AMD
together we advance_

RED HAT OPENSHIFT AI INTEGRATION AND VALIDATION GUIDE

VERSION: 1.3 | OCTOBER 2026

CONTENTS

1. Overview 5
Architecture Summary 5
Deployment Topology Options 5
2. Prerequisites 7
2.1 Hardware Requirements 7
2.2 Software Requirements 7
2.3 Tested Configuration 8
3. Setup Environment 10
3.1 Choose Your Installation Method 10
3.2 Install Required Tools 10
3.3 Configure Variables (Ansible Method Only) 10
3.4 Set KUBECONFIG 11
4. Verify Cluster Access 12
4.1 Method A: Ansible Playbook Validation 12
4.2 Method B: Manual CLI Validation 13
4.3 Health Assertions 13
5. Install AI/GPU Components 14
5.1 Component Installation Order 14
5.2 Method A: Ansible Playbook (Recommended) 15
5.3 Method B: Manual CLI Installation 15
5.4 Verify GPU Detection 25
5.5 Verify AI Components 26
6. Configure Red Hat OpenShift AI for Model Deployment 29
6.1 Overview 29
6.2 Method A: Ansible Playbook 29
6.3 Method B: Manual CLI 29
6.4 Storage for Model Cache 32
6.5 Create Required Secrets 35
6.6 Deploy Models via RHOAI Dashboard 35
6.7 Deploy Models via CLI 36
6.8 Monitor Deployment 39
6.9 Test Model Endpoint 40
6.10 Validate AIM Deployments 40
6.11 Generate Consolidated Report 40

Appendix **41**

A. Troubleshooting **41**

B. Teardown **43**

C. DNS Configuration for Routes **44**

D. AMD GPU Metrics **45**

E. Kernel Module Management (KMM) and AMD GPU Driver **47**

F. OpenShift Cluster Deployment **50**

G. Model Deployment via RHOAI Dashboard **56**

H. Setting up Samba for iDRAC Virtual Media **58**

I. References **63**

Disclaimer **63**

Trademarks and Product Names **64**

A comprehensive step-by-step guide for deploying and validating Red Hat OpenShift AI with AMD Instinct GPU support on x86 servers.

This guide provides **two installation methods**:

- **Ansible Playbooks** (Recommended) - Automated, repeatable deployments
- **Manual CLI** - Direct `oc` commands for learning or custom deployments

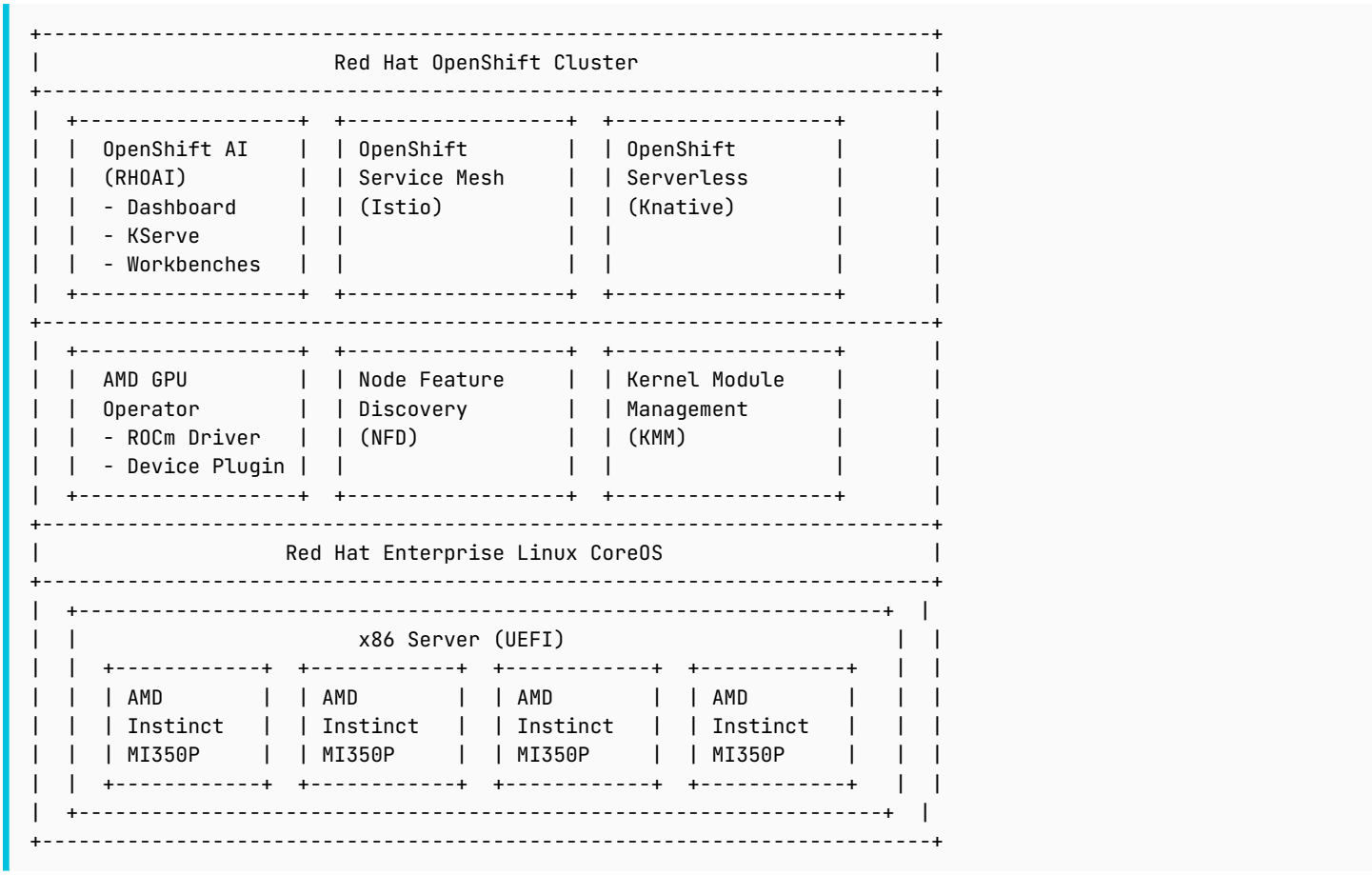
---

1. OVERVIEW

This guide provides detailed instructions for deploying a AI/ML infrastructure using:

- **Red Hat OpenShift Container Platform (OCP)** - Enterprise Kubernetes platform
- **Red Hat OpenShift AI (RHOAI)** - AI/ML platform for model training and inference
- **AMD Instinct GPUs** - High-performance accelerators for AI workloads
- **AMD Inference Microservice (AIM)** - Optimized LLM inference containers

ARCHITECTURE SUMMARY



DEPLOYMENT TOPOLOGY OPTIONS

TOPOLOGY	CONTROL PLANE	WORKERS	USE CASE
SINGLE-NODE OPENSHIFT (SNO)	1	0	Development, PoC, Edge
COMPACT CLUSTER	3	0	Small production
STANDARD CLUSTER	3	2+	Production with GPU workers

This guide focuses on **Single-Node OpenShift (SNO)** deployment, which is ideal for:

- Proof of concept and development
- Resource-constrained environments

- Edge deployments
- Quick validation of AI workloads

2. PREREQUISITES

2.1 HARDWARE REQUIREMENTS

Provisioning Host (Control Machine)

The provisioning host is where you run Ansible playbooks to orchestrate the deployment.

COMPONENT	MINIMUM	RECOMMENDED
CPU	4 cores	8 cores
RAM	8 GB	16 GB
DISK	100 GB	200 GB
OS	RHEL 9.x	RHEL 10.x
NETWORK	1 GbE	10 GbE

SNO Node (Single-Node OpenShift)

COMPONENT	MINIMUM	RECOMMENDED
CPU	8 cores	32+ cores
RAM	32 GB	256 GB+
ROOT DISK	120 GB NVMe	500 GB NVMe
DATA DISK	500 GB NVMe	1 TB+ NVMe
GPU	1x AMD Instinct	2-4x AMD Instinct
NETWORK	1 GbE	10/25 GbE
BMC	Redfish-compliant	Redfish-compliant

Supported AMD GPUs

GPU MODEL	ARCHITECTURE	GFX TARGET	MEMORY
AMD INSTINCT MI355X	CDNA 4	gfx950	288 GB HBM3e
AMD INSTINCT MI350P	CDNA 4	gfx950	144 GB HBM3e
AMD INSTINCT MI350X	CDNA 4	gfx950	288 GB HBM3e
AMD INSTINCT MI325X	CDNA 3	gfx942	256 GB HBM3e
AMD INSTINCT MI300X	CDNA 3	gfx942	192 GB HBM3

2.2 SOFTWARE REQUIREMENTS

Provisioning Host Software

SOFTWARE	VERSION	PURPOSE
RHEL	10.x	Host operating system

SOFTWARE	VERSION	PURPOSE
ANSIBLE CORE	2.16.18	Automation engine
PYTHON	3.12.13	Ansible runtime

Required Credentials

CREDENTIAL	SOURCE	PURPOSE
RED HAT PULL SECRET	cloud.redhat.com	Container image pulls
BMC CREDENTIALS	Server BMC admin	BMC access (iDRAC, iLO, etc.)
SSH KEY PAIR	Generated locally	Cluster node access
HUGGINGFACE TOKEN	huggingface.co	Model downloads
DOCKER HUB CREDENTIALS	docker.io	AIM image pulls

2.3 TESTED CONFIGURATION

This guide has been validated with the following hardware and software configuration.

Hardware Configuration

Provisioning Host:

COMPONENT	SPECIFICATION
CPU	1x AMD EPYC 9355P (32 cores, 64 threads)
RAM	754 GB
ROOT DISK	894 GB NVMe
DATA DISK	1.7 TB NVMe
OS	RHEL 10.3 Beta

SNO Node:

COMPONENT	SPECIFICATION
CPU	2x AMD EPYC 9275F (48 cores total, 96 threads)
RAM	2.2 TB
ROOT DISK	894 GB NVMe
DATA DISK	1.7 TB NVMe
GPU	2x AMD Instinct MI350P (Device ID: 75a8)
OS	Red Hat Enterprise Linux CoreOS 9.8
BMC	Redfish-compliant

Software Stack

COMPONENT	VERSION	NOTES
OPENSHIFT CONTAINER PLATFORM	4.22.0	Single Node OpenShift
RED HAT ENTERPRISE LINUX COREOS	9.8	Immutable OS
KUBERNETES	v1.35.5	Bundled with OCP
RED HAT OPENSHIFT AI	3.4.2	Self-managed
AMD GPU OPERATOR	Latest (alpha channel)	OLM (certified-operators)
AMD GPU DRIVER	6.19.4	ROCm amdgpu
VLLM	0.19.x	LLM inference engine
PYTORCH	2.10.0	Deep learning framework
KSERVE	1.17	Model serving
OPENSHIFT SERVICE MESH	3.4.0	Istio-based
OPENSHIFT SERVERLESS	1.37.1	Knative
NODE FEATURE DISCOVERY	4.22.0	Hardware detection
KERNEL MODULE MANAGEMENT	2.6.1	Driver lifecycle
CERT-MANAGER	v1.20.3	Certificate management

3. SETUP ENVIRONMENT

3.1 CHOOSE YOUR INSTALLATION METHOD

METHOD	BEST FOR	REQUIREMENTS
ANSIBLE PLAYBOOKS	Production, repeatable deployments	Ansible Core 2.16+
MANUAL CLI	Learning, custom deployments, troubleshooting	oc

3.2 INSTALL REQUIRED TOOLS

For Both Methods

```
# Install OpenShift CLI
curl -LO https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-linux.tar.gz
tar -xzf openshift-client-linux.tar.gz
sudo mv oc kubectl /usr/local/bin/

# Verify installation
oc version
```

For Ansible Playbooks (Recommended)

> **Note:** The automation repository is currently private. Contact your AMD representative to request access to the playbooks and automation tooling.

```
# Clone the automation repository
git clone https://github.com/AMD-DEAE-CEME/Redhat-OpenShiftAI-Integration.git
cd redhat-ai-automation

# Install Ansible
sudo dnf install -y ansible-core python3-pip

# Run provisioning host setup
ansible-playbook playbooks/setup_provisioning_host.yml
```

This playbook installs:

- **OpenShift CLI tools**: openshift-install, oc, kubectl
- **Python dependencies**: jinja2, jmespath

3.3 CONFIGURE VARIABLES (ANSIBLE METHOD ONLY)

If using Ansible playbooks, configure the required variables:

3.3.1 Configure secrets.yml

```
cp vars/secrets.yml.example vars/secrets.yml
```

Edit `vars/secrets.yml` and update:

- `pull_secret`: Get from Red Hat Console
  - Required to pull Red Hat operator images

- Paste the entire JSON content as a single-quoted string

Optional but Recommended: Encrypt the secrets file:

```
ansible-vault encrypt vars/secrets.yml
```

If encrypted, add `--ask-vault-pass` when running playbooks.

3.3.2 Configure cluster.yml

Edit `vars/cluster.yml` and update the following key variables:

Required changes:

- **`install_dir`**: Installation directory path (e.g., `/home/amd/ocp-install`)
- **`lvms_device_paths`**: Update to match your SNO node's available disk device (e.g., `/dev/nvme1n1`, `/dev/sdb`)
  - Find available disks: `oc debug node/<node-name> -- chroot /host lsblk`

Optional changes (only if different from defaults):

- **`sno_node_has_gpu`**: Set to `true` if SNO node has AMD Instinct GPU (default: `true`)
- **`model_cache_size`**: Size of AI model cache PVC (default: `500Gi`)

> **Note:** The `cluster.yml` file contains many more configuration options for cluster deployment (network, BMC, DNS, etc.). If you're using an existing cluster, you only need to configure the storage-related variables listed above.

3.3.3 Review versions.yml (Optional)

The `vars/versions.yml` file contains channel configurations for OLM-managed components:

- AMD GPU Operator: alpha channel (installs latest available version)
- Red Hat OpenShift AI: stable-3.4 channel
- AMD KMM Driver: 31.30

3.4 SET KUBECONFIG

Set the `KUBECONFIG` environment variable:

```
export KUBECONFIG=/path/to/your/kubeconfig
```

Verify cluster access:

```
oc get nodes
oc get co
```

4. VERIFY CLUSTER ACCESS

This guide assumes you have an operational Red Hat OpenShift cluster (Single-Node or multi-node) with:

- Cluster accessible via `oc` CLI
- Administrative access (kubeadmin or cluster-admin role)
- Network connectivity to cluster API and routes

> **Note:** If you don't have an existing OpenShift cluster and want to deploy one for **development/test purposes**, see Appendix F: OpenShift Cluster Deployment. For **production deployments**, consult Red Hat's official installation documentation and engage Red Hat support for architecture planning.

4.1 METHOD A: ANSIBLE PLAYBOOK VALIDATION

Run the cluster validation playbook for comprehensive health checks with generated reports:

```
# Set KUBECONFIG environment variable
export KUBECONFIG=/home/amd/ocp-install/auth/kubeconfig

# Run validation
ansible-playbook playbooks/validate_cluster.yml
```

What This Playbook Validates

CATEGORY	CHECKS PERFORMED
CLUSTER VERSION	OpenShift version, cluster ID, update status
NODE HEALTH	Node ready status, kubelet version, OS image, kernel version
CLUSTER OPERATORS	Total operators, available count, degraded count
CORE SERVICES	API server, etcd quorum, authentication status
NETWORKING	Network type (OVN), cluster CIDR, service CIDR, ingress domain
STORAGE	Default storage class, storage operator status
IMAGE REGISTRY	Registry status, management state
CONSOLE	Console availability, URL
MACHINECONFIG	MachineConfigPool status (updated/degraded)
BAREMETALHOST	Provisioning state, power state (for SNO)
AMD GPU	GPU nodes, GPU count, device plugin status, driver version

Generated Reports

REPORT	LOCATION	DESCRIPTION
JSON	`reports/cluster-status.json`	Machine-readable cluster state
MARKDOWN	`reports/cluster-status-<timestamp>.md`	Human-readable summary
HTML	`reports/cluster-status-<timestamp>.html`	Visual dashboard report

4.2 METHOD B: MANUAL CLI VALIDATION

Verify cluster health using direct `oc` commands:

```
# Set KUBECONFIG
export KUBECONFIG=/path/to/your/kubeconfig

# Verify nodes are ready
oc get nodes
# All nodes should show STATUS=Ready

# Check cluster version
oc get clusterversion
# Should show AVAILABLE=True

# Check all cluster operators
oc get co
# All operators should show AVAILABLE=True, PROGRESSING=False, DEGRADED=False

# Check for any degraded operators
oc get co -o json | jq -r '.items[] | select(.status.conditions[] | select(.type=="Degraded"
and .status=="True")) | .metadata.name'
# Should return empty (no degraded operators)

# Verify API server is responding
oc whoami
oc cluster-info

# Check etcd health
oc get etcd -o jsonpath='{.items[0].status.conditions[?(@.type=="EtcdMembersAvailable")].status}'
# Should return "True"

# Check MachineConfigPool status
oc get mcp
# Should show UPDATED=True, UPDATING=False, DEGRADED=False

# Check image registry status
oc get configs.imageregistry.operator.openshift.io/cluster -o jsonpath='{.spec.managementState}'
# Should return "Managed" or "Removed"

# Get console URL
oc whoami --show-console
```

4.3 HEALTH ASSERTIONS

Before proceeding, verify:

- All cluster operators are `Available=True`
- etcd is healthy with quorum
- All nodes are in `Ready` state
- MachineConfigPool is not updating or degraded

Note: GPU device plugin and driver status will show "Not Installed" until you complete Section 5 (Install AI/GPU Components).

If all checks pass, the cluster is ready for AI/GPU component installation.

5. INSTALL AI/GPU COMPONENTS

5.1 COMPONENT INSTALLATION ORDER

The playbook installs components in dependency order:

```
|   - KnativeServing                                                |
|   - Required for KServe CRDs                                      |
+-------------------------------------------------------------------+
                                |
                                v
+-------------------------------------------------------------------+
| Step 9: Red Hat OpenShift AI (RHOAI)                              |
|   - RHOAI Operator                                                |
|   - DataScienceCluster with KServe in RawDeployment mode          |
|   - GatewayConfig for model serving                               |
+-------------------------------------------------------------------+
```

5.2 METHOD A: ANSIBLE PLAYBOOK (RECOMMENDED)

Install All Components

```
ansible-playbook playbooks/install_ai_components.yml --ask-vault-pass
```

Install Individual Components

You can install components individually using tags:

```
# Step 1: cert-manager
ansible-playbook playbooks/install_ai_components.yml --tags cert-manager

# Step 2: Node Feature Discovery (NFD)
ansible-playbook playbooks/install_ai_components.yml --tags nfd

# Step 3: Kernel Module Management (KMM)
ansible-playbook playbooks/install_ai_components.yml --tags kmm

# Step 4: Image Registry
ansible-playbook playbooks/install_ai_components.yml --tags image-registry

# Step 5: amdgpu Kernel Module Blacklist
ansible-playbook playbooks/install_ai_components.yml --tags gpu-blacklist

# Step 6: AMD GPU Operator
ansible-playbook playbooks/install_ai_components.yml --tags gpu-operator

# Step 7: OpenShift Service Mesh
ansible-playbook playbooks/install_ai_components.yml --tags servicemesh

# Step 8: OpenShift Serverless
ansible-playbook playbooks/install_ai_components.yml --tags serverless

# Step 9: Red Hat OpenShift AI (RHOAI)
ansible-playbook playbooks/install_ai_components.yml --tags openshift-ai
```

5.3 METHOD B: MANUAL CLI INSTALLATION

For manual installation using direct `oc` commands, follow the steps below. Execute each step in order.

Step 1: Install cert-manager

Required by NFD, KMM, and AMD GPU Operator for TLS certificates.

Reference: Red Hat cert-manager Operator Documentation - Installing via CLI

```
# Create namespace
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: cert-manager-operator
EOF

# Create OperatorGroup (AllNamespaces mode - recommended for OpenShift v1.15.0+)
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cert-manager-operator
  namespace: cert-manager-operator
spec:
  targetNamespaces: []
  spec: {}
EOF

# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n cert-manager-operator \
  -l operators.coreos.com/openshift-cert-manager-operator.cert-manager-operator \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Wait for pods
oc wait pod -n cert-manager -l app=cert-manager --for=condition=Ready --timeout=300s
```

Step 2: Install Node Feature Discovery (NFD)

Per AMD OpenShift guide: Use Red Hat's NFD Operator for GPU detection.

```
# Create namespace with cluster monitoring
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-nfd
  labels:
    openshift.io/cluster-monitoring: "true"
```

```
  annotations:
    workload.openshift.io/allowed: management
EOF

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

# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: nfd
  namespace: openshift-nfd
spec:
  channel: stable
  installPlanApproval: Automatic
  name: nfd
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n openshift-nfd \
  -l operators.coreos.com/nfd.openshift-nfd \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=300s

# Create NFD instance with AMD GPU detection rules
cat <<EOF | oc apply -f -
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: amd-gpu-operator-nfd-instance
  namespace: openshift-nfd
spec:
  workerConfig:
    configData: |
      core:
        sleepInterval: 60s
      sources:
        pci:
          deviceClassWhitelist:
            - "0200"
            - "03"
            - "12"
          deviceLabelFields:
            - "vendor"
            - "device"
        custom:
        - name: amd-gpu
```

```
          labels:
            feature.node.kubernetes.io/amd-gpu: "true"
          matchAny:
            - matchFeatures:
                - feature: pci.device
                  matchExpressions:
                    vendor: {op: In, value: ["1002"]}
                    device: {op: In, value: [
                      "75a3", "75a8", "75a0", "74a5", "74a2",
                      "74a0", "74a1", "74a9", "740f", "7408",
                      "740c", "738c", "738e"
                    ]}
EOF
```

Step 3: Install Kernel Module Management (KMM)

```
# Create namespace
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-kmm
EOF

# Create OperatorGroup
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kernel-module-management
  namespace: openshift-kmm
EOF

# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: kernel-module-management
  namespace: openshift-kmm
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kernel-module-management
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n openshift-kmm \
  -l operators.coreos.com/kernel-module-management.openshift-kmm \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s
```

Step 4: Enable Image Registry

The image registry is required for KMM to store built GPU driver images. See Appendix E for details on how KMM uses the registry.

```
# Enable registry with emptyDir storage
oc patch configs.imageregistry.operator.openshift.io/cluster \
  --type merge \
  --patch '{"spec":{"managementState":"Managed","storage":{"emptyDir":{}}}}'

# Wait for registry pods
oc wait pod -n openshift-image-registry \
  -l docker-registry=default --for=condition=Ready --timeout=300s
```

Note: For production multi-node clusters, use persistent storage (PVC with ReadWriteMany) instead of `emptyDir`.

Step 5: Blacklist In-Kernel amdgpu Module

WARNING: This will cause the node to reboot automatically.

```
cat <<EOF | oc apply -f -
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: amdgpu-module-blacklist
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: "/etc/modprobe.d/amdgpu-blacklist.conf"
          mode: 420
          overwrite: true
          contents:
            source: "data:text/plain; base64,YmxhY2tsaXN0IGFtZGdwdQppbnN0YWxsIGFtZGdwdSAvYmluL2ZhbHNlCg=="
EOF

# Wait for node reboot and MCP update (5-10 minutes)
oc get mcp master -w
# Wait until UPDATED=True and UPDATING=False
```

Step 6: Install AMD GPU Operator via OLM

The AMD GPU Operator is available in the certified-operators catalog. This follows AMD's official OpenShift installation guide and provides:

- **Native OpenShift integration** - Uses OLM for lifecycle management
- **Automatic updates** - Operator updates managed by OLM
- **Red Hat certified** - Available in the certified-operators catalog
- **Simplified installation** - No manual Helm chart management required

Reference: https://instinct.docs.amd.com/projects/gpu-operator/en/latest/installation/openshift-olm.html

```
# Create namespace
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: amd-gpu-operator
  labels:
    openshift.io/cluster-monitoring: "true"
EOF

# Create OperatorGroup (AllNamespaces mode - required by AMD GPU Operator)
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: amd-gpu-operator-group
  namespace: amd-gpu-operator
spec: {}
EOF

# Subscribe to AMD GPU Operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amd-gpu-operator
  namespace: amd-gpu-operator
spec:
  channel: alpha
  installPlanApproval: Automatic
  name: amd-gpu-operator
  source: certified-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator CSV to be ready
oc wait csv -n amd-gpu-operator \
  -l operators.coreos.com/amd-gpu-operator.amd-gpu-operator \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Wait for DeviceConfig CRD to be available
oc get crd deviceconfigs.amd.com

# Grant required SCCs
oc adm policy add-scc-to-user privileged -z amd-gpu-operator-kmm-device-plugin -n amd-gpu-operator
oc adm policy add-scc-to-user privileged -z amd-gpu-operator-metrics-exporter -n amd-gpu-operator
oc adm policy add-scc-to-user privileged -z amd-gpu-operator-node-labeller -n amd-gpu-operator
oc adm policy add-scc-to-user privileged -z amd-gpu-operator-kmm-module-loader -n amd-gpu-operator

# Grant registry push access for KMM builds
oc policy add-role-to-user system:image-builder -z default -n amd-gpu-operator
oc policy add-role-to-user system:image-pusher -z default -n amd-gpu-operator
oc policy add-role-to-user system:image-builder -z builder -n amd-gpu-operator
oc policy add-role-to-user system:image-pusher -z builder -n amd-gpu-operator

# Create DeviceConfig
# Note: driver version "31.30" matches vars/versions.yml
cat <<EOF | oc apply -f -
apiVersion: amd.com/v1alpha1
kind: DeviceConfig
metadata:
  name: test-cr
  namespace: amd-gpu-operator
spec:
```

```
  driver:
    enable: true
    useSourceImage: true
    image: image-registry.openshift-image-registry.svc:5000/amd-gpu-operator/amdgpu_kmod
    version: "31.30"
  devicePlugin:
    enableDevicePlugin: true
    enableNodeLabeller: true
  metricsExporter:
    enable: true
    image: docker.io/rocm/device-metrics-exporter:v1.5.0-beta.0
    prometheus:
      serviceMonitor:
        enable: true
        interval: "60s"
        attachMetadata:
          node: true
  selector:
    "feature.node.kubernetes.io/amd-gpu": "true"
EOF
```

Note: The driver version `31.30` specified above corresponds to the AMD KMM driver version defined in `vars/versions.yml`. Ensure this matches the version in your configuration.

Step 7: Install OpenShift Service Mesh 3.x

```
# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: servicemeshoperator3
  namespace: openshift-operators
spec:
  channel: stable
  installPlanApproval: Automatic
  name: servicemeshoperator3
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n openshift-operators \
  -l operators.coreos.com/servicemeshoperator3.openshift-operators \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Create Istio control plane
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: istio-system
---
apiVersion: sailoperator.io/v1
kind: Istio
metadata:
  name: default
  namespace: istio-system
```

```
spec:
  version: v1.28-latest
  namespace: istio-system
  values:
    global:
      istioNamespace: istio-system
    pilot:
      autoscaleEnabled: false
      resources:
        requests:
          cpu: 100m
          memory: 512Mi
EOF

# Wait for Istio
oc wait istio default -n istio-system --for=condition=Ready --timeout=600s
```

Step 8: Install OpenShift Serverless

```
# Create namespace and OperatorGroup
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-serverless
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: serverless-operators
  namespace: openshift-serverless
spec: {}
EOF

# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: serverless-operator
  namespace: openshift-serverless
spec:
  channel: stable
  installPlanApproval: Automatic
  name: serverless-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n openshift-serverless \
  -l operators.coreos.com/serverless-operator.openshift-serverless \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Create KnativeServing
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: knative-serving
---
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  ingress:
    kourier:
      enabled: true
  config:
    network:
      ingress-class: kourier.ingress.networking.knative.dev
EOF

# Wait for KnativeServing
oc wait knativeserving knative-serving -n knative-serving --for=condition=Ready --timeout=300s
```

Step 9: Install Red Hat OpenShift AI

```
# Create namespace
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: redhat-ods-operator
  labels:
    openshift.io/cluster-monitoring: "true"
EOF

# Create OperatorGroup
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: rhods-operator-group
  namespace: redhat-ods-operator
spec: {}
EOF

# Subscribe to operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: rhods-operator
  namespace: redhat-ods-operator
spec:
  channel: stable-3.4
  installPlanApproval: Automatic
  name: rhods-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Wait for operator
oc wait csv -n redhat-ods-operator \
  -l operators.coreos.com/rhods-operator.redhat-ods-operator \
```

```
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Create DSCInitialization
cat <<EOF | oc apply -f -
apiVersion: dscinitialization.opendatahub.io/v1
kind: DSCInitialization
metadata:
  name: default-dsci
spec:
  applicationsNamespace: redhat-ods-applications
  monitoring:
    managementState: Managed
    namespace: redhat-ods-monitoring
  serviceMesh:
    managementState: Managed
    controlPlane:
      name: default
      namespace: istio-system
      metricsCollection: Istio
  trustedCABundle:
    managementState: Managed
EOF

sleep 30

# Create DataScienceCluster
cat <<EOF | oc apply -f -
apiVersion: datasciencecluster.opendatahub.io/v1
kind: DataScienceCluster
metadata:
  name: default-dsc
spec:
  components:
    dashboard:
      managementState: Managed
    workbenches:
      managementState: Removed
    modelmeshserving:
      managementState: Removed
    kserve:
      managementState: Managed
      defaultDeploymentMode: RawDeployment
      serving:
        managementState: Managed
        ingressGateway:
          certificate:
            type: OpenshiftDefaultIngress
        name: knative-serving
    datasciencepipelines:
      managementState: Removed
    codeflare:
      managementState: Removed
    ray:
      managementState: Removed
    trainingoperator:
      managementState: Removed
    modelregistry:
      managementState: Managed
EOF
```

```
# Verify KServe is ready
oc get datasciencecluster default-dsc \
  -o jsonpath='{.status.conditions[?(@.type=="KserveReady")].status}'
```

Quick Verification

After completing all 9 steps, verify the installation with these commands:

```
# Check all operators are ready
oc get csv -A | grep -E '(cert-manager|nfd|kmm|gpu|servicemesh|serverless|rhods)' | grep Succeeded

# Check GPU operator pods
oc get pods -n amd-gpu-operator

# Check Service Mesh
oc get istio -n istio-system

# Check Serverless
oc get knativeserving -n knative-serving

# Check RHOAI
oc get datasciencecluster default-dsc
```

If all commands show healthy status, proceed to Section 5.4 for detailed GPU verification and Section 5.5 for AI components validation.

If any component shows errors, check the troubleshooting section in Appendix A.

5.4 VERIFY GPU DETECTION

Run these verification commands regardless of which installation method you used.

```
# Check GPU node labels
oc get nodes -o yaml | grep amd.com/gpu
```

Example output:

```
      amd.com/gpu.cu-count: "128"
      amd.com/gpu.device-id: 75a8
      amd.com/gpu.driver-version: 6.19.4
      amd.com/gpu.family: AI
      amd.com/gpu.product-name: AMD_Radeon_Graphics
      amd.com/gpu.simd-count: "512"
      amd.com/gpu.vram: 144G
      beta.amd.com/gpu.cu-count: "128"
      beta.amd.com/gpu.cu-count.128: "2"
      beta.amd.com/gpu.device-id: 75a8
      beta.amd.com/gpu.device-id.75a8: "2"
      beta.amd.com/gpu.family: AI
      beta.amd.com/gpu.family.AI: "2"
      beta.amd.com/gpu.product-name: AMD_Radeon_Graphics
      beta.amd.com/gpu.product-name.AMD_Radeon_Graphics: "2"
      beta.amd.com/gpu.simd-count: "512"
      beta.amd.com/gpu.simd-count.512: "2"
      beta.amd.com/gpu.vram: 144G
      beta.amd.com/gpu.vram.144G: "2"
      amd.com/gpu: "2"
```

```
# Verify GPU operator pods
oc get pods -n amd-gpu-operator
```

Example output:

```
NAME                                                  READY   STATUS
amd-gpu-operator-gpu-operator-charts-
  controller-manager-6689dr2                          1/1     Running
amd-gpu-operator-workflow-controller-
  7fc5c449b-8v2vq                                     1/1     Running
test-cr-device-plugin-jfjmq                           1/1     Running
test-cr-metrics-exporter-46zh5                        1/1     Running
test-cr-node-labeller-zqfjc                           1/1     Running
```

```
# Check GPU resources on node
oc describe node sno-0.sno.ocp.local | grep -A 5 "Allocatable:"
```

Example output:

```
Allocatable:
  amd.com/gpu:        2
  cpu:                95500m
  ephemeral-storage:  862137112786
  hugepages-1Gi:      0
  hugepages-2Mi:      0
```

```
# Verify device plugin is running
oc get daemonset -n amd-gpu-operator
```

Example output:

```
NAME                    DESIRED CURRENT READY UP-TO-DATE AVAILABLE AGE
test-cr-device-plugin   1       1       1     1          1         3d5h
  NODE SELECTOR:
    feature.node.kubernetes.io/amd-gpu=true,
    kmm.node.kubernetes.io/amd-gpu-operator.test-cr.ready=

test-cr-metrics-exporter 1      1       1     1          1         3d5h
  NODE SELECTOR:
    feature.node.kubernetes.io/amd-gpu=true,
    kmm.node.kubernetes.io/amd-gpu-operator.test-cr.ready=

test-cr-node-labeller   1       1       1     1          1         3d5h
  NODE SELECTOR:
    feature.node.kubernetes.io/amd-gpu=true
```

5.5 VERIFY AI COMPONENTS

Method A: Ansible Playbook

```
ansible-playbook playbooks/validate_ai_components.yml --ask-vault-pass
```

Report generated: `reports/ai-components-<timestamp>.html`

Method B: Manual CLI

```
# Check all operator subscriptions
oc get csv -A | grep -E '(nfd|kmm|gpu|rhods|serverless|servicemesh)'

# Verify DataScienceCluster status
oc get datasciencecluster default-dsc -o jsonpath='{.status.conditions[*].type}' | tr ' ' '\n'
```

Example output:

```
Ready
ProvisioningSucceeded
ComponentsReady
AIPipelinesReady
DashboardReady
FeastOperatorReady
KserveLLMInferenceServiceDependencies
KserveLLMInferenceServiceWideEPDependencies
KserveReady
KueueReady
LlamaStackOperatorReady
MLflowOperatorReady
ModelControllerReady
ModelRegistryReady
ModelsAsServiceReady
RayReady
SparkOperatorReady
TrainerReady
TrainingOperatorReady
TrustyAIReady
```

```
# Check KnativeServing
oc get knativeserving -n knative-serving
```

Example output:

```
NAME              VERSION   READY   REASON
knative-serving   1.17      True
```

```
# Check Service Mesh
oc get istio -n istio-system
```

Example output:

```
NAME      NAMESPACE      PROFILE   REVISIONS   READY   IN USE   ACTIVE REVISION   STATUS    VERSION
AGE
default   istio-system             1           1       0        default           Healthy   v1.28-latest
3d4h
```

```
# Verify istiod is running
oc get pods -n istio-system
```

Example output:

```
NAME                      READY   STATUS    RESTARTS   AGE
istiod-68b5b659b5-4gglh   1/1     Running   0          3d4h
```

6. CONFIGURE RED HAT OPENSHIFT AI FOR MODEL DEPLOYMENT

6.1 OVERVIEW

This section configures the following components:

STEP	COMPONENT	DESCRIPTION
1	**ai-workloads namespace**	Project for model deployments
2	**anyuid SCC**	Security context for AIM containers
3	**AMD Instinct Hardware Profile**	GPU accelerator profile in RHOAI dashboard
4	**AMD AIM Serving Runtime Templates**	Pre-configured model runtimes for vLLM

6.2 METHOD A: ANSIBLE PLAYBOOK

```
ansible-playbook playbooks/setup_openshift_ai.yaml --ask-vault-pass
```

This playbook creates the namespace, grants SCC, creates hardware profile, and applies all serving runtime templates from `cluster-config/applications/ai-workloads/serving-templates/`.

To apply templates separately:

```
oc apply -f cluster-config/applications/ai-workloads/serving-templates/
```

6.3 METHOD B: MANUAL CLI

```
# Step 1: Create ai-workloads namespace
oc create namespace ai-workloads

# Step 2: Grant anyuid SCC to default service account
oc adm policy add-scc-to-user anyuid -z default -n ai-workloads

# Step 3: Create AMD Instinct Hardware Profile
cat <<EOF | oc apply -f -
apiVersion: dashboard.opendatahub.io/v1
kind: HardwareProfile
metadata:
  name: amd-instinct-gpu
  namespace: redhat-ods-applications
spec:
  displayName: AMD Instinct GPU
  description: Hardware profile for AMD Instinct MI300X/MI325X/MI350 GPUs
  enabled: true
  identifier: amd.com/gpu
EOF

# Step 4: Create AMD AIM Serving Runtime Template (example: GPT-OSS-20B)
cat <<'EOF' | oc apply -f -
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: amd-aims-openai-gpt-oss-20b
  namespace: redhat-ods-applications
  labels:
```

```
    opendatahub.io/dashboard: "true"
  annotations:
    openshift.io/display-name: "AMD AIM OpenAI GPT-OSS 20B Runtime"
    opendatahub.io/modelServingSupport: '["single"]'
    opendatahub.io/apiProtocol: "REST"
    opendatahub.io/recommended-accelerators: '["amd.com/gpu"]'
objects:
  - apiVersion: serving.kserve.io/v1alpha1
    kind: ServingRuntime
    metadata:
      name: amd-aims-openai-gpt-oss-20b-runtime
      labels:
        opendatahub.io/dashboard: "true"
      annotations:
        opendatahub.io/template-display-name: "AMD AIM OpenAI GPT-OSS 20B Runtime"
        opendatahub.io/template-name: "amd-aims-openai-gpt-oss-20b"
        opendatahub.io/recommended-accelerators: '["amd.com/gpu"]'
    spec:
      protocolVersions:
        - v1
      supportedModelFormats:
        - name: vLLM
          version: "1"
          autoSelect: true
        - name: aim
          version: "1"
          autoSelect: true
      imagePullSecrets:
        - name: silogen-pull-secret
      volumes:
        - name: model-cache
          persistentVolumeClaim:
            claimName: ai-model-cache
      containers:
        - name: kserve-container
          image: docker.io/silogenai/aim-instinct-openai-gpt-oss-20b:0.12.0-rc7
          imagePullPolicy: IfNotPresent
          securityContext:
            runAsUser: 0
            runAsGroup: 0
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
          env:
            - name: PYTORCH_ROCM_ARCH
              value: gfx950
            - name: HOME
              value: /workspace/model-cache/
            - name: AIM_ALLOW_UNOPTIMIZED
              value: "true"
            - name: HF_HOME
              value: /workspace/model-cache/
            - name: HF_DATASETS_CACHE
              value: /workspace/model-cache/
            - name: AITER_JIT_DIR
              value: /workspace/model-cache/.aiter
            - name: VLLM_CACHE_ROOT
              value: /workspace/model-cache/vllm-cache
            - name: HF_TOKEN
```

```
            valueFrom:
              secretKeyRef:
                name: hf-token
                key: token
        volumeMounts:
          - name: model-cache
            mountPath: /workspace/model-cache
        startupProbe:
          httpGet:
            path: /health
            port: 8000
          periodSeconds: 20
          failureThreshold: 180
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          periodSeconds: 10
          failureThreshold: 3
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          periodSeconds: 30
          failureThreshold: 3
        ports:
          - containerPort: 8000
            name: http
            protocol: TCP
        resources:
          requests:
            cpu: "8"
            memory: 32Gi
            amd.com/gpu: "1"
          limits:
            cpu: "32"
            memory: 96Gi
            amd.com/gpu: "1"
EOF
```

Available AIM templates (in `cluster-config/applications/ai-workloads/serving-templates/`):

TEMPLATE	MODEL	IMAGE
AMD-AIM-OPENAI-GPT-OSS-20B	OpenAI GPT-OSS 20B	silogenai/aim-instinct-openai-gpt-oss-20b
AMD-AIM-OPENAI-GPT-OSS-120B	OpenAI GPT-OSS 120B	silogenai/aim-instinct-openai-gpt-oss-120b
AMD-AIM-META-LLAMA-3-3-70B	Meta Llama 3.3 70B	silogenai/aim-instinct-meta-llama-3.3-70b
AMD-AIM-GOOGLE-GEMMA-3-27B	Google Gemma 3 27B	silogenai/aim-instinct-google-gemma-3-27b
AMD-AIM-GOOGLE-GEMMA-4-26B	Google Gemma 4 26B	silogenai/aim-instinct-google-gemma-4-26b
AMD-AIM-GOOGLE-GEMMA-4-31B	Google Gemma 4 31B	silogenai/aim-instinct-google-gemma-4-31b
AMD-AIM-MISTRAL-SMALL-3-2-24B	Mistral Small 3.2 24B	silogenai/aim-instinct-mistral-small-3.2-24b

6.4 STORAGE FOR MODEL CACHE

AIM containers download model weights on first boot. Persistent storage is required to cache these models across pod restarts.

Storage Options:

OPTION	BEST FOR	ACCESS MODE	PERFORMANCE
LVMS	SNO deployments	RWO	High (local NVMe/SSD)
NFS	Multi-node or existing NFS infrastructure	RWX	Medium (network)

Option 1: LVMS Storage (Recommended for SNO)

Use this option for Single-Node OpenShift deployments.

LVMS provides in-cluster storage using local disks with high performance and zero external dependencies.

What this creates:

- LVMS Operator in `openshift-storage` namespace
- LVMCluster using your available disk (e.g., `/dev/nvme1n1`)
- StorageClass (`lvms-vg1`)
- PersistentVolumeClaim (`ai-model-cache`) in `ai-workloads` namespace

Prerequisites:

- Available disk device on SNO node (unmounted, not in use)
- Disk must not be partitioned or in use

Find available disks:

```
oc debug node/<node-name> -- chroot /host lsblk
```

Advantages:

- 2-3x faster model loading (local NVMe vs network NFS)
- No external dependencies
- In-cluster management via operator
- RWO sufficient for SNO (all pods on same node)

Limitations:

- Only works for Single-Node OpenShift
- Cannot share across multiple nodes (RWO only)

Method A: Ansible Playbook

Update device paths in `vars/cluster.yml`:

```
ai_model_cache_backend: lvms
lvms_device_paths:
  - "/dev/nvme1n1"  # CHANGE: Your data disk
```

Run the playbook:

```
ansible-playbook playbooks/setup_lvms_storage.yml --ask-vault-pass
```

Method B: Manual CLI

```
# Step 1: Create openshift-storage namespace
cat <<EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-storage
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
EOF

# Step 2: Create OperatorGroup for LVMS
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-storage-operatorgroup
  namespace: openshift-storage
spec:
  targetNamespaces:
    - openshift-storage
EOF

# Step 3: Subscribe to LVMS Operator
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: lvms-operator
  namespace: openshift-storage
spec:
  channel: stable-4.17
  installPlanApproval: Automatic
  name: lvms-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

# Step 4: Wait for LVMS Operator to be ready
oc wait csv -n openshift-storage \
  -l operators.coreos.com/lvms-operator.openshift-storage \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=600s

# Step 5: Create LVMCluster
# IMPORTANT: Update device paths to match your available disks
cat <<EOF | oc apply -f -
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: lvms-cluster
  namespace: openshift-storage
spec:
```

```
  storage:
    deviceClasses:
      - name: vg1
        default: true
        thinPoolConfig:
          name: thin-pool-1
          sizePercent: 90
          overprovisionRatio: 10
        deviceSelector:
          paths:
            - /dev/nvme1n1  # CHANGE: Your data disk path
EOF

# Step 6: Wait for LVMCluster to be ready
oc get lvmcluster lvms-cluster -n openshift-storage -w
# Wait until STATUS shows "Ready" (Ctrl+C to exit watch)

# Step 7: Verify StorageClass was created
oc get storageclass lvms-vg1

# Step 8: Create ai-workloads namespace (if not exists)
oc create namespace ai-workloads --dry-run=client -o yaml | oc apply -f -

# Step 9: Create PersistentVolumeClaim for AI model cache
cat <<EOF | oc apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ai-model-cache
  namespace: ai-workloads
  labels:
    purpose: ai-models
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: lvms-vg1
  resources:
    requests:
      storage: 500Gi
EOF

# Step 10: Verify PVC (will be Pending until first pod uses it)
oc get pvc ai-model-cache -n ai-workloads
```

> **Note:** The PVC will remain in `Pending` state with LVMS's `WaitForFirstConsumer` binding mode. It will automatically bind when the first pod (model deployment) uses it.

Option 2: NFS Storage (Multi-node clusters)

Use this option if you need multi-node support or prefer external storage.

```
ansible-playbook playbooks/setup_nfs_on_host.yml --ask-vault-pass
```

This creates:

- NFS server on provisioning host at `/home/amd/srv/nfs`

- PersistentVolume (`ai-model-cache-pv`)
- PersistentVolumeClaim (`ai-model-cache`) in `ai-workloads` namespace

Advantages:

- Works for both SNO and multi-node clusters (RWX)
- Data persists independently of cluster

Limitations:

- External dependency on NFS server
- Network overhead (slower than local storage)

Verify Storage

```
# For LVMS
oc get storageclass lvms-vg1
oc get lvmcluster -n openshift-storage
oc get pvc -n ai-workloads

# For NFS
oc get pv ai-model-cache-pv
oc get pvc -n ai-workloads

# PVC should show: STATUS=Bound
```

6.5 CREATE REQUIRED SECRETS

HuggingFace Token (for gated models)

```
oc create secret generic hf-token \
  --from-literal=token=<your-huggingface-token> \
  -n ai-workloads
```

Docker Hub Pull Secret (for AIM images)

```
oc create secret docker-registry silogen-pull-secret \
  --docker-server=docker.io \
  --docker-username=<your-dockerhub-username> \
  --docker-password=<your-dockerhub-password> \
  -n ai-workloads
```

6.6 DEPLOY MODELS VIA RHOAI DASHBOARD

Access the Dashboard

```
# Get dashboard URL
oc get route rhods-dashboard -n redhat-ods-applications \
  -o jsonpath='{.spec.host}'
# Open in browser:
# https://rhods-dashboard-redhat-ods-applications.apps.sno.ocp.local
```

> **Note:** If using `/etc/hosts` for DNS (see Appendix C), add the dashboard URL to your hosts file:
>
> ```
> echo "<cluster-ip> \
>   rhods-dashboard-redhat-ods-applications.apps.<cluster-domain>" \
>   >> /etc/hosts
> ```

Deploy a Model

1. Navigate to **AI hub** -> **Models** -> **Deploy model**
2. **Step 1 - Model details:**
   - **Model location:** Select `URI`
   - **URI:** `pvc://ai-model-cache`
   - Check **Create a connection to this location**
   - **Name:** `gpt-oss-20b`
   - **Model type:** `Generative AI model (Example, LLM)`
   - Click **Next**
3. **Step 2 - Model deployment:**
   - **Project:** Select `ai-workloads`
   - **Model deployment name:** `gpt-oss-20b`
   - **Hardware profile:** Select `AMD Instinct GPU`
   - Expand **Customize resource requests:**
     - **CPU:** `12` Cores
     - **Memory:** `128` GiB
     - **AMD GPU:** `1`
   - **Deployment resource:** Select **Manual selection**
     - Choose `amd-aims-openai-gpt-oss-20b-runtime`
   - **Number of replicas:** `1`
   - Click **Next**
4. **Step 3 - Advanced settings:**
   - Check **Make model deployment available through an external route**
   - **Token authentication:** Uncheck (for testing)
   - Click **Next**
5. **Step 4 - Review:** Verify settings and click **Deploy**

6.7 DEPLOY MODELS VIA CLI

Pre-configured deployment files are available in `cluster-config/applications/ai-workloads/cli-model-deployments/`. Each file contains the ServingRuntime and InferenceService for direct deployment.

Available Models

MODEL	FILE	SIZE	PRECISION
OPENAI GPT-OSS 20B	`openai-gpt-oss-20b.yaml`	20B	mxfp4

MODEL	FILE	SIZE	PRECISION
OPENAI GPT-OSS 120B	openai-gpt-oss-120b.yaml	120B	mxfp4
META LLAMA 3.3 70B	meta-llama-3-3-70b.yaml	70B	fp8
MISTRAL SMALL 3.2 24B	mistral-small-3-2-24b.yaml	24B	fp16
GOOGLE GEMMA 3 27B	google-gemma-3-27b.yaml	27B	bf16

Method A: Deploy from Repo Files

```
# Deploy OpenAI GPT-OSS 20B
oc apply -f cluster-config/applications/ai-workloads/cli-model-deployments/openai-gpt-oss-20b.yaml
```

Method B: Deploy Inline (Example: GPT-OSS-20B)

```
cat <<'EOF' | oc apply -f -
---
apiVersion: serving.kserve.io/v1alpha1
kind: ServingRuntime
metadata:
  name: amd-aims-openai-gpt-oss-20b-runtime
  namespace: ai-workloads
  labels:
    opendatahub.io/dashboard: "true"
  annotations:
    opendatahub.io/template-display-name: "AMD AIM OpenAI GPT-OSS 20B Runtime"
    opendatahub.io/recommended-accelerators: '["amd.com/gpu"]'
spec:
  protocolVersions:
    - v1
  supportedModelFormats:
    - name: vLLM
      version: "1"
      autoSelect: true
    - name: aim
      version: "1"
      autoSelect: true
  imagePullSecrets:
    - name: silogen-pull-secret
  volumes:
    - name: model-cache
      persistentVolumeClaim:
        claimName: ai-model-cache
  containers:
    - name: kserve-container
      image: docker.io/silogenai/aim-instinct-openai-gpt-oss-20b:0.12.0-rc7
      imagePullPolicy: IfNotPresent
      securityContext:
        runAsUser: 0
        runAsGroup: 0
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      env:
        - name: PYTORCH_ROCM_ARCH
          value: gfx950
```

```
        - name: HOME
          value: /workspace/model-cache/
        - name: AIM_ALLOW_UNOPTIMIZED
          value: "true"
        - name: HF_HOME
          value: /workspace/model-cache/
        - name: HF_DATASETS_CACHE
          value: /workspace/model-cache/
        - name: VLLM_CACHE_ROOT
          value: /workspace/model-cache/vllm-cache
        - name: AITER_JIT_DIR
          value: /workspace/model-cache/.aiter
        - name: HF_TOKEN
          valueFrom:
            secretKeyRef:
              name: hf-token
              key: token
      volumeMounts:
        - name: model-cache
          mountPath: /workspace/model-cache
      startupProbe:
        httpGet:
          path: /health
          port: 8000
        periodSeconds: 20
        failureThreshold: 180
      readinessProbe:
        httpGet:
          path: /health
          port: 8000
        periodSeconds: 10
        failureThreshold: 3
      livenessProbe:
        httpGet:
          path: /health
          port: 8000
        periodSeconds: 30
        failureThreshold: 3
      ports:
        - containerPort: 8000
          name: http
          protocol: TCP
      resources:
        requests:
          cpu: "12"
          memory: 256Gi
          amd.com/gpu: "1"
        limits:
          cpu: "32"
          memory: 256Gi
          amd.com/gpu: "1"
---
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: openai-gpt-oss-20b
  namespace: ai-workloads
  labels:
    opendatahub.io/dashboard: "true"
```

```
    networking.kserve.io/visibility: exposed
  annotations:
    serving.kserve.io/deploymentMode: RawDeployment
    serving.knative.openshift.io/enablePassthrough: "true"
    sidecar.istio.io/inject: "false"
    openshift.io/display-name: "OpenAI GPT-OSS 20B"
spec:
  predictor:
    minReplicas: 1
    maxReplicas: 1
    model:
      modelFormat:
        name: vLLM
      runtime: amd-aims-openai-gpt-oss-20b-runtime
      storageUri: pvc://ai-model-cache
      resources:
        requests:
          cpu: "12"
          memory: 256Gi
          amd.com/gpu: "1"
        limits:
          cpu: "32"
          memory: 256Gi
          amd.com/gpu: "1"
EOF
```

Delete a Deployment

```
# Delete by file
oc delete -f cluster-config/applications/ai-workloads/cli-model-deployments/openai-gpt-oss-20b.yaml

# Or delete by resource name
oc delete inferenceservice openai-gpt-oss-20b -n ai-workloads
oc delete servingruntime amd-aims-openai-gpt-oss-20b-runtime -n ai-workloads
```

6.8 MONITOR DEPLOYMENT

```
# Watch InferenceService status
oc get isvc -n ai-workloads -w

# Check predictor pods
oc get pods -n ai-workloads

# View logs (model loading can take 25-30 minutes on first boot)
oc logs -n ai-workloads -l serving.kserve.io/inferenceservice=gpt-oss-20b -f
```

First Boot Timeline

PHASE	DURATION	ACTIVITY
POD SCHEDULING	1-2 min	GPU resources allocated
MODEL DOWNLOAD	10-20 min	Weights downloaded from HuggingFace
JIT COMPILATION	5-10 min	AMD AITER/vLLM kernel compilation
READY	-	Model serving requests

Subsequent restarts: 3-5 minutes (cached weights and kernels)

6.9 TEST MODEL ENDPOINT

```
# Get route URL
export MODEL_URL=$(oc get route gpt-oss-20b -n ai-workloads -o jsonpath='{.spec.host}')

# List available models
curl -k https://${MODEL_URL}/v1/models

# Chat completion
curl -k https://${MODEL_URL}/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "openai/gpt-oss-20b",
    "messages": [{"role": "user", "content": "Hello, how are you?"}],
    "max_tokens": 100
  }'
```

6.10 VALIDATE AIM DEPLOYMENTS

```
ansible-playbook playbooks/validate_aim_deployment.yml --ask-vault-pass
```

Report generated: `reports/aim-deployment-<timestamp>.html`

6.11 GENERATE CONSOLIDATED REPORT

Run all validations and generate a consolidated stack report:

```
ansible-playbook playbooks/validate_stack.yml --ask-vault-pass
```

This runs all three validation playbooks (cluster, AI components, AIM deployments) and generates:

- Individual JSON reports in `reports/`
- Consolidated HTML report: `reports/stack-validation-<timestamp>.html`
- Consolidated JSON report: `reports/stack-validation-<timestamp>.json`

The consolidated report includes:

- Overall stack health status
- Cluster validation results
- AI component versions
- AIM deployment status

APPENDIX

A. TROUBLESHOOTING

LVMS Storage Issues

LVMCluster stuck in "Progressing" state

Symptom:

```
Warning  ResourceReconciliationIncomplete  LVMClusterReconciler  csi node does not have driver topolvm.io
```

Diagnose:

```
# Check vg-manager pod status
oc get pods -n openshift-storage

# Check vg-manager logs for errors
oc logs -n openshift-storage -l app.kubernetes.io/name=vg-manager --tail=50

# Check LVMCluster status
oc describe lvmcluster -n openshift-storage
```

"Cannot use /dev/nvme1n1: device is partitioned"

Cause: The disk has GPT partition table signatures even if no partitions exist.

Diagnose:

```
# Check for existing signatures
oc debug node/<node-name> -- chroot /host wipefs /dev/nvme1n1
```

Fix:

```
# Wipe all signatures from the disk (WARNING: destroys all data)
oc debug node/<node-name> -- chroot /host wipefs -a /dev/nvme1n1

# Delete and recreate LVMCluster
oc delete lvmcluster lvms-cluster -n openshift-storage
sleep 10
ansible-playbook playbooks/setup_lvms_storage.yml --ask-vault-pass
```

LVMS PVC stuck in Pending

Diagnose:

```
# Check PVC status
oc describe pvc ai-model-cache -n ai-workloads

# Check StorageClass exists
oc get storageclass | grep lvms

# Check LVMCluster is Ready
oc get lvmcluster -n openshift-storage
```

Common causes:

- LVMCluster not Ready → wait or check vg-manager logs
- StorageClass not created → LVMCluster needs to be Ready first
- Disk not available → verify device path in `vars/cluster.yml`

Find available disks on SNO node

```
# List all block devices
oc debug node/<node-name> -- chroot /host lsblk -d -o NAME,SIZE,TYPE,FSTYPE,MOUNTPOINT

# Check for existing LVM
oc debug node/<node-name> -- chroot /host pvs
oc debug node/<node-name> -- chroot /host vgs
```

GPU Not Detected

```
# Check NFD detected GPU
oc get nodes -o json | jq '.items[].metadata.labels | select(. | keys[] | contains("amd.com"))'

# Check GPU operator logs
oc logs -n amd-gpu-operator deployment/amd-gpu-operator-controller-manager

# Check GPU operator CSV status
oc get csv -n amd-gpu-operator

# Verify MachineConfig applied
oc get mc | grep amdgpu-blacklist
oc get mcp master
```

Model Deployment Stuck

```
# Check InferenceService status
oc describe isvc <model-name> -n ai-workloads

# Check predictor pod events
oc describe pod -n ai-workloads -l serving.kserve.io/inferenceservice=<model-name>

# Check if GPU is available
oc describe node | grep -A 5 "Allocated resources"

# Check PVC is bound (models need storage)
oc get pvc -n ai-workloads
```

B. TEARDOWN

Remove Specific Model Deployments

```
# Delete a specific model deployment
oc delete -f \
  cluster-config/applications/ai-workloads/cli-model-deployments/mistral-small-3-2-24b.yaml

# Or delete by name
oc delete inferenceservice mistral-small-3-2-24b -n ai-workloads
oc delete servingruntime amd-aims-mistral-small-3-2-24b-runtime \
  -n ai-workloads
oc delete route mistral-small-3-2-24b -n ai-workloads
```

Remove OpenShift AI Workload Configuration

```
ansible-playbook playbooks/cleanup_openshift_ai.yaml --ask-vault-pass
```

This removes:

- Hardware Profile
- Serving Runtime Templates
- ai-workloads namespace
- anyuid SCC bindings

Remove LVMS Storage

```
# Remove LVMS PVC, LVMCluster, and operator
ansible-playbook playbooks/teardown_lvms_storage.yml --ask-vault-pass

# Also delete ai-workloads namespace
ansible-playbook playbooks/teardown_lvms_storage.yml --ask-vault-pass -e delete_ai_namespace=true
```

Remove NFS Storage

```
# Remove OpenShift resources only (keep NFS server and data)
ansible-playbook playbooks/teardown_nfs_on_host.yml --ask-vault-pass

# Remove everything including NFS server and data
ansible-playbook playbooks/teardown_nfs_on_host.yml --ask-vault-pass -e remove_nfs_server=true
```

Remove AI Components (GPU Operator, OpenShift AI, etc.)

> **Important:** Before running the teardown playbook, manually delete all InferenceServices first. The AMD GPU kernel module (`amdgpu`) cannot be unloaded while pods are using the GPU, which will cause the DeviceConfig finalizer to hang indefinitely.
>
> ```
> # Delete all InferenceServices first
> oc delete inferenceservice --all -n ai-workloads
>
> # Wait for pods to terminate (releases GPU)
> oc wait --for=delete pod -l serving.kserve.io/inferenceservice -n ai-workloads --timeout=300s
>
> # Verify no pods are using GPUs
> oc get pods -n ai-workloads
> ```

```
ansible-playbook playbooks/teardown_ai_components.yml --ask-vault-pass
```

This removes (in dependency order):

- OpenShift AI (RHOAI) with KServe
- OpenShift Serverless
- OpenShift Service Mesh
- AMD GPU Operator (OLM subscription)
- Kernel Module Management (KMM)
- cert-manager
- Node Feature Discovery (NFD)
- amdgpu kernel module blacklist (triggers node reboot)

Destroy Cluster

```
ansible-playbook playbooks/teardown_cluster.yml --ask-vault-pass
```

This removes the entire OpenShift cluster and reclaims the node.

C. DNS CONFIGURATION FOR ROUTES

OpenShift routes require DNS resolution to access services like the OpenShift console, RHOAI dashboard, and model endpoints. There are two methods to configure DNS:

Method 1: /etc/hosts (Recommended for Testing)

The simplest approach is to add entries to your local `/etc/hosts` file. This is ideal for development and testing environments.

Add the following entries to `/etc/hosts` on your client machine (replace IP with your SNO node IP):

```
# OpenShift Core Routes
10.216.177.95  console-openshift-console.apps.sno.ocp.local
10.216.177.95  oauth-openshift.apps.sno.ocp.local

# OpenShift AI Routes
10.216.177.95 rhods-dashboard-redhat-ods-applications.apps.sno.ocp.local
10.216.177.95  model-catalog.apps.sno.ocp.local
10.216.177.95  rh-ai.apps.sno.ocp.local

# Model Endpoints (add as you deploy models)
10.216.177.95  <model-name>—ai-workloads.apps.sno.ocp.local
```

To find routes that need DNS entries:

```
oc get routes -A -o jsonpath='{range .items[*]}{.spec.host}{"\n"}{end}'
```

Pros: Simple, no additional setup required **Cons:** Must manually add each new route

Method 2: DNS Server with Wildcard (Production)

For production environments or frequent model deployments, set up a DNS server with wildcard support. This eliminates the need to manually add entries for each route.

Run the infrastructure setup playbook on the provisioning host:

```
ansible-playbook playbooks/setup_infra.yml
```

This configures BIND DNS with:

- `api.sno.ocp.local` → SNO node IP
- `api-int.sno.ocp.local` → SNO node IP
- `*.apps.sno.ocp.local` → SNO node IP (wildcard - all routes work automatically)

Then configure your client to use the provisioning host as DNS server, or add a forwarder in your corporate DNS.

Pros: All routes work automatically, no manual updates **Cons:** Requires DNS server setup

D. AMD GPU METRICS

The AMD GPU Operator includes a Device Metrics Exporter that exports GPU telemetry to Prometheus. These metrics are available in the OpenShift monitoring stack.

Enable Metrics Exporter

Metrics are enabled in the DeviceConfig CR (cluster-config/operators/amd-gpu-operator/deviceconfig.yaml):

```
spec:
  metricsExporter:
    enable: true
    image: docker.io/rocm/device-metrics-exporter:v1.5.0-beta.0
    prometheus:
      serviceMonitor:
        enable: true
        interval: "60s"
        attachMetadata:
          node: true
```

Verify the namespace is labeled for cluster monitoring:

```
oc get namespace amd-gpu-operator -o jsonpath='{.metadata.labels.openshift\.io/cluster-monitoring}'
# Should output: true
```

Note: The namespace was created with this label in Step 6 of the installation. If the output is empty, add the label manually:

```
oc label namespace amd-gpu-operator openshift.io/cluster-monitoring=true
```

View Metrics in OpenShift Console

1. Navigate to **Observe → Metrics**
2. Enter a PromQL query (e.g., `gpu_used_vram`)
3. Metrics appear once the ServiceMonitor target is healthy

Key GPU Metrics

METRIC	DESCRIPTION
POWER & TEMPERATURE	
GPU_PACKAGE_POWER	Current GPU package power (W)
GPU_AVERAGE_PACKAGE_POWER	Average GPU package power (W)
GPU_EDGE_TEMPERATURE	GPU edge temperature (°C)
GPU_JUNCTION_TEMPERATURE	GPU junction/hotspot temperature (°C)
GPU_MEMORY_TEMPERATURE	GPU memory temperature (°C)
GPU_HBM_TEMPERATURE	HBM memory temperature (°C)
UTILIZATION	
GPU_GFX_ACTIVITY	Graphics engine utilization (%)
GPU_UMC_ACTIVITY	Unified Memory Controller activity (%)
GPU_MMA_ACTIVITY	Matrix Math Accelerator activity (%)
MEMORY	
GPU_TOTAL_VRAM	Total VRAM capacity (bytes)
GPU_USED_VRAM	Used VRAM (bytes)

METRIC	DESCRIPTION
GPU_FREE_VRAM	Free VRAM (bytes)
GPU_VRAM_MAX_BANDWIDTH	Maximum VRAM bandwidth (GB/s)
CLOCK & VOLTAGE	
GPU_CLOCK	Current GPU clock frequency (MHz)
GPU_MIN_CLOCK	Minimum GPU clock frequency (MHz)
GPU_MAX_CLOCK	Maximum GPU clock frequency (MHz)
GPU_VOLTAGE	Current GPU voltage (mV)
PCIE	
PCIE_BANDWIDTH	Current PCIe bandwidth utilization
PCIE_SPEED	Current PCIe link speed
PCIE_RX	PCIe receive throughput
PCIE_TX	PCIe transmit throughput
ECC ERRORS	
GPU_ECC_CORRECT_*	Correctable ECC errors by component
GPU_ECC_UNCORRECT_*	Uncorrectable ECC errors by component
HEALTH	
GPU_HEALTH	Overall GPU health status
GPU_ENERGY_CONSUMED	Total energy consumed (J)

Example PromQL Queries

```
# GPU memory utilization percentage
(gpu_used_vram / gpu_total_vram) * 100

# Average GPU temperature across all GPUs
avg(gpu_junction_temperature)

# Total power consumption
sum(gpu_package_power)

# GPUs with high temperature (>80°C)
gpu_junction_temperature > 80
```

For the complete list of available metrics, see the AMD Device Metrics Exporter documentation.

E. KERNEL MODULE MANAGEMENT (KMM) AND AMD GPU DRIVER

The AMD GPU Operator uses Kernel Module Management (KMM) to build, sign, and load the `amdgpu` kernel module on OpenShift nodes. This section explains how the driver is loaded and where it comes from.

How KMM Works

KMM is an OpenShift operator that manages out-of-tree kernel modules. For AMD GPUs, it:

1. **Builds the driver** from a source container image matching the CoreOS kernel version
2. **Stores the built image** in OpenShift's internal registry
3. **Loads the module** on nodes that match the selector criteria
4. **Manages lifecycle** including upgrades and node reboots

Driver Source and Build Process

The AMD GPU Operator configures KMM via the DeviceConfig CR (cluster-config/operators/amd-gpu-operator/deviceconfig.yaml):

```
spec:
  driver:
    enable: true
    useSourceImage: true
    image: image-registry.openshift-image-registry.svc:5000/amd-gpu-operator/amdgpu_kmod
    version: "31.30"
```

FIELD	DESCRIPTION
`USESOURCEIMAGE: TRUE`	Build driver from AMD's source container rather than using pre-built binaries
`IMAGE`	Destination in OpenShift's internal registry where the built driver is stored
`VERSION`	AMD GPU driver version (31.30 corresponds to ROCm 7.1+). Configured in `vars/versions.yml`

Driver Source Container

The AMD GPU Operator **automatically detects** the CoreOS version on the node and constructs the source image URL:

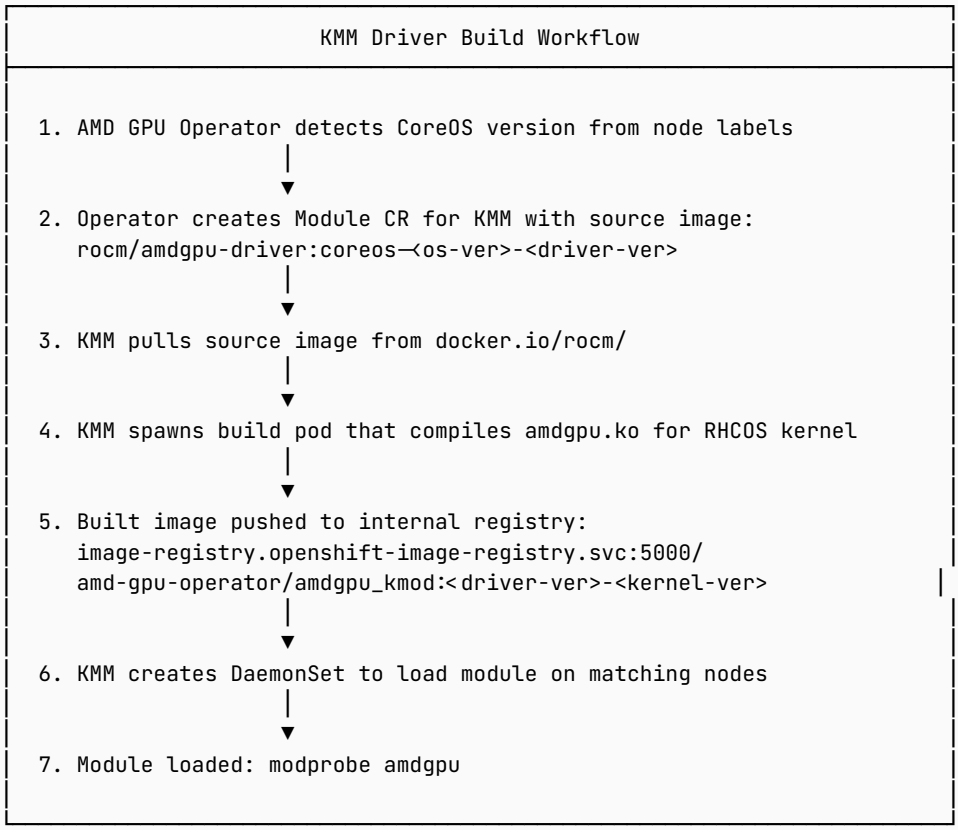
```
docker.io/rocm/amdgpu-driver:coreos—<os-version>-<driver-version>
```

For example, on RHCOS 9.8 with driver version 31.30:

- **Image**: `docker.io/rocm/amdgpu-driver:coreos-9.8-31.30`
- **Contents**: Kernel module source code and build scripts for the `amdgpu` driver

The operator queries the node's OS version label and uses the `version` field from DeviceConfig to determine which source image to pull. This ensures compatibility without manual configuration.

Build Workflow



Verify Driver Loading

```
# Check KMM Module status
oc get module -n amd-gpu-operator

# Check if driver build completed
oc get pods -n amd-gpu-operator -l kmm.node.kubernetes.io/job-type=build

# Check if driver is loaded on node
oc debug node/<node-name> -- chroot /host lsmod | grep amdgpu

# View KMM logs
oc logs -n openshift-kmm -l app.kubernetes.io/name=kmm

# Check built driver image in internal registry
oc get imagestream -n amd-gpu-operator
```

Troubleshooting KMM

ISSUE	DIAGNOSIS	SOLUTION
BUILD POD FAILS	`oc logs -n amd-gpu-operator <build-pod>`	Check kernel version compatibility
MODULE NOT LOADING	`oc describe module -n amd-gpu-operator`	Verify node selector matches
IMAGE PULL ERROR	Check `oc get events -n amd-gpu-operator`	Verify registry access and pull secrets

ISSUE	DIAGNOSIS	SOLUTION
DRIVER VERSION MISMATCH	`oc get module -o yaml`	Update `version` in DeviceConfig to match CoreOS

Image Registry Requirements

KMM requires the OpenShift internal image registry to store built driver images. The registry must be enabled **before** installing the AMD GPU Operator.

Registry URL:

```
image-registry.openshift-image-registry.svc:5000
```

Storage Considerations:

ENVIRONMENT	STORAGE TYPE	BEHAVIOR
SINGLE-NODE OPENSHIFT (SNO)	`emptyDir`	Driver images are ephemeral and rebuilt after node reboots. Acceptable for dev/test.
PRODUCTION MULTI-NODE	PVC (ReadWriteMany)	Driver images persist across pod restarts. Required for production.

For production storage configuration, see Red Hat's image registry documentation.

Verify Registry Status:

```
# Check registry management state
oc get configs.imageregistry.operator.openshift.io/cluster -o jsonpath='{.spec.managementState}'

# Check registry pods
oc get pods -n openshift-image-registry

# View built driver images (after GPU operator is installed)
oc get imagestream -n amd-gpu-operator
```

Expected output after GPU driver is built:

```
NAME          IMAGE REPOSITORY                                                              TAGS
amdgpu_kmod   image-registry.openshift-image-registry.svc:5000/amd-gpu-operator/amdgpu_kmod   31.30—<kernel-
version>
```

Why Source Build vs Pre-built?

The `useSourceImage: true` approach is preferred because:

1. **Kernel compatibility**: Driver is compiled against the exact kernel running on your nodes
2. **Security**: No need for unsigned pre-built modules
3. **Flexibility**: Works across different RHCOS versions
4. **Supportability**: Red Hat-supported workflow for out-of-tree drivers

For more information, see the KMM documentation.

F. OPENSHIFT CLUSTER DEPLOYMENT

> **Note:** This section is **optional** and describes a **development/test setup only**. For production OpenShift clusters, follow Red Hat's official installation documentation and consult with Red Hat support for architecture planning, high availability, and production best practices.

This section provides step-by-step instructions for deploying a Single Node OpenShift (SNO) cluster using Agent-based installation for development and testing purposes.

Prerequisites

Before deployment, ensure you have:

- Physical server with AMD Instinct GPU
- RHEL 9.x provisioning host with network access to target server
- Red Hat pull secret (cloud.redhat.com)
- BMC/iDRAC/iLO credentials for target server
- Network configuration (IP, subnet, gateway, DNS)

Step 1: Configure Ansible Inventory and Variables

Edit `inventory/hosts.ini` with your provisioning host and SNO node details:

```
[provisioning_host]
localhost ansible_connection=local ansible_become=true

[sno_node]
sno-0.sno.ocp.local bmc_ip=10.216.177.124 mac=40:5b:7f:8e:bf:10 ip=10.216.177.95

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

Edit `vars/cluster.yml` with your environment details:

```
# Cluster Identity
cluster_name: sno
base_domain: ocp.local
install_dir: /home/amd/ocp-install     # CHANGE: installation directory path

# OpenShift Version
ocp_version: "4.22.0"
ocp_mirror: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp"

# Network Configuration
machine_network_cidr: 10.216.176.0/22   # SNO node's actual subnet

# SNO Node Configuration
sno_node_ip: 10.216.177.163             # CHANGE: IP for the SNO node
sno_node_mac: "40:5b:7f:8e:bf:10"       # CHANGE: MAC address for SNO node primary NIC
sno_node_idrac_ip: 10.216.177.124       # CHANGE: iDRAC IP for SNO node BMC
sno_node_hostname: sno-0.sno.ocp.local  # CHANGE: FQDN for SNO node
sno_node_gateway: 10.216.179.254        # CHANGE: Gateway for SNO node subnet

# Provisioning Host NIC
baremetal_nic: eno17395np0              # CHANGE: NIC on provisioning host

# NTP Servers
ntp_servers:
  - time.cloudflare.com
  - pool.ntp.org

# BMC Settings
bmc_scheme: idrac-virtualmedia          # CHANGE: idrac-virtualmedia | redfish-virtualmedia
```

```
bmc_system_id: System.Embedded.1        # CHANGE: System.Embedded.1 (Dell) | system (OpenBMC)

# DNS Configuration
dns_server: 10.216.149.60               # Provisioning host (runs BIND DNS)
dns_forwarder: 10.218.15.1              # Corporate DNS (for external queries)

# GPU Configuration
sno_node_has_gpu: true

# Storage Configuration
sno_root_disk_serial: "CN0NH7WDFCP005CH01HM"  # CHANGE: Serial number of root disk

# AI Model Cache Storage Configuration
# Note: Run setup_lvms_storage.yml OR setup_nfs_on_host.yml to deploy storage
ai_model_cache_backend: lvms            # Documentation only: lvms (Recommended for SNO) or nfs
ai_model_cache_pvc_name: "ai-model-cache"

# LVMS Configuration (used by setup_lvms_storage.yml playbook)
lvms_namespace: "openshift-storage"
lvms_device_class_name: "vg1"
lvms_device_paths:
  - "/dev/nvme1n1"                      # CHANGE: Data disk for LVMS (must exist and be unused)
lvms_thin_pool_size_percent: 90         # Use 90% of device for thin pool
lvms_overprovision_ratio: 10            # Allow 10x overprovisioning
model_cache_size: "500Gi"               # Size of AI model cache PVC

# NFS Configuration (used by setup_nfs_on_host.yml playbook)
nfs_export_path: "/home/amd/srv/nfs"   # NFS export directory on provisioning host
nfs_export_size: "500Gi"                # Size of NFS export and PVC
```

Edit `vars/secrets.yml` with credentials:

```
pull_secret: '{"auths": {...}}'  # From cloud.redhat.com
sno_bmc_password: "your-bmc-password"
ssh_public_key: "ssh-rsa AAAA... your-key"
```

Step 2: Setup Provisioning Host

Install required tools and download OpenShift binaries:

```
ansible-playbook playbooks/setup_provisioning_host.yml --ask-vault-pass
```

This installs:

- `openshift-install` binary
- `oc` and `kubectl` CLI tools
- `helm` v3.21.2
- Python dependencies (jinja2, jmespath)
- Required packages (nmstate, netaddr)

Step 3: Setup DNS and NTP (Required)

IMPORTANT: DNS configuration is **required** for Agent-based installer. The bootstrap process requires DNS resolution for cluster API endpoints before installation can proceed.

Setup DNS and NTP on the provisioning host:

```
ansible-playbook playbooks/setup_infra.yml --ask-vault-pass
```

This configures:

- **BIND DNS** with required records:
  - `api.sno.ocp.local` → SNO node IP
  - `api-int.sno.ocp.local` → SNO node IP
  - `*.apps.sno.ocp.local` → SNO node IP (wildcard)
- **Chrony (NTP)** for time synchronization
- **Firewall rules** for DNS service

The playbook also verifies that DNS records resolve correctly before proceeding.

> **Note:** While `/etc/hosts` entries can work for post-installation route access, they **cannot replace DNS** during the Agent-based installation process. The installer requires a functioning DNS server.

Step 4: Generate OpenShift Installation ISO

Run the cluster deployment playbook to generate the installation ISO:

```
ansible-playbook playbooks/deploy_cluster.yml --ask-vault-pass
```

The playbook performs:

1. Generates Agent-based installer manifests
2. Creates OpenShift installation ISO at `{{ install_dir }}/agent.x86_64.iso`

ISO Location:

```
ls -la /home/amd/ocp-install/agent.x86_64.iso
```

Step 5: Mount ISO via BMC Virtual Media (Manual)

> **Note:** ISO mounting is a **manual process**. Access your server's BMC (iDRAC/iLO) to mount the ISO.

For Dell iDRAC:

1. Download the ISO from the provisioning host to your local machine:

   ```
   scp amd@<provisioning_host_ip>:/home/amd/ocp-install/agent.x86_64.iso .
   ```

2. Access iDRAC web console: `https://<bmc_ip>` (e.g., `https://10.216.177.124`)
3. Login with BMC credentials
4. Navigate to **Configuration → Virtual Console** and click **Launch Virtual Console**
5. In the Virtual Console window, click **Virtual Media → Connect Virtual Media**
6. Click **Map CD/DVD → Browse** and select the downloaded `agent.x86_64.iso`
7. Click **Map Device** to mount the ISO
8. Reboot the server: **Power → Reset System (warm reboot)**
9. During boot, press **F11** to enter Boot Manager

10. Select **Virtual CD/DVD** as the boot device

Step 6: Monitor Installation

Once the server boots from the ISO, installation begins automatically:

Installation Timeline:

- RHCOS installation: 10-15 minutes
- Bootstrap process: 20-30 minutes
- Cluster operators ready: 10-15 minutes
- **Total:** ~45-60 minutes

Step 7: Verify Cluster

Wait for installation to complete:

```
# Monitor installation progress
tail -f /home/amd/ocp-install/.openshift_install.log

# Wait for cluster operators
export KUBECONFIG=/home/amd/ocp-install/auth/kubeconfig
oc get co

# Verify all operators are Available=True, Progressing=False, Degraded=False
```

Validate cluster deployment:

```
ansible-playbook playbooks/validate_cluster.yml --ask-vault-pass
```

Step 8: Access OpenShift Console

Get the console URL and credentials:

```
# Console URL
oc get route console -n openshift-console \
  -o jsonpath='{.spec.host}'

# Admin password
cat /home/amd/ocp-install/auth/kubeadmin-password
```

Add the console route to `/etc/hosts`:

```
echo "10.216.177.95  console-openshift-console.apps.sno.ocp.local" \
  >> /etc/hosts
```

Access the console at:
`https://console-openshift-console.apps.sno.ocp.local`

Step 9: Verify Basic Cluster Health

Check overall cluster health:

```
# Verify all cluster operators are available
oc get clusteroperators

# Check node status
oc get nodes

# Verify critical pods are running
oc get pods -A | grep -E "openshift-|kube-"
```

All cluster operators should show `AVAILABLE=True`, `PROGRESSING=False`, and `DEGRADED=False`.

Troubleshooting

Installation stuck at "Waiting for all hosts to be ready":

This is typically caused by host validation failures. SSH into the node and check the assisted-service logs:

```
ssh core@<sno-node-ip>
sudo journalctl | grep -i "assisted-service\|assisted_service" | tail -50
```

Look for validation failures in the log output. Common issues:

1. **"Host does not belong to machine network CIDRs"**

   The node's IP address doesn't match the `machineNetwork.cidr` configured in `install-config.yaml`.

   Diagnose: Check the ISO configuration vs actual node IP:

   ```
   # On provisioning host - check what CIDR was embedded in the ISO
   cat /home/amd/ocp-install/install-config.yaml.bak | grep -A2 machineNetwork

   # On SNO node - check actual IP
   ip addr show | grep "inet "
   ```

   Fix: Ensure `machine_network_cidr` in `vars/cluster.yml` contains the node's IP address, then regenerate the ISO:

   ```
   # Example: If node IP is 10.216.177.163, CIDR should be 10.216.176.0/22 (or appropriate subnet)
   ansible-playbook playbooks/deploy_cluster.yml --ask-vault-pass
   ```

2. **DNS resolution failing or wrong DNS server configured**

   The node is using corporate DNS instead of the provisioning host DNS where OpenShift records are configured.

   Diagnose: Check DNS configuration on the node:

   ```
   ssh core@<sno-node-ip>
   cat /etc/resolv.conf
   # Should show provisioning host IP (e.g., 10.216.149.60), not corporate DNS
   ```

   Fix: Update `dns_server` in `vars/cluster.yml` to point to the provisioning host:

   ```
   dns_server: 10.216.149.60  # Provisioning host running BIND DNS
   ```

   Then run DNS setup and regenerate the ISO:

```
ansible-playbook playbooks/setup_infra.yml --ask-vault-pass
ansible-playbook playbooks/deploy_cluster.yml --ask-vault-pass
```

3. NTP synchronization failure

Check if time is synced:

```
ssh core@<sno-node-ip>
timedatectl
```

Installation hangs during bootstrap:

- Check BMC console for boot issues
- Verify virtual media is mounted:
  `ipmitool -I lanplus -H <bmc-ip> -U <user> -P <pass> sol activate`
- Check installer logs: `tail -f .openshift_install.log`

Node not booting from ISO:

- Verify BMC virtual media mount status
- Check boot order in BIOS (virtual media should be first)
- Try manual reboot with ipmitool: `ipmitool power reset`

Cluster operators degraded:

- Check operator logs:
  `oc logs -n openshift—<operator-namespace> <pod-name>`
- Verify DNS resolution:
  `dig api.sno.ocp.local` and `dig *.apps.sno.ocp.local`
- Check network connectivity from node to internet (for image pulls)

Verifying ISO configuration before deployment:

Always verify the ISO configuration matches your intended settings before rebooting the node:

```
# Check the backup files created during ISO generation
cat /home/amd/ocp-install/install-config.yaml.bak
cat /home/amd/ocp-install/agent-config.yaml.bak

# Key things to verify:
# - machineNetwork.cidr contains the SNO node IP
# - dns-resolver.config.server points to provisioning host
# - rendezvousIP matches the SNO node IP
# - networkConfig.interfaces has correct IP and prefix-length
```

G. MODEL DEPLOYMENT VIA RHOAI DASHBOARD

This appendix provides visual step-by-step guidance for deploying models through the RHOAI Dashboard.

Step 1: Access RHOAI Dashboard

Navigate to the RHOAI dashboard and select **Models → Deploy model**.

Step 2: Model Location

Configure the model storage location:

- **Model location:** Select `URI`
- **URI:** `pvc://ai-model-cache`
- Check **Create a connection to this location**
- **Name:** Enter model name (e.g., `gpt-oss-20b`)
- **Model type:** `Generative AI model (Example, LLM)`

Model Deploy Step 2

Model Deploy Step 2

Step 3: Model Deployment Configuration

Configure deployment settings:

- **Project:** Select `ai-workloads`
- **Model deployment name:** Enter deployment name
- **Hardware profile:** Select `AMD Instinct GPU`
- **Customize resource requests:**
  - **CPU:** `12` Cores
  - **Memory:** `128` GiB
  - **AMD GPU:** `1`
- **Deployment resource:** Select **Manual selection**
  - Choose appropriate AMD AIM serving runtime
- **Number of replicas:** `1`

Model Deploy Step 3

Model Deploy Step 3

Step 4: Select Serving Runtime

Choose the appropriate AMD AIM serving runtime template for your model:

Model Deploy Step 4

Model Deploy Step 4

Step 5: Advanced Settings

Configure advanced deployment options:

- Check **Make model deployment available through an external route**
- **Token authentication:** Uncheck (for testing)

Model Deploy Step 5

Model Deploy Step 5

Step 6: Review and Deploy

Review all configuration settings and click **Deploy**:

Model Deploy Step 6

Model Deploy Step 6

The model deployment will begin, and you can monitor its progress in the **Models** section of the RHOAI dashboard.

H. SETTING UP SAMBA FOR IDRAC VIRTUAL MEDIA

This appendix provides instructions for configuring a Samba (CIFS/SMB) file share on Red Hat Enterprise Linux to host ISO files for iDRAC Virtual Media remote file share deployments.

Overview

iDRAC Virtual Media Remote File Share feature supports CIFS/SMB protocol for mounting ISO files remotely. This eliminates the need to physically access servers or use local media for OS installations and updates.

Key Points:

- NFS is **not supported** for iDRAC Virtual Media Remote File Share
- Only CIFS/SMB (Samba) protocol is supported
- Requires proper SELinux configuration on RHEL
- DNS server configuration must be separate from Samba authentication

Prerequisites

- Red Hat Enterprise Linux system (provisioning host)
- Root or sudo access
- ISO files to share (e.g., OpenShift discovery ISO)
- Network connectivity between iDRAC and RHEL system

Installation and Configuration

Step 1: Install Samba

```
sudo dnf install samba samba-common samba-client -y
```

Step 2: Create shared directory

This example uses `/home/amd/srv/smb` as the share location:

```
sudo mkdir -p /home/amd/srv/smb
sudo chmod 755 /home/amd/srv/smb
```

Step 3: Set ownership

```
# Set appropriate ownership (adjust user/group as needed)
sudo chown -R amd:amd /home/amd/srv/smb
```

Step 4: Configure Samba

Edit the Samba configuration file:

```
sudo nano /etc/samba/smb.conf
```

Add the following configuration at the end of the file:

```
[smb]
   path = /home/amd/srv/smb
   browseable = yes
   read only = yes
   guest ok = no
   valid users = idrac
   create mask = 0644
   directory mask = 0755
```

Configuration parameters explained:

- `path` - Directory path containing ISO files
- `browseable` - Share appears in network browse lists
- `read only` - Prevents modifications (recommended for ISO shares)
- `guest ok` - Requires authentication (set to `no`)
- `valid users` - Only specified users can access (separate from DNS user)
- Share name `[smb]` will be used in the UNC path

Step 5: Create Samba user for iDRAC authentication

```
# Create a system user for iDRAC authentication (no shell access)
sudo useradd -M -s /sbin/nologin idrac

# Set Samba password for this user
sudo smbpasswd -a idrac
# Enter password when prompted (you'll use this in iDRAC)
```

> **Important:** This `idrac` user is for Samba authentication only and is separate from the `amd` user that runs DNS services.

Step 6: Configure SELinux (RHEL-specific)

SELinux must be configured to allow Samba to share files, especially from home directories:

```
# Install SELinux management tools if not present
sudo dnf install policycoreutils-python-utils -y

# Set SELinux context for the Samba share
sudo semanage fcontext -a -t samba_share_t "/home/amd/srv/smb(/.*)?"
sudo restorecon -Rv /home/amd/srv/smb

# Enable Samba to read home directories (required if share is under /home)
sudo setsebool -P samba_enable_home_dirs on

# Enable Samba to export read-only shares
sudo setsebool -P samba_export_all_ro on
```

> **Note:** The `-P` flag makes SELinux boolean changes persistent across reboots.

Step 7: Configure firewall

```
# Allow Samba through firewall
sudo firewall-cmd --permanent --add-service=samba
sudo firewall-cmd --reload

# Verify firewall rules
sudo firewall-cmd --list-services | grep samba
```

Step 8: Start and enable Samba service

```
# Start Samba service
sudo systemctl start smb

# Enable Samba to start on boot
sudo systemctl enable smb

# Check service status
sudo systemctl status smb
```

Verification and Testing

Test Samba configuration syntax

```
# Validate smb.conf syntax
sudo testparm

# Should show "Loaded services file OK" and display configuration
```

Test local Samba connectivity

```
# List shares available on localhost
smbclient -L //localhost -U idrac

# Access the share interactively
smbclient //localhost/smb -U idrac
# Enter password when prompted
# Use 'ls' to list files, 'quit' to exit
```

Test from network

From another machine or the iDRAC interface:

```
# Replace RHEL_IP with your provisioning host's IP
smbclient //RHEL_IP/smb -U idrac
```

Check Samba logs for troubleshooting

```
# View Samba daemon logs
sudo tail -f /var/log/samba/log.smbd

# View client connection logs
sudo tail -f /var/log/samba/log.
```

iDRAC Configuration

Configure the iDRAC Virtual Media Remote File Share with the following settings:

Navigation: iDRAC → Configuration → Virtual Media → Remote File Share 1

FIELD	VALUE	EXAMPLE
DIRECTORY/FILE PATH	`\\<RHEL_IP>\smb\<filename.iso>`	`\\10.216.149.60\smb\agent.x86_64.iso`
DOMAIN NAME	(leave empty for local authentication)	
USER NAME	`idrac`	`idrac`
PASSWORD	(password set with smbpasswd)	
EXPIRED OR INVALID CERTIFICATE ACTION	`Yes`	`Yes`

UNC Path Format:

- Use double backslashes: `\\server\share\file.iso`
- Replace `<RHEL_IP>` with your provisioning host's IP address
- Replace `<filename.iso>` with the actual ISO filename
- Share name is `smb` (as defined in smb.conf)

Example full path:

```
\\10.216.149.60\smb\agent.x86_64.iso
```

File Management

Adding ISO files to the share

```
# Copy ISO files to the Samba share directory
sudo cp /path/to/your-file.iso /home/amd/srv/smb/

# Set proper permissions
sudo chmod 644 /home/amd/srv/smb/your-file.iso
sudo chown amd:amd /home/amd/srv/smb/your-file.iso
```

List available ISO files

```
ls -lh /home/amd/srv/smb/
```

Troubleshooting

Connection refused from iDRAC

Check firewall:

```
sudo firewall-cmd --list-all
```

Check Samba is listening:

```
sudo ss -tulpn | grep :445
sudo ss -tulpn | grep :139
```

Permission denied errors

Check SELinux denials:

```
sudo ausearch -m avc -ts recent | grep -i samba
```

Temporarily disable SELinux for testing (not recommended for production):

```
sudo setenforce 0
# Test connection
# Re-enable: sudo setenforce 1
```

Verify file permissions:

```
ls -lZ /home/amd/srv/smb/
```

Authentication failures

Test Samba user:

```
sudo pdbedit -L -v | grep idrac
```

Reset Samba password:

```
sudo smbpasswd idrac
```

DNS Server vs Samba Authentication

Important distinction:

- The provisioning host runs **two separate services**:
  1. **DNS server (BIND)** - Uses system user `amd` to run the service
  2. **Samba server** - Uses Samba user `idrac` for authentication

These are completely separate:

- DNS service runs as Linux user `amd`
- Samba clients authenticate as Samba user `idrac`
- The SNO node uses DNS at `10.216.149.60` (no authentication)
- iDRAC uses Samba at `\\10.216.149.60\smb` (with `idrac` user authentication)

Can't browse share from Windows

Enable NetBIOS:

```
sudo systemctl start nmb
sudo systemctl enable nmb
sudo firewall-cmd --permanent --add-service=samba
sudo firewall-cmd --reload
```

Security Considerations

1. **Read-Only Share**: The example configuration sets `read only = yes` to prevent accidental modifications to ISO files.
2. **Authentication Required**: `guest ok = no` ensures only authenticated users can access the share.
3. **User Isolation**: The `idrac` user has no shell access (`/sbin/nologin`) and is only used for Samba authentication.
4. **SELinux**: Keep SELinux in enforcing mode and configure proper contexts rather than disabling it.

5. **Firewall**: Only allow Samba from trusted networks. Consider limiting access:

```
# Allow Samba only from specific subnet
sudo firewall-cmd --permanent --zone=trusted --add-source=10.216.0.0/16
sudo firewall-cmd --permanent --zone=trusted --add-service=samba
sudo firewall-cmd --reload
```

6. **Password Strength**: Use strong passwords for the Samba `idrac` user.

Alternative Locations

If you prefer a different share location, adjust paths accordingly:

Example: Using /srv/iso-share

```
# Create directory
sudo mkdir -p /srv/iso-share
sudo chmod 755 /srv/iso-share

# SELinux context
sudo semanage fcontext -a -t samba_share_t "/srv/iso-share(/.*)?"
sudo restorecon -Rv /srv/iso-share

# Update smb.conf
[iso-share]
   path = /srv/iso-share
   ...
```

Then use `\\10.216.149.60\iso-share\file.iso` in iDRAC.

I. REFERENCES

- Red Hat OpenShift Documentation
- Red Hat OpenShift AI Documentation
- AMD ROCm Documentation
- AMD GPU Operator Documentation
- KServe Documentation

Document Version: 1.3
Last Updated: October 2026

DISCLAIMER

This document has been produced by Advanced Micro Devices, Inc. (AMD) and reflects AMD's independent analysis, testing, and documentation. The information contained herein has been developed without input, review, or endorsement from Red Hat, Inc., The Linux Foundation, Docker, Inc., Hugging Face, Inc., or any other third-party vendors, partners, or organizations mentioned in this document. Any references to third-party products, services, or trademarks are provided solely for technical guidance and integration purposes and do not imply endorsement, sponsorship, or affiliation.

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or

revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

TRADEMARKS AND PRODUCT NAMES

Red Hat, Red Hat Enterprise Linux, Red Hat OpenShift, and Red Hat Ansible are trademarks of Red Hat, LLC, registered in the United States and other countries. Red Hat OpenShift AI, Red Hat OpenShift Container Platform, Red Hat OpenShift Service Mesh, Red Hat OpenShift Serverless, and Red Hat Enterprise Linux CoreOS are trademarks of Red Hat, LLC or its subsidiaries.

All other product names, trademarks, registered trademarks, and service marks mentioned herein are the property of their respective owners.

© 2026 Advanced Micro Devices, Inc. All rights reserved.