshift-4.18-fips-mirror.tar.gz /path/to/removable/media/

# Option 2: Secure file transfer (if temporary connection allowed)
# scp openshift-4.18-fips-mirror.tar.gz user@octools.test.com:/tmp/

echo "⚠️  IMPORTANT: Transfer the following to air-gapped environment:"
echo "   - openshift-4.18-fips-mirror.tar.gz"
echo "   - imageset-config.yaml"
echo "   - pull-secret.json"
```

Phase 2: Air-Gapped Infrastructure Preparation

Context: This phase is performed INSIDE the air-gapped environment to prepare the base infrastructure that will support the OpenShift cluster.

Step 2.1: Prepare Bare Metal Infrastructure

Purpose: Wipe and prepare the physical servers that will host OpenShift

```
# For each bare metal server (host01, host02, host03, nfs-server):

# 1. Boot from network or USB and wipe all disks
echo "🔧 Wiping bare metal servers..."

# On each server, wipe the OS disk
sudo dd if=/dev/zero of=/dev/sda bs=1M count=1000
sudo wipefs -a /dev/sda

# On OpenShift hosts, also wipe the data disk
sudo dd if=/dev/zero of=/dev/sdb bs=1M count=1000
sudo wipefs -a /dev/sdb

# Configure static IP addressing for each server
# host01: 192.168.1.21
# host02: 192.168.1.22
# host03: 192.168.1.23
# nfs-server: 192.168.1.15
```

Step 2.2: Create vSphere VMs

Purpose: Create the supporting virtual machines needed for OpenShift deployment

Create VMs with the following specifications:

- Hostname: dns
  - CPU: 4vCPU
  - MEM: 4GB
  - DISK: 100GB
- Hostname: content
  - CPU: 4vCPU
  - MEM: 4GB
  - DISK: 100GB
  - OS: REHL9
- Hostname: mirror-registry
  - CPU: 8vCPU
  - MEM: 16GB
  - DISK: 1TB
  - OS: REHL9
- Hostname: octools
  - CPU: 8vCPU
  - MEM: 16GB
  - DISK: 200GB
  - OS: REHL9
- Hostname: bootstrap
  - CPU: 4vCPU
  - MEM: 16GB
  - DISK: 120GB
  - OS: RHCOS

Phase 3: Supporting Services Configuration

Context: Configure the essential services (DNS, NTP, NFS, HTTP, Registry) that OpenShift requires to function in the disconnected environment.

Step 3.1: Configure DNS and NTP Server

Purpose: Provide name resolution and time synchronization for all OpenShift components

```
# Install required packages
sudo dnf install -y bind bind-utils chrony

echo "🕐 Configuring internal NTP server..."
# Configure chronyd as NTP server for the air-gapped environment
sudo tee /etc/chrony.conf << 'EOF'
```

```
# Use public servers as upstream (only during initial setup if internet available)
# Comment out these lines after initial sync in air-gapped environment
pool 2.rhel.pool.ntp.org iburst

# Allow clients from local network
allow 192.168.1.0/24

# Serve time even if not synchronized to a source (important for air-gapped)
local stratum 3

# Record the rate at which the system clock gains/loses time
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC)
rtcsync

# Increase the minimum number of selectable sources required to adjust
# the system clock
minsources 2

# Allow NTP client access from local network
clientloglimit 1000000

# Specify directory for log files
logdir /var/log/chrony

# Select which information is logged
log measurements statistics tracking
EOF

# Enable and start chronyd
sudo systemctl enable --now chronyd

# Configure firewall for NTP
sudo firewall-cmd --permanent --add-service=ntp
sudo firewall-cmd --reload

echo "🌐 Configuring DNS server..."
# Configure BIND DNS server
sudo tee /etc/named.conf << 'EOF'
options {
    listen-on port 53 { 127.0.0.1; 192.168.1.13; };
    listen-on-v6 port 53 { ::1; };
    directory "/var/named";
    dump-file "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    allow-query { localhost; 192.168.1.0/24; };
    recursion yes;

    forwarders {
        8.8.8.8;
        8.8.4.4;
    };

    dnssec-enable yes;
    dnssec-validation yes;
};

logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};

zone "." IN {
    type hint;
    file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";

zone "test.com" IN {
    type master;
    file "test.com.zone";
    allow-update { none; };
};

zone "1.168.192.in-addr.arpa" IN {
    type master;
    file "1.168.192.in-addr.arpa.zone";
    allow-update { none; };
};
EOF

# Create forward DNS zone
sudo tee /var/named/test.com.zone << 'EOF'
$TTL 86400
```

```
@   IN  SOA dns.test.com. admin.test.com. (
        2024010101  ; Serial
        3600        ; Refresh
        1800        ; Retry
        1209600     ; Expire
        86400 )     ; Minimum TTL

; Name servers
@       IN  NS      dns.test.com.

; Supporting Infrastructure
dns                 IN  A   192.168.1.13
octools             IN  A   192.168.1.11
tmpregistry         IN  A   192.168.1.10
content             IN  A   192.168.1.13
nfs                 IN  A   192.168.1.15

; OpenShift Infrastructure
host01.ove          IN  A   192.168.1.21
host02.ove          IN  A   192.168.1.22
host03.ove          IN  A   192.168.1.23
bootstrap.ove       IN  A   192.168.1.25

; OpenShift API and Apps
api.ove             IN  A   192.168.1.30
api-int.ove         IN  A   192.168.1.30
*.apps.ove          IN  A   192.168.1.31

# Create reverse DNS zone
sudo tee /var/named/1.168.192.in-addr.arpa.zone << 'EOF'
$TTL 86400
@   IN  SOA dns.test.com. admin.test.com. (
        2024010101  ; Serial
        3600        ; Refresh
        1800        ; Retry
        1209600     ; Expire
        86400 )     ; Minimum TTL

@       IN  NS      dns.test.com.

; PTR records
13      IN  PTR     dns.test.com.
11      IN  PTR     octools.test.com.
10      IN  PTR     tmpregistry.test.com.
13      IN  PTR     content.test.com.
15      IN  PTR     nfs.test.com.
21      IN  PTR     host01.ove.test.com.
```

```
22      IN  PTR     host02.ove.test.com.
23      IN  PTR     host03.ove.test.com.
25      IN  PTR     bootstrap.ove.test.com.
30      IN  PTR     api.ove.test.com.
31      IN  PTR     apps.ove.test.com.
EOF

# Set proper permissions and start DNS
sudo chown named:named /var/named/*.zone
sudo chmod 644 /var/named/*.zone

# Validate DNS configuration
named-checkconf
named-checkzone test.com /var/named/test.com.zone
named-checkzone 1.168.192.in-addr.arpa /var/named/1.168.192.in-addr.arpa.zone

# Start and enable DNS services
sudo systemctl enable --now named

# Configure firewall
sudo firewall-cmd --permanent --add-service=dns
sudo firewall-cmd --reload
```

Step 3.2: Configure NFS Storage Server

Purpose: Provide persistent storage for OpenShift components and applications

```
# Install NFS packages on the dedicated bare metal server
sudo dnf install -y nfs-utils

echo "💾 Configuring NFS storage with LVM..."
# Create logical volumes for different storage tiers
sudo pvcreate /dev/sdb  # Assuming 3TB disk is /dev/sdb
sudo vgcreate storage-vg /dev/sdb

# Create logical volumes for SSD and HDD storage classes
sudo lvcreate -L 1500G -n ssd-lv storage-vg
sudo lvcreate -L 1400G -n hdd-lv storage-vg

# Format filesystems
sudo mkfs.xfs /dev/storage-vg/ssd-lv
sudo mkfs.xfs /dev/storage-vg/hdd-lv

# Create mount points
sudo mkdir -p /exports/nfs-ssd
sudo mkdir -p /exports/nfs-hdd
```

```
# Add to fstab for persistent mounting
echo "/dev/storage-vg/ssd-lv /exports/nfs-ssd xfs defaults 0 0" | sudo tee -a /etc/fstab
echo "/dev/storage-vg/hdd-lv /exports/nfs-hdd xfs defaults 0 0" | sudo tee -a /etc/fstab

# Mount filesystems
sudo mount -a

# Configure NFS exports for OpenShift
sudo tee /etc/exports << 'EOF'
/exports/nfs-ssd 192.168.1.0/24(rw,sync,no_root_squash,no_all_squash,security_label)
/exports/nfs-hdd 192.168.1.0/24(rw,sync,no_root_squash,no_all_squash,security_label)
EOF

# Enable and start NFS services
sudo systemctl enable --now rpcbind
sudo systemctl enable --now nfs-server

# Export filesystems
sudo exportfs -rv

# Configure firewall for NFS
sudo firewall-cmd --permanent --add-service=nfs
sudo firewall-cmd --permanent --add-service=mountd
sudo firewall-cmd --permanent --add-service=rpc-bind
sudo firewall-cmd --reload

# Test NFS exports
showmount -e localhost
```

Step 3.3: Configure Content Server

Purpose: Host RHCOS images and ignition files for OpenShift node installation

```
# Install Apache HTTP Server
sudo dnf install -y httpd

echo "📁 Configuring content server directories..."
# Create content directories
sudo mkdir -p /var/www/html/rhcos
sudo mkdir -p /var/www/html/ignition
sudo mkdir -p /var/www/html/manifests

# Configure Apache
sudo systemctl enable --now httpd

# Configure firewall
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload

# Set SELinux context for web content
sudo setsebool -P httpd_read_user_content 1
sudo restorecon -R /var/www/html
```

Step 3.4: Configure Mirror Registry

Purpose: Host mirrored container images for OpenShift installation and operations

```
# Install Podman
sudo dnf install -y podman

echo "🏭 Installing mirror registry..."
# Download and install mirror-registry (if not already transferred)
curl -LO https://console.redhat.com/openshift/downloads/tool-mirror-registry
tar xvf mirror-registry.tar.gz

# Install mirror registry with self-signed certificates
sudo ./mirror-registry install \
  --quayHostname tmpregistry.test.com \
  --quayRoot /opt/quay-install

# Note: Save the generated credentials output from installation
echo "⚠️  IMPORTANT: Save the generated registry credentials!"

# Configure firewall
sudo firewall-cmd --permanent --add-port=8443/tcp
sudo firewall-cmd --reload

# Test registry access
echo "🔐 Testing registry access..."
podman login -u init -p <generated_password> tmpregistry.test.com:8443 --tls-verify=false
```

Phase 4: Image Mirroring and Transfer

Context: Transfer the mirrored images from external environment and populate the internal mirror registry.

Step 4.1: Prepare FIPS-Enabled Tools Host

Purpose: Set up the host that will run OpenShift installation tools with FIPS compliance

⚠️ **CRITICAL**: This host MUST be FIPS-enabled before running any OpenShift installation commands.

```
echo "🔒 Enabling FIPS mode on octools host..."
# Enable FIPS mode FIRST - this requires a reboot
sudo fips-mode-setup --enable
echo "⚠️  REBOOTING to enable FIPS mode..."
sudo reboot

# After reboot, verify FIPS is enabled
echo "🔍 Verifying FIPS mode..."
fips-mode-setup --check
if [ $? -ne 0 ]; then
    echo "❌ ERROR: FIPS mode is not enabled!"
    exit 1
fi

# Verify FIPS enabled flag
cat /proc/sys/crypto/fips_enabled  # Should show "1"

# Verify crypto policy is set to FIPS
update-crypto-policies --show  # Should show "FIPS"

# Test FIPS compliance
echo "test" | openssl md5 2>&1 && echo "❌ ERROR: MD5 should be disabled in FIPS mode!" || echo "✅ MD5 properly disabled - FIPS compliant"

echo "📦 Installing OpenShift tools on FIPS-enabled host..."
# Install required packages
sudo dnf install -y curl wget tmux vim jq git

# Download and install OpenShift tools
# Download oc CLI
curl -L https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable-4.18/openshift-client-linux.tar.gz -o oc.tar.gz
tar xvf oc.tar.gz
sudo mv oc kubectl /usr/local/bin/
chmod +x /usr/local/bin/oc /usr/local/bin/kubectl

# Download oc-mirror v2
curl -L https://console.redhat.com/openshift/downloads/tool-mirror-registry -o oc-mirror.tar.gz
tar xvf oc-mirror.tar.gz
sudo mv oc-mirror /usr/local/bin/
chmod +x /usr/local/bin/oc-mirror

# Download openshift-install
curl -L https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable-4.18/openshift-install-linux.tar.gz -o openshift-install.tar.gz
```

```
tar xvf openshift-install.tar.gz
sudo mv openshift-install /usr/local/bin/
chmod +x /usr/local/bin/openshift-install

# Download butane
curl -L https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane-amd64
-o butane
sudo mv butane /usr/local/bin/
chmod +x /usr/local/bin/butane

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

# Configure system to use internal NTP
sudo tee /etc/chrony.conf << 'EOF'
# Use internal NTP server
server 192.168.1.13 iburst

# Record the rate at which the system clock gains/loses time
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC)
rtcsync

# Specify directory for log files
logdir /var/log/chrony

# Select which information is logged
log measurements statistics tracking
EOF

sudo systemctl restart chronyd

# Verify time sync with internal NTP
chrony sources -v

# Verify tool installations
echo "🔧 Verifying tool installations..."
oc version
oc-mirror --help
openshift-install version
butane --version
helm version
```

Step 4.2: Transfer and Extract Mirror Content

Purpose: Move the mirrored images into the air-gapped environment and extract them

```
# Transfer the mirror archive to octools VM (via USB/network as appropriate)
echo "📦 Extracting mirror content..."
cd /tmp
# Assuming archive was transferred to /tmp/openshift-4.18-fips-mirror.tar.gz
tar -xzf openshift-4.18-fips-mirror.tar.gz

# Verify extraction
ls -la /tmp/working-dir/
```

Step 4.3: Configure Registry Authentication

Purpose: Set up authentication to push images to the internal mirror registry

```
echo "🔐 Configuring registry authentication..."
# Get registry certificate for trust
openssl s_client -connect tmpregistry.test.com:8443 -showcerts | awk '/BEGIN/,/END/{print $0}' | sudo tee /etc/pki/ca-trust/source/anchors/tmpregistry.crt
sudo update-ca-trust

# Configure authentication for internal registry
mkdir -p ~/.docker
cat << 'EOF' > ~/.docker/config.json
{
  "auths": {
    "tmpregistry.test.com:8443": {
      "auth": "<base64_encoded_init:password>",
      "email": "admin@test.com"
    }
  }
}
EOF

# Create authentication credentials (replace with actual base64 encoded credentials)
echo -n 'init:<registry_password>' | base64 -w0

# Test registry authentication
podman login tmpregistry.test.com:8443 --tls-verify=false
```

Step 4.4: Mirror Images to Internal Registry

Purpose: Push all OpenShift images from disk to the internal mirror registry

```
echo "🚀 Mirroring images to internal registry..."
# This process uploads all images to the internal registry
oc-mirror -c /tmp/imageset-config.yaml --from file:///tmp/working-dir
docker://tmpregistry.test.com:8443 --v2

# Verify mirroring completed successfully
if [ $? -eq 0 ]; then
    echo "✅ Image mirroring to internal registry completed successfully"
else
    echo "❌ Image mirroring failed - check logs and retry"
    exit 1
fi

# Verify cluster resources were generated
ls -la /tmp/working-dir/cluster-resources/
```

PHASE 5: OPENSHIFT INSTALLATION

Context: Install the OpenShift cluster using the mirrored images and FIPS-compliant configuration.

Step 5.1: Pre-Installation FIPS Validation

Purpose: Ensure FIPS compliance before starting OpenShift installation

```
echo "🔒 === MANDATORY FIPS Pre-Installation Verification ==="
echo "1. FIPS Mode Status:"
fips-mode-setup --check
if [ $? -ne 0 ]; then
    echo "❌ ERROR: FIPS mode is not enabled on this host!"
    echo "You MUST enable FIPS mode and reboot before proceeding:"
    echo "sudo fips-mode-setup --enable"
    echo "sudo reboot"
    exit 1
fi

echo "2. FIPS Enabled Flag:"
FIPS_ENABLED=$(cat /proc/sys/crypto/fips_enabled 2>/dev/null)
if [ "$FIPS_ENABLED" != "1" ]; then
    echo "❌ ERROR: FIPS is not properly enabled (fips_enabled = $FIPS_ENABLED)"
    exit 1
fi
```

```
echo "3. Crypto Policy:"
CRYPTO_POLICY=$(update-crypto-policies --show)
if [ "$CRYPTO_POLICY" != "FIPS" ]; then
    echo "❌ ERROR: Crypto policy is not set to FIPS (current: $CRYPTO_POLICY)"
    echo "Run: sudo update-crypto-policies --set FIPS"
    exit 1
fi

echo "4. OpenSSL FIPS Test:"
echo "test" | openssl md5 2>&1 && echo "❌ ERROR: MD5 should be disabled in FIPS mode!" && exit 1
echo "✅ MD5 properly disabled - FIPS compliant"

echo ""
echo "✅ FIPS verification PASSED - proceeding with OpenShift installation"
echo ""
```

Step 5.2: Download and Prepare RHCOS Images

Purpose: Obtain the Red Hat CoreOS images needed for node installation

```
echo "💿 Downloading RHCOS images..."
mkdir -p ~/openshift-install
cd ~/openshift-install

# Download RHCOS images for OpenShift 4.18
curl -L https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.18/latest/rhcos-live.x86_64.iso -o rhcos-live.iso
curl -L https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.18/latest/rhcos-metal.x86_64.raw.gz -o rhcos-metal.raw.gz

# Copy RHCOS images to content server for network access
scp rhcos-* root@content.test.com:/var/www/html/rhcos/

echo "✅ RHCOS images available on content server"
```

Step 5.3: Create Installation Configuration

Purpose: Generate the OpenShift installation configuration with FIPS enabled

```
echo "⚙️ Creating FIPS-enabled install-config.yaml..."
# Create install-config.yaml with FIPS enabled
cat << 'EOF' > install-config.yaml
apiVersion: v1
baseDomain: test.com
metadata:
  name: ove
```

```
networking:
  networkType: OVNKubernetes
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  machineNetwork:
  - cidr: 192.168.1.0/24
compute:
- name: worker
  replicas: 0
controlPlane:
  name: master
  replicas: 3
  platform:
    baremetal: {}
platform:
  baremetal:
    apiVIPs:
    - 192.168.1.30
    ingressVIPs:
    - 192.168.1.31
    hosts:
    - name: host01
      role: master
      bmc:
        address: redfish://192.168.1.21
        username: admin
        password: password
      bootMACAddress: "aa:bb:cc:dd:ee:01"
      rootDeviceHints:
        deviceName: "/dev/sda"
    - name: host02
      role: master
      bmc:
        address: redfish://192.168.1.22
        username: admin
        password: password
      bootMACAddress: "aa:bb:cc:dd:ee:02"
      rootDeviceHints:
        deviceName: "/dev/sda"
    - name: host03
      role: master
      bmc:
        address: redfish://192.168.1.23
        username: admin
        password: password
      bootMACAddress: "aa:bb:cc:dd:ee:03"
      rootDeviceHints:
        deviceName: "/dev/sda"
fips: true
pullSecret: |
  <your_pull_secret_with_internal_registry>
sshKey: |
  <your_ssh_public_key>
imageContentSources:
- mirrors:
  - tmpregistry.test.com:8443/openshift/release-images
  source: quay.io/openshift-release-dev/ocp-release
- mirrors:
  - tmpregistry.test.com:8443/openshift/release
  source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
additionalTrustBundle: |
  -----BEGIN CERTIFICATE-----
  <registry_certificate_content>
  -----END CERTIFICATE-----
EOF

# Verify FIPS is enabled in configuration
grep -q "fips: true" install-config.yaml && echo "✅ FIPS enabled in install-config.yaml" || echo "❌ ERROR: FIPS not enabled in install-config.yaml"
```

Step 5.4: Generate Custom Manifests

Purpose: Create custom configurations for internal NTP and FIPS compliance

```
echo "📄 Creating cluster manifests..."
openshift-install create manifests

echo "🕐 Creating custom NTP configuration manifests..."
# Create custom NTP configuration for masters
cat << 'EOF' > manifests/99-master-chrony-configuration.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-master-chrony-configuration
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyBVc2UgaW50ZXJuYWwgTlRQIHNlcnZlcgpzZXJ2ZXIgMTkyLjE2OC4xLjEzIGlidXJzdAoKIyBSZWNv
cmQgdGhlIHJhdGUgYXQgd2hpY2ggdGhlIHN5c3RlbSBjbG9jayBnYWlucy9sb3NlcyB0aW1lCmRyaWZ0ZmlsZSAvd
mFyL2xpYi9jaHJvbnkvZHJpZnQKCiMgQWxsb3cgdGhlIHN5c3RlbSBjbG9jayB0byBiZSBzdGVwcGVkIGluIHRoZS
BmaXJzdCB0aHJlZSB1cGRhdGVzCm1ha2VzdGVwIDEuMCAzCgojIEVuYWJsZSBrZXJuZWwgc3luY2hyb25pemF0aW9
uIG9mIHRoZSByZWFsLXRpbWUgY2xvY2sgKFJUQykKcnRjc3luYwoKIyBJbmNyZWFzZSB0aGUgbWluaW11bSBudW1i
ZXIgb2Ygc2VsZWN0YWJsZSBzb3VyY2VzIHJlcXVpcmVkIHRvIGFkanVzdAojIHRoZSBzeXN0ZW0gY2xvY2sKbWluc
291cmNlcyAxCgojIFNwZWNpZnkgZGlyZWN0b3J5IGZvciBsb2cgZmlsZXMKbG9nZGlyIC92YXIvbG9nL2Nocm9ueQ
oKIyBTZWxlY3Qgd2hpY2ggaW5mb3JtYXRpb24gaXMgbG9nZ2VkCmxvZyBtZWFzdXJlbWVudHMgc3RhdGlzdGljcyB
0cmFja2luZwo=
        mode: 420
        overwrite: true
        path: /etc/chrony.conf
    systemd:
      units:
      - enabled: true
        name: chronyd.service
EOF

# Create custom NTP configuration for workers
cat << 'EOF' > manifests/99-worker-chrony-configuration.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-chrony-configuration
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyBVc2UgaW50ZXJuYWwgTlRQIHNlcnZlcgpzZXJ2ZXIgMTkyLjE2OC4xLjEzIGlidXJzdAoKIyBSZWNv
cmQgdGhlIHJhdGUgYXQgd2hpY2ggdGhlIHN5c3RlbSBjbG9jayBnYWlucy9sb3NlcyB0aW1lCmRyaWZ0ZmlsZSAvd
mFyL2xpYi9jaHJvbnkvZHJpZnQKCiMgQWxsb3cgdGhlIHN5c3RlbSBjbG9jayB0byBiZSBzdGVwcGVkIGluIHRoZS
BmaXJzdCB0aHJlZSB1cGRhdGVzCm1ha2VzdGVwIDEuMCAzCgojIEVuYWJsZSBrZXJuZWwgc3luY2hyb25pemF0aW9
uIG9mIHRoZSByZWFsLXRpbWUgY2xvY2sgKFJUQykKcnRjc3luYwoKIyBJbmNyZWFzZSB0aGUgbWluaW11bSBudW1i
ZXIgb2Ygc2VsZWN0YWJsZSBzb3VyY2VzIHJlcXVpcmVkIHRvIGFkanVzdAojIHRoZSBzeXN0ZW0gY2xvY2sKbWluc
291cmNlcyAxCgojIFNwZWNpZnkgZGlyZWN0b3J5IGZvciBsb2cgZmlsZXMKbG9nZGlyIC92YXIvbG9nL2Nocm9ueQ
oKIyBTZWxlY3Qgd2hpY2ggaW5mb3JtYXRpb24gaXMgbG9nZ2VkCmxvZyBtZWFzdXJlbWVudHMgc3RhdGlzdGljcyB
0cmFja2luZwo=
        mode: 420
        overwrite: true
        path: /etc/chrony.conf
    systemd:
      units:
      - enabled: true
        name: chronyd.service
EOF

echo "🔒 Creating FIPS kernel argument manifests..."
# Create FIPS compliance manifest for additional kernel arguments
cat << 'EOF' > manifests/99-master-fips-kargs.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-master-fips-kargs
spec:
  kernelArguments:
  - fips=1
EOF

cat << 'EOF' > manifests/99-worker-fips-kargs.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-fips-kargs
spec:
  kernelArguments:
  - fips=1
EOF
```

Step 5.5: Generate Ignition Configurations

Purpose: Create the ignition files that will configure the OpenShift nodes

```
echo "🔥 Generating ignition configurations..."
# Create ignition configs
openshift-install create ignition-configs

# Verify ignition files were created
ls -la *.ign
echo "✅ Ignition files generated:"
echo "   - bootstrap.ign (for bootstrap node)"
echo "   - master.ign (for master nodes)"
echo "   - worker.ign (for worker nodes)"

# Copy ignition files to content server for network access during installation
echo "📤 Copying ignition files to content server..."
scp *.ign root@content.test.com:/var/www/html/ignition/
```

Step 5.6: Install Bootstrap Node

Purpose: Create and start the temporary bootstrap node that orchestrates cluster installation

```
echo "🚀 Starting bootstrap node installation..."
echo "📋 Bootstrap installation steps:"
echo "1. Create bootstrap VM with 4c, 16GB, 120GB SSD"
echo "2. Boot from RHCOS live ISO"
echo "3. Run coreos-installer with bootstrap ignition"

# Manual steps to perform on bootstrap VM:
echo "
🖥️  On the bootstrap VM console:
sudo coreos-installer install /dev/sda \\
  --ignition-url http://content.test.com/ignition/bootstrap.ign \\
  --insecure-ignition

# Wait for installation to complete, then reboot
sudo reboot
"

echo "⏳ Waiting for bootstrap node to become ready..."
echo "   Monitor logs: journalctl -b -f -u release-image.service -u bootkube.service"
```

Step 5.7: Install Control Plane Nodes

Purpose: Install the three master nodes that will form the OpenShift control plane

```
echo "🎯 Installing master nodes..."
echo "📋 Master node installation steps:"

# For each master node (host01, host02, host03):
echo "
🖥️  For each master node, boot from RHCOS live ISO and run:

# On host01 (192.168.1.21):
sudo coreos-installer install /dev/sda \\
  --ignition-url http://content.test.com/ignition/master.ign \\
  --insecure-ignition
```

```
# On host02 (192.168.1.22):
sudo coreos-installer install /dev/sda \\
  --ignition-url http://content.test.com/ignition/master.ign \\
  --insecure-ignition

# On host03 (192.168.1.23):
sudo coreos-installer install /dev/sda \\
  --ignition-url http://content.test.com/ignition/master.ign \\
  --insecure-ignition

# After installation, reboot each node
sudo reboot
"

echo "⏳ Waiting for master nodes to join the cluster..."
```

Step 5.8: Monitor Installation Progress

Purpose: Track the installation progress and handle the bootstrap removal

```
echo "📊 Monitoring OpenShift installation progress..."

# Monitor bootstrap completion
echo "⏳ Waiting for bootstrap to complete..."
openshift-install wait-for bootstrap-complete --log-level=info

if [ $? -eq 0 ]; then
    echo "✅ Bootstrap phase completed successfully"
    echo "🗑️  Bootstrap node can now be safely removed"
else
    echo "❌ Bootstrap phase failed - check logs"
    exit 1
fi

# At this point, remove or shutdown the bootstrap VM
echo "📋 Remove bootstrap VM from load balancer and power down"

# Wait for installation to complete
echo "⏳ Waiting for installation to complete..."
openshift-install wait-for install-complete --log-level=info

if [ $? -eq 0 ]; then
    echo "🎉 OpenShift installation completed successfully!"
    echo "📋 Cluster access information:"
    cat auth/kubeconfig
else
    echo "❌ Installation failed - check logs and troubleshoot"
    exit 1
fi

echo "✅ OpenShift 4.18 cluster with FIPS compliance is now running"
```

Phase 6: Post-Installation Configuration

Context: Configure the newly installed OpenShift cluster with mirrored content and basic settings.

Step 6.1: Configure Cluster Authentication

Purpose: Set up access to the newly installed cluster

```
echo "🔑 Configuring cluster access..."
# Set up kubeconfig for cluster access
export KUBECONFIG=~/openshift-install/auth/kubeconfig

# Test cluster access
oc get nodes
oc get clusterversion

# Show cluster information
echo "📋 Cluster Information:"
oc cluster-info
oc get clusteroperators

echo "✅ Cluster access configured"
```

Step 6.2: Apply Mirrored Content Configuration

Purpose: Configure the cluster to use the mirrored images and operator catalogs

```
echo "🔄 Applying mirrored content configuration..."
# Apply the generated cluster resources from oc-mirror
oc apply -f /tmp/working-dir/cluster-resources/

# Verify ImageDigestMirrorSet and ImageTagMirrorSet are applied
oc get imagedigestmirrorset
oc get imagetagmirrorset

# Verify CatalogSources are available
oc get catalogsource -n openshift-marketplace

# Wait for catalog sources to be ready
echo "⏳ Waiting for catalog sources to become ready..."
```

```
oc wait --for=condition=Ready catalogsource --all -n openshift-marketplace --timeout=300s

echo "✅ Mirrored content configuration applied successfully"
```

Step 6.3: Validate FIPS Compliance

Purpose: Ensure the cluster is running in FIPS mode as expected

```
echo "🔒 Validating cluster FIPS compliance..."

# Check FIPS status on all nodes
for node in $(oc get nodes -o jsonpath='{.items[*].metadata.name}'); do
    echo "🔍 Checking FIPS on $node:"
    oc debug node/$node -- chroot /host fips-mode-setup --check
    echo "FIPS kernel arg: $(oc debug node/$node -- chroot /host cat /proc/cmdline 
2>/dev/null | grep -o 'fips=[0-9]*')"
    echo "FIPS enabled flag: $(oc debug node/$node -- chroot /host cat 
/proc/sys/crypto/fips_enabled 2>/dev/null)"
    echo "---"
done

# Verify cluster-wide FIPS configuration
oc get proxy cluster -o yaml | grep -i fips

echo "✅ FIPS compliance validation completed"
```

Phase 7: Platform Plus Components Installation

Context: Install the additional Platform Plus components including storage, virtualization, and management tools.

Step 7.1: Configure NFS Storage Classes

Purpose: Set up persistent storage using the NFS server for OpenShift components

```
echo "💾 Configuring NFS storage classes..."

# Deploy NFS CSI driver
echo "📦 Installing NFS CSI driver..."
oc create namespace nfs-csi-driver

# Create NFS CSI driver deployment
cat << 'EOF' | oc apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: csi-nfs-controller
  namespace: nfs-csi-driver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: csi-nfs-controller
  template:
    metadata:
      labels:
        app: csi-nfs-controller
    spec:
      serviceAccountName: csi-nfs-controller-sa
      containers:
      - name: csi-provisioner
        image: tmpregistry.test.com:8443/sig-storage/csi-provisioner:v5.2.0
        args:
        - "--csi-address=$(ADDRESS)"
        - "--v=2"
        - "--leader-election=true"
        - "--leader-election-namespace=nfs-csi-driver"
        env:
        - name: ADDRESS
          value: /csi/csi.sock
        volumeMounts:
        - mountPath: /csi
          name: socket-dir
      - name: csi-nfs-driver
        image: tmpregistry.test.com:8443/sig-storage/nfsplugin:v4.9.0
        args:
        - "--endpoint=$(CSI_ENDPOINT)"
        - "--nodeid=$(NODE_ID)"
        - "--v=2"
        env:
        - name: CSI_ENDPOINT
          value: unix:///csi/csi.sock
        - name: NODE_ID
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        volumeMounts:
        - mountPath: /csi
          name: socket-dir
      volumes:
      - name: socket-dir
        emptyDir: {}
EOF

# Create SSD storage class
echo "⚡ Creating NFS SSD storage class..."
cat << 'EOF' | oc apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-ssd
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: nfs.csi.k8s.io
parameters:
  server: 192.168.1.15
  share: /exports/nfs-ssd
  mountPermissions: "0755"
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:
  - hard
  - nfsvers=4.1
EOF

# Create HDD storage class
echo "💿 Creating NFS HDD storage class..."
cat << 'EOF' | oc apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-hdd
provisioner: nfs.csi.k8s.io
parameters:
  server: 192.168.1.15
  share: /exports/nfs-hdd
  mountPermissions: "0755"
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:
  - hard
  - nfsvers=4.1
EOF

echo "✅ NFS storage classes configured"
```

Step 7.2: Configure Image Registry with Persistent Storage

Purpose: Set up the internal image registry to use persistent storage

```
echo "🖼️ Configuring image registry with persistent storage..."

# Create PVC for image registry
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: image-registry-storage
  namespace: openshift-image-registry
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 500Gi
  storageClassName: nfs-ssd
EOF

# Configure image registry to use NFS storage
oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch
'{"spec":{"storage":{"pvc":{"claim":"image-registry-storage"}}}}'

# Set image registry to managed
oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch
'{"spec":{"managementState":"Managed"}}'

# Verify image registry deployment
echo "⏳ Waiting for image registry to become ready..."
oc wait --for=condition=Available deployment/image-registry -n openshift-image-registry -
-timeout=300s
```

```
echo "✅ Image registry configured with persistent storage"
```

Step 7.3: Install OpenShift Virtualization

Purpose: Enable VM workloads alongside containers

```
echo "🖥️ Installing OpenShift Virtualization..."

# Create namespace for OpenShift Virtualization
oc create namespace openshift-cnv

# Create OperatorGroup
cat << 'EOF' | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
  - openshift-cnv
EOF

# Create Subscription for OpenShift Virtualization
echo "📦 Installing kubevirt-hyperconverged operator..."
cat << 'EOF' | oc apply -f -
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"
EOF

# Wait for operator installation
echo "⏳ Waiting for OpenShift Virtualization operator to install..."
oc wait --for=condition=Succeeded csv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv -n openshift-cnv --timeout=600s

# Create HyperConverged instance with FIPS-compatible settings
echo "🔧 Creating HyperConverged instance..."
```

```
cat << 'EOF' | oc apply -f -
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
  workloads:
    nodePlacement:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
  featureGates:
    deployTektonTaskResources: false
    disableMDevConfiguration: false
    enableCommonBootImageImport: true
    deployVmConsoleProxy: true
  certConfig:
    ca:
      duration: 8760h0m0s
      renewBefore: 720h0m0s
    server:
      duration: 8760h0m0s
      renewBefore: 720h0m0s
  tlsSecurityProfile:
    type: Intermediate
    intermediate:
      ciphers:
      - ECDHE-ECDSA-AES128-GCM-SHA256
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES256-GCM-SHA384
      - ECDHE-RSA-AES256-GCM-SHA384
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      minTLSVersion: VersionTLS12
EOF

# Monitor deployment progress
echo "⏳ Waiting for OpenShift Virtualization to become ready..."
oc wait --for=condition=Available hco kubevirt-hyperconverged -n openshift-cnv --
timeout=1200s

echo "✅ OpenShift Virtualization installed and ready"
```

Step 7.4: Install Advanced Cluster Management

Purpose: Enable multi-cluster management capabilities

```
echo "🌐 Installing Advanced Cluster Management..."

# Create namespace
oc create namespace open-cluster-management

# Create OperatorGroup
cat << 'EOF' | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: open-cluster-management
  namespace: open-cluster-management
spec:
  targetNamespaces:
  - open-cluster-management
EOF

# Install ACM operator
echo "📦 Installing advanced-cluster-management operator..."
cat << 'EOF' | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: advanced-cluster-management
  namespace: open-cluster-management
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: advanced-cluster-management
  channel: "release-2.12"
EOF

# Wait for operator installation
echo "⏳ Waiting for ACM operator to install..."
oc wait --for=condition=Succeeded csv -l operators.coreos.com/advanced-cluster-management.open-cluster-management -n open-cluster-management --timeout=600s
```

Step 7.5: Configure Monitoring with Persistent Storage

Purpose: Set up monitoring stack with persistent storage for metrics retention

```
echo "📊 Configuring monitoring with persistent storage..."
```

```
# Create monitoring storage configuration
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        spec:
          storageClassName: nfs-ssd
          resources:
            requests:
              storage: 100Gi
    alertmanagerMain:
      volumeClaimTemplate:
        spec:
          storageClassName: nfs-ssd
          resources:
            requests:
              storage: 10Gi
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: nfs-ssd
          resources:
            requests:
              storage: 10Gi
EOF

# Create user workload monitoring configuration
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: nfs-ssd
          resources:
            requests:
              storage: 50Gi
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: nfs-ssd
          resources:
            requests:
              storage: 10Gi
EOF

echo "✅ Monitoring configured with persistent storage"
```

Phase 8: Validation and Testing

Context: Validate that all components are working correctly and test FIPS compliance.

Step 8.1: Comprehensive FIPS Validation

Purpose: Ensure the entire cluster is FIPS-compliant

```
echo "🔒 === Comprehensive FIPS Compliance Validation ==="

# Create FIPS validation job
cat << 'EOF' | oc apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: fips-validation
spec:
  template:
    spec:
      containers:
      - name: fips-checker
        image: tmpregistry.test.com:8443/ubi9/ubi:latest
        command:
        - /bin/bash
        - -c
        - |
          echo "=== FIPS Mode Validation ==="
          fips-mode-setup --check
          echo "Exit code: $?"
          echo ""
          echo "=== Kernel Command Line ==="
          cat /proc/cmdline
          echo ""
          echo "=== OpenSSL FIPS Test ==="
          openssl md5 /dev/null 2>&1 || echo "FIPS mode detected - MD5 disabled as expected"
          echo ""
          echo "=== Crypto Policy ==="
          update-crypto-policies --show
          echo ""
          echo "=== FIPS Module Status ==="
          if [ -f /proc/sys/crypto/fips_enabled ]; then
            echo "FIPS enabled: $(cat /proc/sys/crypto/fips_enabled)"
          else
            echo "FIPS status file not found"
          fi
        securityContext:
          privileged: true
      restartPolicy: Never
      hostNetwork: true
      hostPID: true
  backoffLimit: 1
EOF

# Check FIPS validation results
echo "⏳ Waiting for FIPS validation to complete..."
oc wait --for=condition=Complete job/fips-validation --timeout=300s
oc logs job/fips-validation

# Clean up validation job
oc delete job fips-validation

# Verify FIPS compliance across all nodes
echo "🔍 Validating FIPS on all cluster nodes..."
for node in $(oc get nodes -o name); do
  echo "=== Checking FIPS on $node ==="
  oc debug $node -- chroot /host fips-mode-setup --check
done

echo "✅ FIPS validation completed"
```

Step 8.2: Test OpenShift Virtualization

Purpose: Verify VM capabilities are working correctly

```
echo "🖥️ Testing OpenShift Virtualization..."

# Create a test VM with FIPS compliance
```

```
cat << 'EOF' | oc apply -f -
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: test-vm-fips
  namespace: default
spec:
  running: true
  template:
    metadata:
      labels:
        kubevirt.io/vm: test-vm-fips
    spec:
      domain:
        cpu:
          cores: 2
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
          interfaces:
          - name: default
            masquerade: {}
        machine:
          type: pc-q35-rhel9.2.0
        resources:
          requests:
            memory: 2Gi
      networks:
      - name: default
        pod: {}
      volumes:
      - containerDisk:
          image: tmpregistry.test.com:8443/rhel9/rhel-guest-image:latest
        name: containerdisk
      - cloudInitNoCloud:
          userData: |
            #cloud-config
            users:
              - name: rhel
                sudo: ['ALL=(ALL) NOPASSWD:ALL']
                ssh_authorized_keys:
                  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAAACAQ...
            runcmd:
              - fips-mode-setup --enable
              - systemctl reboot
        name: cloudinitdisk
EOF

# Wait for VM to be ready
echo "⏳ Waiting for test VM to start..."
oc wait --for=condition=Ready vmi/test-vm-fips --timeout=300s

# Check VM status
oc get vm test-vm-fips
oc get vmi test-vm-fips

echo "✅ OpenShift Virtualization test VM created successfully"
```

Step 8.3: Test Storage Classes

Purpose: Verify persistent storage is working correctly

```
echo "💾 Testing storage classes..."

# Create test PVC for SSD storage
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-pvc-ssd-fips
  annotations:
    security.openshift.io/fips-compliant: "true"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: nfs-ssd
EOF

# Create test PVC for HDD storage
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-pvc-hdd-fips
  annotations:
    security.openshift.io/fips-compliant: "true"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: nfs-hdd
EOF

# Wait for PVCs to be bound
echo "⏳ Waiting for test PVCs to be bound..."
oc wait --for=condition=Bound pvc/test-pvc-ssd-fips --timeout=300s
oc wait --for=condition=Bound pvc/test-pvc-hdd-fips --timeout=300s

# Check PVC status
oc get pvc test-pvc-ssd-fips test-pvc-hdd-fips

echo "✅ Storage classes tested successfully"
```

Step 8.4: Test NTP Synchronization

Purpose: Verify all nodes are synchronized with internal NTP

```
echo "🕐 Testing NTP synchronization..."

# Create NTP test pod
cat << 'EOF' | oc apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: ntp-sync-test
spec:
  containers:
  - name: ntp-test
    image: tmpregistry.test.com:8443/ubi9/ubi:latest
    command:
    - /bin/bash
    - -c
    - |
      echo "=== NTP Synchronization Test ==="
      dnf install -y chrony
      echo "server 192.168.1.13 iburst" > /etc/chrony.conf
      systemctl start chronyd
      sleep 10
      echo "1. Chrony sources:"
      chrony sources -v
      echo ""
      echo "2. Chrony tracking:"
      chrony tracking
      echo ""
      echo "3. Time status:"
      timedatectl status
      echo ""
      echo "4. Testing NTP connectivity:"
      chrony sourcestats
      sleep 60
    securityContext:
      privileged: true
  hostNetwork: true
  restartPolicy: Never
EOF

# Monitor NTP test
echo "⏳ Running NTP synchronization test..."
oc wait --for=condition=Succeeded pod/ntp-sync-test --timeout=300s
oc logs ntp-sync-test

# Clean up test pod
oc delete pod ntp-sync-test

echo "✅ NTP synchronization test completed"
```

Step 8.5: Validate Operator Installation

Purpose: Ensure all Platform Plus operators are functioning correctly

```
echo "🔧 Validating operator installations..."

# Check all operator statuses
echo "📋 Checking subscription statuses:"
oc get subscription -A

echo ""
echo "📋 Checking CSV statuses:"
oc get csv -A

echo ""
echo "📋 Checking operator pod statuses:"
oc get pods -n openshift-cnv | grep -E "(Running|Completed)"
oc get pods -n open-cluster-management | grep -E "(Running|Completed)"
echo "✅ Operator validation completed"
```

Step 8.6: Final Cluster Health Check

Purpose: Comprehensive cluster health and functionality validation

```
echo "🏥 Performing final cluster health check..."

# Check cluster operators
echo "🔧 Cluster Operators Status:"
oc get clusteroperators

# Check node status
echo "🖥 Node Status:"
oc get nodes -o wide

# Check critical namespaces
echo "📦 Critical Namespace Status:"
for ns in openshift-kube-apiserver openshift-etcd openshift-kube-controller-manager openshift-kube-scheduler; do
  echo "Namespace: $ns"
  oc get pods -n $ns | grep -v Completed
  echo "---"
done

# Check storage
echo "💾 Storage Status:"
oc get storageclass
oc get pv

# Check catalog sources
echo "📋 Catalog Sources:"
oc get catalogsource -n openshift-marketplace

# Check virtualization
echo "🖥 Virtualization Status:"
oc get hco -n openshift-cnv
oc get kubevirt -n openshift-cnv

echo "✅ Final cluster health check completed"
```

TROUBLESHOOTING GUIDE

Context: Common issues and their solutions for the FIPS-enabled disconnected OpenShift deployment.

DNS and Network Issues

```
echo "🌐 DNS Troubleshooting Commands:"
```

```
# Test DNS resolution from nodes
oc debug node/host01.ove.test.com -- chroot /host nslookup api.ove.test.com

# Check DNS configuration on nodes
oc debug node/host01.ove.test.com -- chroot /host cat /etc/resolv.conf

# Verify DNS server is responding
dig @192.168.1.13 api.ove.test.com

# Test etcd SRV records
dig @192.168.1.13 _etcd-server-ssl._tcp.ove.test.com SRV

# Check network connectivity between nodes
oc debug node/host01.ove.test.com -- chroot /host ping host02.ove.test.com
```

Registry and Image Issues

```
echo "🏭 Registry Troubleshooting Commands:"

# Check if registry certificate is trusted
oc debug node/host01.ove.test.com -- chroot /host openssl s_client -connect
tmpregistry.test.com:8443 -verify_return_error

# Check ImageContentSourcePolicy configuration
oc get imagecontentsourcepolicy
oc get imagedigestmirrorset
oc get imagetagmirrorset

# Verify mirror configuration
oc get images.config.openshift.io cluster -o yaml

# Check node's registries.conf
oc debug node/host01.ove.test.com -- chroot /host cat /etc/containers/registries.conf

# Test image pull from mirror registry
oc debug node/host01.ove.test.com -- chroot /host podman pull
tmpregistry.test.com:8443/ubi9/ubi:latest

# Check registry pod logs
oc logs -n openshift-image-registry deployment/image-registry
```

FIPS Compliance Issues

```
echo "🔒 FIPS Troubleshooting Commands:"

# Check FIPS mode status on nodes
oc debug node/host01.ove.test.com -- chroot /host fips-mode-setup --check
```

```
# Verify FIPS kernel arguments
oc debug node/host01.ove.test.com -- chroot /host cat /proc/cmdline | grep fips

# Check FIPS compliance for OpenSSL
oc debug node/host01.ove.test.com -- chroot /host openssl md5 /dev/null

# Verify FIPS mode in containers
oc run fips-test --image=tmpregistry.test.com:8443/ubi9/ubi:latest --rm -it --
restart=Never -- fips-mode-setup --check

# Check FIPS enabled services
oc debug node/host01.ove.test.com -- chroot /host systemctl status fips-mode-setup

# Verify crypto policies
oc debug node/host01.ove.test.com -- chroot /host update-crypto-policies --show

# Check for FIPS-related machine configs
oc get machineconfig | grep fips
```

NTP Synchronization Issues

```
echo "🕐 NTP Troubleshooting Commands:"

# Check NTP synchronization status on nodes
oc debug node/host01.ove.test.com -- chroot /host chrony sources -v

# Verify chrony configuration is applied
oc debug node/host01.ove.test.com -- chroot /host cat /etc/chrony.conf

# Check time synchronization status
oc debug node/host01.ove.test.com -- chroot /host timedatectl status

# Test NTP connectivity to internal server
oc debug node/host01.ove.test.com -- chroot /host chrony sourcestats

# Check for time sync issues in logs
oc debug node/host01.ove.test.com -- chroot /host journalctl -u chronyd

# Verify NTP server is accessible
ping 192.168.1.13
telnet 192.168.1.13 123
```

Storage Issues

```
echo "💾 Storage Troubleshooting Commands:"
```

```
# Check NFS connectivity from nodes
oc debug node/host01.ove.test.com -- chroot /host showmount -e 192.168.1.15

# Test NFS mount
oc debug node/host01.ove.test.com -- chroot /host mount -t nfs 192.168.1.15:/exports/nfs-ssd /mnt

# Check storage class configuration
oc get storageclass nfs-ssd -o yaml

# Check CSI driver pods
oc get pods -n nfs-csi-driver

# Check PVC status
oc get pvc --all-namespaces

# Check persistent volume status
oc get pv

# Test storage performance
oc debug node/host01.ove.test.com -- chroot /host dd if=/dev/zero of=/tmp/test bs=1M count=100
```

OpenShift Virtualization Issues

```
echo "🖥️ Virtualization Troubleshooting Commands:"

# Check HyperConverged status
oc get hco -n openshift-cnv kubevirt-hyperconverged -o yaml

# Check KubeVirt status
oc get kv -n openshift-cnv kubevirt-kubevirt -o yaml

# Check virt-launcher pods
oc get pods -n openshift-cnv | grep virt-launcher

# Check node readiness for virtualization
oc get nodes -o json | jq '.items[].status.allocatable | {"kubevirt.io/kvm": .["kubevirt.io/kvm"], "tun": .["tun.network.kubevirt.io"]}'

# Check VM status
oc get vm --all-namespaces
oc get vmi --all-namespaces

# Check virtualization events
oc get events -n openshift-cnv --sort-by='.lastTimestamp'
```

```
# Verify virtualization features
oc debug node/host01.ove.test.com -- chroot /host lsmod | grep kvm
```

Operator Issues

```
echo "🔧 Operator Troubleshooting Commands:"

# Check subscription status
oc get subscription -A

# Check InstallPlan status
oc get installplan -A

# Check CSV status
oc get csv -A

# Check operator pod logs
oc logs -n openshift-cnv deployment/hco-operator
oc logs -n open-cluster-management deployment/multicluster-operators-hub-subscription

# Check catalog source status
oc get catalogsource -n openshift-marketplace

# Check for operator events
oc get events -n openshift-marketplace --sort-by='.lastTimestamp'

# Verify operator image availability
oc get csv -A -o
jsonpath='{.items[*].spec.install.spec.deployments[*].spec.template.spec.containers[*].im
age}' | tr ' ' '\n' | sort -u
```

Installation Issues

```
echo "🚀 Installation Troubleshooting Commands:"

# Check bootstrap logs
ssh core@bootstrap.ove.test.com sudo journalctl -b -f -u release-image.service -u
bootkube.service

# Check installer logs
openshift-install gather bootstrap --bootstrap 192.168.1.25 --master 192.168.1.21

# Check cluster operator status
oc get clusteroperators

# Check pending CSRs
oc get csr
```

```
# Check machine configs
oc get machineconfig
oc get machineconfigpool

# Check ignition files
curl -k http://content.test.com/ignition/bootstrap.ign | jq .
```

Performance Issues

```
echo "⚡ Performance Troubleshooting Commands:"

# Check resource usage on nodes
oc adm top nodes
oc adm top pods --all-namespaces

# Check disk I/O
oc debug node/host01.ove.test.com -- chroot /host iostat -x 1

# Check network performance
oc debug node/host01.ove.test.com -- chroot /host ss -tuln

# Check memory usage
oc debug node/host01.ove.test.com -- chroot /host free -m

# Check CPU usage
oc debug node/host01.ove.test.com -- chroot /host top -b -n1
```