



# Technology Partner Program

Use Case Documentation

Author: Red Hat



Revision History	
04-8-2025	New Integration

Table 1: Partner information	
Date	04-08-2025
Partner Name	Red Hat
Website	<a href="http://www.RedHat.com">www.RedHat.com</a>
Product Name	OpenShift Virtualization
Partner Contact	John Allen Gibson, Global ISV PAM <a href="mailto:joqibson@redhat.com">joqibson@redhat.com</a>
Support Contact	
Product Description	<p>Red Hat® OpenShift® Virtualization, an included feature of Red Hat OpenShift, provides a modern platform for organizations to run and deploy their new and existing VM workloads. The solution allows for easy migration and management of traditional VMs onto a trusted, consistent, and comprehensive hybrid cloud application platform.</p> <p>OpenShift Virtualization simplifies the migration of your VMs while offering a path for infrastructure modernization, taking advantage of the simplicity and speed of a cloud-native application platform. It aims to preserve existing virtualization investments while embracing modern management principles—and it's the foundation for Red Hat's comprehensive virtualization solution.</p>

## Use Cases for Integration with Palo Alto Networks

### Use Case

**VMware Alternative:** With the acquisition of VMware by Broadcom, companies that leverage VMware virtualization products are forced to confront several challenges that could negatively impact their production.

Customer concerns over the possible disruption of products and support, increases in pricing, alteration of licensing agreements, or, worst, the risk of vendor lock-in, which could result from the absorption of VMware into Broadcom, have many companies searching for a viable replacement.



## Use Case

**Cloud Integration and Hybrid Solutions:** Many businesses are migrating to hybrid multi-cloud environments. The use of Kubernetes for container orchestration OpenShift for cloud infrastructure provides better compatibility with cloud-native applications.

Table 2: Palo Alto Networks Products for Integration			
Palo Alto Networks Product	Integration Status	Palo Alto Networks Versions Tested	Red Hat Versions Tested
Cortex XDR			
Xpanse			
GlobalProtect			
IoT Security			
Prisma Access			
Prisma Cloud			
Prisma SaaS			
Prisma SD-WAN			
Next-Generation Firewall	Validated	PAN-OS 11.2.5	OpenShift 4.17
Panorama	Validated	PAN-OS 11.2.5	OpenShift 4.17
VM-Series	Validated	PAN-OS 11.2.5	OpenShift 4.17
WildFire			
Other			

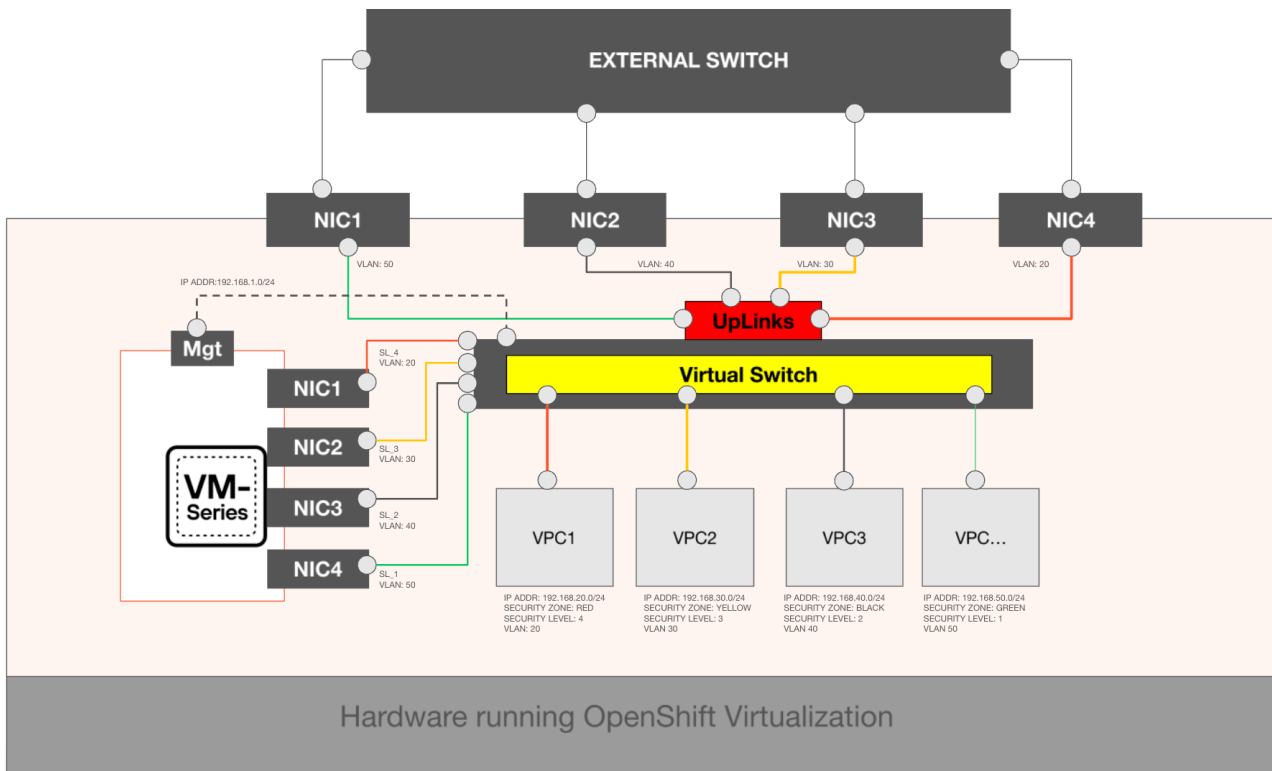
## Instructions

To our valued partner, the purpose of the Integration Guide is to give customers a unified, comprehensive document that provides step-by-step configuration instructions, as well as any supporting reference and troubleshooting information, with the end goal of facilitating reproducible and successful deployment of both products. It is important to ensure that customers can successfully deploy the joint solution without having to visit multiple documents, as well as have step-by-step configuration settings that have been tested and validated, and are known to work without issues. Please ensure you remove this section from the final version of the guide.

## Integration Benefits

- **Enhanced Security for Virtual Machines:** Advance threat prevention for VMS running on OpenShift, layer 7 application visibility and control, protection against known and unknown threats, and real-time intelligence integration
- **Unified Security Policy Management:** consistent security policies across VM workloads, centralized policy management through Panorama, automated security policy deployment, and reduced complexity in security operations
- **Network Segmentation:** Micro-segmentation capabilities for VM workloads, Zero-trust security model implementation, isolation between different VM environments, secure communication between containers and VMs
- **Advance Threat Protection:** URL filtering, Anti-malware protection, IPS.IDS capabilities, DNS security, file blocking and analysis
- **Compliance Benefits:** Detailed logging and reporting, audit trail for regulatory compliance, policy enforcement documentation, security posture visibility
- **Operational Advantages:** Automated security deployment, reduced manual configuration, simplified security management , and consistent security across hybrid environments
- **Performance Benefits:** Hardware-accelerated security processing, optimized traffic flow, reduced latency, scale-out architecture support
- **Cloud-Native Security:** Native integration with OpenShifts' CNI, support for dynamic workload scaling, container-aware security policies, and cloud-native app protection
- **Visibility and Analytics:** Deep packet inspection, application identification, user identification, threat correlation, and security analytics and reporting
- **Disaster Recovery:** VM-Level protection, backup and recovery integration, business continuity support, and high availability configurations

## Integration Diagram



## Before You Begin

List out dependencies needed before the customer begins.

For a bare metal deployment of OpenShift Container Platform (OCP) with OpenShift Virtualization, the minimum hardware requirements depend on whether you're deploying a compact (minimal) cluster (for testing/dev) or a production-grade cluster. Below are the minimum requirements for a compact cluster (non-production, PoC, or lab environment):


### 1. Minimum Hardware Requirements (Bare Metal)

#### Control Plane (Master) Nodes

- Number of Nodes 3 (Highly Available)
- vCPUs per Node 4 cores (8 vCPUs recommended)
- RAM per Node 16 GB (32 GB recommended)
- Storage per Node
  - OS Disk 100 GB (for `/var``, `/usr``, etc.)
  - Etc Disk 50 GB (SSD/NVMe recommended for performance)
  - Container Storage 100 GB (for `/var/lib/containers``)

#### Worker Nodes (with OpenShift Virtualization)

- Number of Nodes 2 (1 is possible but not HA)
- vCPUs per Node 8 cores (16+ recommended for VMs)
- RAM per Node 32 GB (64 GB+ recommended for running VMs)
- Storage per Node
  - OS Disk 100 GB
  - Container Storage 100 GB (for `/var/lib/containers``)
  - Virtual Machine Storage 200 GB+ (for VM disks, preferably fast storage like SSD/NVMe)



## Bootstrap Node (Temporary)

- vCPUs 4 cores
- RAM 16 GB
- Storage 100 GB (only needed during installation)

## 2. Networking Requirements

Network Interface 1 Gbps NIC (10 Gbps recommended for production)

DNS & Load Balancer: Required for API and ingress traffic.

Static IPs: Required for control plane and worker nodes.

## 3. Storage Requirements

- OpenShift Virtualization needs persistent storage for VMs (e.g., OpenShift Data Foundation (ODF) or external storage like NFS, iSCSI, Ceph).
- Minimum for ODF (if used):
  - 3 nodes with 500 GB raw storage each (for a minimal storage cluster).

## 4. Additional Notes

- Single-Node OpenShift (SNO): Possible for testing, but not recommended for virtualization workloads.
- Production Recommendations:
  - Control Plane: 8+ vCPUs, 64 GB RAM per node.
  - Workers: 16+ vCPUs, 128 GB RAM per node (for running multiple VMs).
  - Storage: Fast disks (NVMe) for VM performance.

- **List out requirements for successful integration.**

Red Hat requirements:

- Access to Red Hat Support portal
- All necessary permissions for downloading on necessary software packages

Palo Alto Network requirements:

- Access to Palo Alto Networks Support Portal with access and permissions to download software updates
- License to NGFW and subscriptions to products you will be running on NGFW.

GitHub

- Active GitHub account

- **List out any dependencies in OS, version, Panorama, etc.**

Check [www.redhat.com](http://www.redhat.com) to find the complete list of hardware and software necessary to deploy OpenShift on a baremetal machine.

To successfully integrate Red Hat OpenShift Virtualization with Palo Alto Networks VM-Series Next-Generation Firewalls, you need to ensure that each component is compatible with the others. Below is a detailed list of dependencies, including operating systems, versions, Panorama, and Red Hat OpenShift requirements.

## 1. Red Hat OpenShift Requirements

### OpenShift Version

- Recommended Version: OpenShift Container Platform (OCP) 4.17 or later.
- Compatibility: Ensure that the OpenShift version supports virtualization features. OpenShift 4.17 introduced significant improvements for virtualization.

### Operating System

- 
- Base OS: Red Hat Enterprise Linux (RHEL) 9.x is recommended for the OpenShift nodes.
  - Kernel Version: Ensure that the kernel version is compatible with OpenShift and supports KVM-based virtualization.

## 2. Palo Alto Networks VM-Series Next-Generation Firewall Requirements

### VM-Series Version

- Recommended Version: PAN-OS 10.2 or later.
- Compatibility: Ensure that the PAN-OS version supports integration with Kubernetes and OpenShift.

### Deployment Environment

- VM Type: Use the VM-Series firewalls as virtual machines (VMs) on the OpenShift cluster.
- Resources: Allocate adequate CPU, memory, and storage resources based on the expected workload and throughput.

## 3. Panorama Requirements

### Panorama Version

- Recommended Version: Panorama 10.2 or later.
- Compatibility: Ensure that Panorama can manage the VM-Series firewalls deployed on OpenShift.

## 4. Networking & Integration Dependencies

### Networking

- Virtual Networking: Ensure that OpenShift networking (such as OpenShift SDN or OVN-Kubernetes) is configured to allow the VM-Series to monitor and control traffic.
- IP Addressing: Assign appropriate IP addresses to VM-Series firewalls and ensure routing is correctly configured.

- **List out requirements for API keys, if applicable.**

### Palo Alto Networks API Key Requirements

#### Palo Alto Networks API Access

- API Key Generation: You need to generate an API Key from the Palo Alto Networks firewall or Panorama. This key will be used to authenticate API requests
- Permissions: the user account used to generate the API key must have sufficient privileges to perform the necessary actions. Ensure the account has the following roles:
  - Read and write permissions on relevant configurations
  - Access to logs

### Red Hat OpenShift API Key Requirements

#### OpenShift API Access

- Service Account: Create a service account in Openshift specifically for the integration.
- Roles and Permissions: Assign appropriate roles to the service account to allow it to access and manage resources required for integration. Common roles may include:
  - Edit: To manage application resources
  - View: To read application resources
  - Custom roles may be created based on the specific needs for your integration
- **List out any applicable dependencies.**

#### Server environment

- Operational OpenShift Cluster and administrative rights on the cluster
- Palo Alto Networks VM-Series Firewall with proper licensing for the VM-Series Firewall features you intend to use

#### Networking

- Defined Network Security Group: Make sure that the required ports are open between OpenShift and the Palo Alto Networks firewall for traffic management. Common ports include:
  - Management: TCP 443

- API access: TCP 443 (if using API integration)
- Routing configuration: Proper routing must be established to direct traffic through the Palo Alto Networks firewall for inspection and policy enforcement

#### Software

- Palo Alto Networks API access:
  - Ensure you have access to the Palo Alto Networks API, which may require an API key or token for authentication.
  - Familiarity with RESTful APIs for programmatic access to the firewall
- OpenShift CLI Tools
  - oc command-line tool for managing OpenShift resources and deploying applications

#### Integration Steps

1. Install the VM-Series Firewall: Deploy the VM-Series firewall as a pod in your OpenShift cluster.
2. Configure Firewall Policies: Set up security policies in the VM-Series firewall to monitor and secure traffic between OpenShift services and external networks.
3. Panorama Configuration: Integrate the VM-Series firewalls with Panorama for centralized management and visibility.

## Palo Alto Networks Configuration

### Partner Product Configuration

- List the steps for the customer to integrate the products.
- Provide screenshots with real-world examples that clearly depict the step-by-step process required for the customer to configure the Palo Alto Networks product(s) in their environment
- Provide “reference” links to the corresponding pages on your support portal/technical documentation site, such that customers can quickly and easily pivot to read more about the feature set.

### Prerequisites

Use the Red Hat Console to stand up a bare metal cluster using the [Assisted-Installer](#). Use the following recommendations as you progress the installer:

- Be sure that your cluster has at least 2 physical interfaces (or more) and that at least one of those interfaces is provided by a supported [SR-IOV capable](#) NIC.
- Be sure that your cluster has 2 physical disks (or more).
- Be sure that you have a trunked VLAN interface going to one of the two interfaces.
- Do not install the storage or virtualization operators as part of the installation progress. These guides will tell you how to install and configure each operator manually. This offers more control and provides admins with a better understanding of how to install and manage these services.

Please ensure you have completed the below steps before moving onto setting up the Palo Alto VM-Series on OpenShift Virtualization via the Installation Procedures:

1. Set up and configure the LVM Operator according to the instructions located [HERE](#)
2. Set up and configure the OpenShift Virtualization Operator according to the instructions located [HERE](#)
3. Set up a PerformanceProfile, similar to the example provided [HERE](#)
4. Set up and configure the NMState Operator according to the installation instructions located [HERE](#)
5. Set up and configure the SR-IOV Operator according to the installation instructions located [HERE](#)

## Step 1: Installation of LVM on OpenShift v4.17.x

This Gist will provide a very quick, basic installation of Logical Volume Manager for OpenShift.

### Table of Contents

- [Getting Started](#)
- [Operator Installation](#)
  - [LVM Deployment and Customization](#)
- [Creating a Second StorageClass](#)
- [Create a StorageProfile for CNV](#)
- [Appendix A: Example LVMCluster Deployment Using Labels](#)
- [Appendix B: Useful Disk Tooling](#)

### Getting Started

1. Get information about the disks on the SNO deployment

```
NODE_NAME=$(oc get no -o name)

cat <<EOF | oc debug $NODE_NAME
chroot /host
lsblk -o NAME,ROTA,SIZE,TYPE
EOF
```

The output will look like the following:

```
sh-5.1# lsblk -o NAME,ROTA,SIZE,TYPE
sh-5.1# exit
NAME    ROTA    SIZE TYPE
sda     1       1.8T disk
sdb     1       5.5T disk
sdc     1       7.3T disk
sdd     1       3.6T disk
sde     1       931G disk
|-sde1  1        1M part
|-sde2  1       127M part
|-sde3  1       384M part
`-sde4  1    930.5G part
sr0     1      1024M rom
```

Removing debug pod ...

2. Now I want you to run the following command (SNO is required, and CLI should be working already), which will list out all of the mappings between disk assignments (i.e. `/dev/sdc`) and disk by-path (i.e. `/dev/disk/by-path/`).

```
NODE_NAME=$(oc get no -o name)
```

```
cat <<EOF | oc debug $NODE_NAME
chroot /host
ls -aslc /dev/disk/by-path/
EOF
```

The output will look like the following:

```
sh-5.1# ls -aslc /dev/disk/by-path/
sh-5.1# exit
total 0
0 drwxr-xr-x. 2 root root 260 Sep  7 00:30 .
0 drwxr-xr-x. 9 root root 180 Sep  7 00:30 ..
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:00:17.0-ata-8 -> ../../sr0
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:00:17.0-ata-8.0 -> ../../sr0
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:0:0 -> ../../sda
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:1:0 -> ../../sdc
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:2:0 -> ../../sdb
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:3:0 -> ../../sdd
0 lrwxrwxrwx. 1 root root  9 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:4:0 -> ../../sde
0 lrwxrwxrwx. 1 root root 10 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:4:0-part1 ->
../../sde1
0 lrwxrwxrwx. 1 root root 10 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:4:0-part2 ->
../../sde2
0 lrwxrwxrwx. 1 root root 10 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:4:0-part3 ->
../../sde3
0 lrwxrwxrwx. 1 root root 10 Sep  7 00:30 pci-0000:1a:00.0-scsi-0:2:4:0-part4 ->
../../sde4
```

Removing debug pod ...

3. Next, wipe the disk that you're planning to use for LVM. If you plan on using multiple disks, I recommend that the disks are of the same speed and type, and that they are non-rotational disks (which can be determined from the output above).

*CRITICAL: STOP!! DO NOT simply copy and paste what you see below. Copy, edit for your use case/SNO environment, and then paste. For example, I am using **dev/sdc** (single disk). If you need or want to use a different disk, take your time and edit the following information accordingly.*

```
cat <<EOF | oc debug $NODE_NAME
chroot /host
sudo wipefs -af /dev/sdc
sudo sgdisk --zap-all /dev/sdc
sudo dd if=/dev/zero of=/dev/sdc bs=1M count=100 oflag=direct,dsync
sudo blkdiscard /dev/sdc
EOF
```

## Operator Installation

- Now you can install the following LVM operator manifests. This will install the operator, but it will not install the LVM instance quite yet. We will do this in the next step (with the CR).

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

---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-storage-operatorgroup
  namespace: openshift-storage
spec:
  targetNamespaces:
  - openshift-storage

---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: lvms
  namespace: openshift-storage
spec:
  installPlanApproval: Automatic
  name: lvms-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

### LVM Deployment and Customization

- Next, using the information you retrieved in the [previous section](#) in Step 1 and 2, you can install the LVM CR instance. This will create our LVM storage environment.

**WARNING:** DO NOT simply copy and paste what you see below. Copy, edit for your use case/SNO environment, and then paste. For example, I am using `/dev/sdb` (single disk). If you need or want to use a different disk, take your time and edit the following information accordingly. If you need to add another disk, you can add another option under the line with `/dev/sdb`.

```

cat <<EOF | oc apply -f -
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: lvmcluster
  namespace: openshift-storage
spec:
  storage:
    deviceClasses:
      - name: vg1
        deviceSelector:
          paths:
            - /dev/disk/by-path/pci-0000:1a:00.0-scsi-0:2:1:0
        fstype: xfs
        thinPoolConfig:
          chunkSizeCalculationPolicy: Static
          name: thin-pool-1
          overprovisionRatio: 10
          sizePercent: 90
EOF

```

NOTE: The manifest above will create a *StorageClass* called *Lvms-vg1*. *vg1* is taken from the field above *spec.deviceClasses.[name]*. LVM deployments prepend *StorageClass* objects with *Lvms*, so this gives us a resulting name of *Lvms-vg1*.

### Creating a Second StorageClass

- Now we're going to create another *StorageClass*, but in this case we're going to create a duplicate of *lvms-vg1* which allows for *immediate* binding of *PersistentVolumeClaims* (or claims that are bound, regardless of a requesting workload or corresponding *PersistentVolume*. This is useful in some scenarios, but it won't hurt to use this by default. Because of this, I will make this *StorageClass* default (which is going to be useful when deploying OpenShift Virtualization).

```

cat <<EOF | oc apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: lvms-vg1-immediate
  annotations:
    description: Provides RWO and RWOP Filesystem & Block volumes
    storageclass.kubernetes.io/is-default-class: 'true'
    storageclass.kubevirt.io/is-default-virt-class: 'true'
provisioner: topolvm.io
parameters:
  csi.storage.k8s.io/fstype: xfs
  topolvm.io/device-class: vg1
reclaimPolicy: Delete

```

```
allowVolumeExpansion: true
volumeBindingMode: Immediate
EOF
```

### Create a StorageProfile for CNV

## STOP: DO NOT CONTINUE UNLESS YOU HAVE ALREADY FIRST INSTALLED OPENSIFT VIRTUALIZATION

The following is ONLY applicable AFTER [Installing OpenShift Virtualization on SNO](#). A `StorageProfile` modification is a required after installing OpenShift Virtualization if you want to bring VMs up quickly (i.e. if you notice new VMs lag when starting, it's because you haven't created the correct `StorageProfile` and need to follow the instructions below). A `StorageProfile` is a CR which is included as part of [OpenShift Virtualization/KubeVirt](#). You can read more about `StorageProfile` [HERE](#).

1. With the `lvms-vg1-immediate` `StorageClass` deployed, you will need to create a corresponding `StorageProfile`. Deploy the following `StorageProfile` for the `lvms-vg1-immediate` `StorageClass`. (further documentation can be found [HERE](#)).

```
cat <<EOF | oc apply -f -
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: lvms-vg1-immediate
  labels:
    app: containerized-data-importer
    app.kubernetes.io/component: storage
    app.kubernetes.io/managed-by: cdi-controller
    app.kubernetes.io/part-of: hyperconverged-cluster
    cdi.kubevirt.io: ''
spec:
  cloneStrategy: snapshot
EOF
```

### Example LVMCluster Deployment Using Labels

Below is a sample `LVMCluster` that I use in my lab (just to show you a full example).

```
oc apply -f - <<EOF
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: lvmcluster
  namespace: openshift-storage
spec:
  storage:
    deviceClasses:
```

```

- deviceSelector:
  paths:
    - '/dev/disk/by-path/pci-0000:03:00.0-scsi-0:2:1:0'
fstype: xfs
name: vg1
nodeSelector:
  nodeSelectorTerms:
    - matchExpressions:
      - key: topology.kubernetes.io/lvm-disk
        operator: In
        values:
          - sdb
thinPoolConfig:
  chunkSizeCalculationPolicy: Static
  name: thin-pool-1
  overprovisionRatio: 10
  sizePercent: 90

```

EOF

**IMPORTANT:** For this to work correctly you **MUST** use the following label!

```
NODE_NAME=$(oc get no -o name)
```

```
oc label $NODE_NAME topology.kubernetes.io/lvm-disk=sdb
```

### Useful Disk Tooling

If you want to explore your disk speed, and other information listed above (paths, devices, etc), you can use the following script that I've created (for SNO environments): [HERE](#)

## Step 2: Installation of OpenShift Virtualization Operator on v4.17.x

**IMPORTANT:** Be sure to complete [Installing LVM on SNO \(OpenShift v4.17.x\)](#) before starting this guide.

### Table of Contents

- [Installation](#)
- [Modifying the Storage Profile](#)

### Installation

7. Deploy the OpenShift Virtualization operator with the following command.

```

oc apply -f - <<EOF
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv

```

```

---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv

---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.17.1
  channel: "stable"
EOF

```

- Next, apply the following `HyperConverged` CR to your cluster to install OpenShift Virtualization.

```

oc apply -f - <<EOF
kind: HyperConverged
apiVersion: hco.kubevirt.io/v1beta1
metadata:
  annotations:
    deployOVS: 'false'
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec: {}
EOF

```

### Modifying the Storage Profile

**NOTICE:** The following information is only relevant if you've deployed [LVM Storage](#) (typically for SNO deployments). ODF does not require the following changes.

Have you noticed that VMs start slowly when using OCPV on SNO deployments? The following section will help!

Modifying the `StorageProfile` will improve the overall speed/performance of starting virtual machines on OCPV. By default, OpenShift Data Foundation (ODF) creates a `StorageProfile` with the `cloneStrategy` key set to `snapshot`, but the LVM

operator is set this value to `copy` by default. You will want to change this if/when using the LVM operator. The process is very simple, and will be covered below.

9. You will have one (1) **StorageProfile** for each **StorageClass** in your OpenShift cluster. You can view the **StorageProfile** objects with the following command. These objects are not namespaced, similar to **StorageClass** objects.

```
> oc get storageprofile
```

```
NAME                AGE
lvms-vg1            6d5h
lvms-vg1-immediate  6d5h
nfs-openshift      6d2h
```

```
> oc get storageclass
```

```
NAME                PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
lvms-vg1            topolvm.io           Delete          WaitForFirstConsumer
true                12d
lvms-vg1-immediate (default)  topolvm.io           Delete          Immediate
true                12d
nfs-openshift      nfs.csi.k8s.io       Delete          Immediate
false                6d2h
```

10. You can view the YAML for the **StorageProfile** by using the following command (we will use `lvms-vg1` as an example. Look for the `spec`, and

```
oc get storageprofile lvms-vg1-immediate -o yaml
```

It will likely look like the sample below. By default, you can see that the `cloneStrategy` is set to `copy` (look at the `status` section). We don't want this, and would prefer to set `snapshot`.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  creationTimestamp: "2024-08-30T15:16:03Z"
  generation: 3
  labels:
    app: containerized-data-importer
    app.kubernetes.io/component: storage
    app.kubernetes.io/managed-by: cdi-controller
    app.kubernetes.io/part-of: hyperconverged-cluster
    app.kubernetes.io/version: 4.16.1
    cdi.kubevirt.io: ""
  name: lvms-vg1-immediate
  ownerReferences:
  - apiVersion: cdi.kubevirt.io/v1beta1
    blockOwnerDeletion: true
```

```

    controller: true
    kind: CDI
    name: cdi-kubevirt-hyperconverged
    uid: 2825a572-ca0c-4fa2-bf30-9f05b0100e92
  resourceVersion: "13429408"
  uid: 94530d4b-99f1-4616-add9-b078264a8c05
spec: {}
status:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce
    volumeMode: Block
  - accessModes:
    - ReadWriteOnce
    volumeMode: Filesystem
  cloneStrategy: copy
  dataImportCronSourceFormat: pvc
  provisioner: topolvm.io
  snapshotClass: lvms-vg1
  storageClass: lvms-vg1-immediate

```

11. If your `StorageProfile` looks like the one above (set to `copy` vs `snapshot`), then you can use the following patch command to change the `cloneStrategy`.

**IMPORTANT:** *The following command will NOT\* work if you didn't first follow the instructions for installing the LVM StorageClass exactly like I suggested [HERE](#)\**

```
oc patch storageprofile lvms-vg1-immediate -p '{"spec":{"cloneStrategy": "snapshot"}}' --type=merge
```

### Sample Virtual Machine

Below is a sample virtual machine that can be used during a proof of concept (PoC), which puts together a few concepts that have been used in several other sections.

```

---
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora-devel-001
  namespace: jinkit-vms
  labels:
    app: fedora-devel-001
    kubevirt.io/dynamic-credentials-support: 'true'
    vm.kubevirt.io/template: fedora-server-small
    vm.kubevirt.io/template.namespace: openshift
    vm.kubevirt.io/template.revision: '1'

```

```

vm.kubevirt.io/template.version: v0.31.1
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      creationTimestamp: null
      name: fedora-devel-001
    spec:
      sourceRef:
        kind: DataSource
        name: fedora
        namespace: openshift-virtualization-os-images
      storage:
        resources:
          requests:
            storage: 75Gi
  running: true
  template:
    metadata:
      annotations:
        vm.kubevirt.io/flavor: small
        vm.kubevirt.io/os: fedora
        vm.kubevirt.io/workload: server
      creationTimestamp: null
      labels:
        kubevirt.io/domain: fedora-devel-001
        kubevirt.io/size: small
        network.kubevirt.io/headlessService: headless
    spec:
      accessCredentials:
      - sshPublicKey:
          propagationMethod:
            noCloud: {}
          source:
            secret:
              secretName: ssh-user-bjozsa
      architecture: amd64
      domain:
        cpu:
          cores: 1
          sockets: 4
          threads: 1
        devices:
          autoattachPodInterface: false

```

```

disks:
  - disk:
      bus: virtio
      name: rootdisk
  - disk:
      bus: virtio
      name: cloudinitdisk
interfaces:
  - bridge: {}
      macAddress: '02:f4:20:00:00:0e'
      model: virtio
      name: fedora-devel-001-eth0
logSerialConsole: false
rng: {}
features:
  acpi: {}
  smm:
      enabled: true
firmware:
  bootloader:
      efi: {}
machine:
  type: pc-q35-rhel9.4.0
memory:
  guest: 8Gi
resources: {}
networks:
  - multus:
      networkName: v0004-ens8f1-access
      name: fedora-devel-001-eth0
terminationGracePeriodSeconds: 180
volumes:
  - dataVolume:
      name: fedora-devel-001
      name: rootdisk
  - cloudInitNoCloud:
      networkData: |
      version: 2
      ethernet:
        enp1s0:
          addresses:
            - 192.168.6.35/22
          gateway4: 192.168.4.1
          nameservers:
            search:

```

```

- jinkit.com
addresses:
- 192.168.3.5
ntp:
servers:
- 0.pool.ntp.org
userData: |
#cloud-config
user: fedora
password: fedora
chpasswd:
  expire: false
ssh_authorized_keys:
- ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCADE1F7Fz3MGg0zst9h/2+5/pbeqCFFHlfaS0Iu4Bhsr7RenaTdzVpbT+9WpSr
rjdxDK9P3KProPwY2njgIt0EgfJ06MnRLE9dQDz0UIQ8caIH7o1zxy60dblonP5A82EuVUnZ0IGmAWSzUWsKef793tW
jLRx127eS1Bn8zbiI+m91Q8ypkLYSB9MMxQehupfzNzJpjvFA5dncZ2S7C8TFIPFtwBe9ITEb+w2phWvAE0SRjU3rLX
wCOWHT+7NRwkFfhK/moalPGDIyMjATPOJrtKKQtzSdyHeh9WyKOjJu8tXiM/4jFpOYmg/aMJeGr0/9fdxPe+zPismC/
FaLuv00ACgJ5b13tIfwD020fB2J4+qXtTz2geJVirxzkoo/6cKtb1cN/JjrYjwhfXR/dTehY59srgmQ5V1hzbUx1e4l
Ms+yZ78Xrf2Q0+7BikKJsy4CDHqvRdcLlpRq1pe3R9oODRdoFZhkKWywFCpi52ioR4CVbc/tCewzMzNSKZ/3P00ItBi
5IA5ex23dEVO/Mz1uyPrjgVx/U2N8J6yo900zX/Gftv/e3RKwGIUPpqZpzIUH/N0deTtpoSIaL5t8Ki8d3eZuiLZJY5
gan7tKUWDAL0JvJK+EEzs1YziBh91Dx1Yit0YeD+ztq/j0l0S8d0G3Q9Bhwk1ILT6PuBI2nAEOS0Q==
bjozsa@redhat.com
write_files:
- path: /etc/environment
  content: |
    http_proxy=http://proxy.example.com:3128
    https_proxy=http://proxy.example.com:3128
    no_proxy=localhost,127.0.0.1,.example.com,192.168.0.0/16
runcmd:
- echo "export http_proxy=http://proxy.example.com:3128" >>
/etc/profile.d/proxy.sh
- echo "export https_proxy=http://proxy.example.com:3128" >>
/etc/profile.d/proxy.sh
- echo "export no_proxy=localhost,127.0.0.1,.example.com,192.168.0.0/16" >>
/etc/profile.d/proxy.sh
name: cloudinitdisk


```

### Step 3: Performance Profiles for SNO (OpenShift 4.17.x)

**CRITICAL: DO NOT APPLY THE MANIFEST BELOW UNLESS YOU KNOW WHAT YOU'RE DOING!**

You will need to edit the performance profile sample (below) according to your own environment! Read and understand what you are applying first.

So, while a lot of my other gists include installations and configuration of operators, this one is a little different. There is no operator to install, because the `PerformanceProfile` object is already defined with OpenShift installations now. The



information below is just an example of a `PerformanceProfile`; some very specific options are used in my deployments. You only need this applied, if you're enabling SR-IOV (which some applications/workloads require). Use a performance profile that is recommended for the workloads you intend/need to run.

## Details

For better details on configuring a `PerformanceProfile`, visit the following [Red Hat documentation](#) for custom-tuning your system.

Additionally, you will find the following [GitHub document](#) useful in explaining each of the sections (as shown in the example below).

## Working Example

Below is a sample of a real world scenario. Do not apply this in your own environment, but read through the link above for further details.

```
oc apply -f - <<EOF
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance-sno
spec:
  additionalKernelArgs:
    - audit=0
    - mce=off
    - nmi_watchdog=0
    - efi=runtime
    - intel_iommu=on
  cpu:
    isolated: 2-93
    reserved: 0-1,94-95
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 8
      node: 0
      size: 1G
    - count: 8
      node: 1
      size: 1G
  numa:
    topologyPolicy: best-effort
  net:
    userLevelNetworking: true
  realTimeKernel:
    enabled: false
```

```
nodeSelector:
  node-role.kubernetes.io/master: ""
machineConfigPoolSelector:
  machineconfiguration.openshift.io/role: master
```

EOF

## Step 4: NMState Operator

### Installation: NMState Operator (OpenShift 4.17.x)

12. It is recommended to install the [OpenShift NMState Operator](#) when installing the [SR-IOV Operator](#), and this is primarily so that administrators can manage/configure NICs using a declarative `nmccli` model. Additionally, administrators will notice that the NMState operator will add additional WebUI dashboards under the **Network** navigation pane (NetworkAttachmentDefinitions, NodeNetworkConfigurationPolicy, and NodeNetworkState). `**NodeNetworkState*` is extremely useful in determining NIC topology and other information.

Although this is *optional*, I would recommend installing the [NMState operator](#) by deploying the following manifests.

```
oc apply -f - <<EOF
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-nmstate
    name: openshift-nmstate
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "true"
    name: openshift-nmstate
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: NMState.v1.nmstate.io
  name: openshift-nmstate
  namespace: openshift-nmstate
spec:
  targetNamespaces:
  - openshift-nmstate
  upgradeStrategy: Default
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```

metadata:
  labels:
    operators.coreos.com/kubernetes-nmstate-operator.openshift-nmstate: ""
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kubernetes-nmstate-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

```

- Now you want to configure a **default** NMState CR, with no further configuration. This will configure the operator to deploy an NMState management environment, which we can use *later* to configure interfaces *if we need to*.

```

oc apply -f - <<EOF
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
spec: {}
EOF

```

- After the NMState operator has been successfully installed, you will get a pop-up message informing you to refresh the webpage. When this happens, click the blue hyperlink in the pop-up window to perform this refresh. You can manually refresh the page as well, if that's what you prefer.

### Changing Network Settings with NMState

- Run the following interface script from this [LINK](#). Follow the instructions carefully. You will have output that looks similar to the following example below.

```

Starting pod/roderika-debug-jqzb6 ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.3.99
If you don't see a command prompt, try pressing enter.

```

Interface	IP Address	MAC Address	Link State	Link Speed	MTU
Vendor	Device ID	Driver	Total VFs	Configured VFs	PCIe Address
eno1np0	-	bnxt_en	8	0	0000:18:00.0
14e4	16d8	bnxt_en	8	0	0000:18:00.1
eno2np1	-	bnxt_en	8	0	0000:18:00.1
14e4	16d8	bnxt_en	8	0	0000:18:00.1
enp0s20f0u14u3	169.254.1.2	cdc_ether	b0:7b:25:d5:99:0d	UNKNOWN	425Mb/s 1500
-	cdc_ether	-	-	usb-0000:00:14.0-14.3	-
ens1f0	-	ice	6c:fe:54:5a:32:78	DOWN	1500
8086	1592	ice	128	0	0000:5e:00.0
ens1f1	-	ice	6c:fe:54:5a:32:79	DOWN	1500

```

8086      1592      ice          128          0          0000:5e:00.1
ens8f0    -          -          40:a6:b7:46:87:40 UP          10000Mb/s 9000
8086      158b      i40e         64           0          0000:b1:00.0
ens8f1    -          -          40:a6:b7:46:87:41 UP          10000Mb/s 9000
8086      158b      i40e         64           0          0000:b1:00.1

```

Removing debug pod ...

In the case of the example below, the interfaces I really want to focus on are `ens1f0`.

```

ens1f0    -          -          6c:fe:54:5a:32:78 DOWN        -          1500
8086      1592      ice          128          0          0000:5e:00.0
ens1f1    -          -          6c:fe:54:5a:32:79 DOWN        -          1500
8086      1592      ice          128          0          0000:5e:00.1

```

This is an [Intel E810-CQDA2 \(100G x2\)](#) NIC. I can tell this because it uses the [ICE](#) driver. But if you look at some other details about this NIC, you will notice that there's no link speed and that the MTU is set to 1500. We will make these changes in the following steps.

16. Please have a look at the following *example*. We are applying the following `NodeNetworkConfigurationPolicy` to simply enable and configure MTU settings for the [Intel E810-CQDA2 \(100G x2\)](#) NIC. This manifest looks very similar to an `NMStateConfig` object. [See the differences down below](#)

NOTE: Your environment will be different! DO NOT BLINDLY COPY AND PASTE!!!

```

oc apply -f - <<EOF
---
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: e810-policy-roderika
spec:
  nodeSelector:
    kubernetes.io/hostname: "roderika"
  desiredState:
    interfaces:
      - name: ens1f0
        description: Standard Ethernet via E810-100G NIC port 0
        type: ethernet
        state: up
        mtu: 9000
        ipv4:
          enabled: false
        ipv6:
          enabled: false
      - name: ens1f1
        description: Standard Ethernet via E810-100G NIC port 1

```

```
type: ethernet
state: up
mtu: 9000
ipv4:
  enabled: false
ipv6:
  enabled: false
```

EOF

#### Verification of NNCP

You can verify that the NNCP was configured correctly by looking at the YAML output of the object, either on the CLI or via the webUI (for this, go to **Home > API Explorer** and search for **NodeNetworkConfigurationPolicy**). In the output, look at the status field. For our example, you want to see the following result: "message": "1/1 nodes successfully configured", which means that the deployment was successful.

Example: shell `oc get nncp e810-policy-roderika -o jsonpath="{.status.conditions}" | jq .`

```
Results: json [ { "lastHeartbeatTime": "2024-09-06T13:42:04Z",
"lastTransitionTime": "2024-09-06T13:42:04Z", "message": "1/1 nodes successfully
configured", "reason": "SuccessfullyConfigured", "status": "True",
"type": "Available" }, { "lastHeartbeatTime": "2024-09-06T13:42:04Z",
"lastTransitionTime": "2024-09-06T13:42:04Z", "reason": "SuccessfullyConfigured",
"status": "False", "type": "Degraded" }, { "lastHeartbeatTime":
"2024-09-06T13:42:04Z", "lastTransitionTime": "2024-09-06T13:42:04Z",
"reason": "ConfigurationProgressing", "status": "False", "type":
"Progressing" } ]
```

You can also use the following `jq` example to verify that the MTU has changed on the interfaces `ens1f0` and `ens1f1`.

Example:

```
oc get NodeNetworkState roderika -o json | jq '.status.currentState.interfaces[] |
select(.name == "ens1f0" or .name == "ens1f1") | {name, mtu}'
```

Results:

```
{
  "name": "ens1f0",
  "mtu": 9000
}
{
  "name": "ens1f1",
  "mtu": 9000
}
```

## Special Notes about NNCP


One key very important and major between an `NMStateConfig`, which you are likely already familiar with, and a `NodeNetworkConfigurationPolicy`(NNCP) is that the `NodeNetworkConfigurationPolicy` is *asymmetric* in nature. Applying changes like described above will work perfectly fine, however if you need to remove the changes (due to a misconfiguration, etc), you will need to intentionally reverse the changes in your NNCP and reapply them. Deleting NNCP deployments will **NOT** revert the changes back to their original settings.

17. For example, if I wanted to adjust the MTU from our applied changes above (with a value of 9000) to an MTU of 1500 (the original values), simply *deleting* the NNCP will not work. I would need to apply the following manifest to *reverse* the changes.

```
oc apply -f - <<EOF
---
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: e810-policy-roderika
spec:
  nodeSelector:
    kubernetes.io/hostname: "roderika"
  desiredState:
    interfaces:
      - name: ens1f0
        type: ethernet
        state: down
        mtu: 1500
        ipv4:
          enabled: false
        ipv6:
          enabled: false
      - name: ens1f1
        type: ethernet
        state: down
        mtu: 1500
        ipv4:
          enabled: false
        ipv6:
          enabled: false
EOF
```

18. Then you will need to wait until the changes are applied successfully (like our previous steps above). You can loop the previous command if you'd prefer.

Looping the status field for the NNCP:



```
while sleep 1; do oc get nncp e810-policy-roderika -o jsonpath="{.status.conditions}" | jq . ; done
```

This means that it is still progressing:

```
[
  {
    "lastHeartbeatTime": "2024-09-06T14:46:01Z",
    "lastTransitionTime": "2024-09-06T14:46:01Z",
    "status": "Unknown",
    "type": "Available"
  },
  {
    "lastHeartbeatTime": "2024-09-06T14:46:01Z",
    "lastTransitionTime": "2024-09-06T14:46:01Z",
    "status": "Unknown",
    "type": "Degraded"
  },
  {
    "lastHeartbeatTime": "2024-09-06T14:46:01Z",
    "lastTransitionTime": "2024-09-06T14:46:01Z",
    "status": "Unknown",
    "type": "Progressing"
  }
]
```

This means that it was applied successfully:

```
[
  {
    "lastHeartbeatTime": "2024-09-06T14:52:57Z",
    "lastTransitionTime": "2024-09-06T14:52:57Z",
    "message": "1/1 nodes successfully configured",
    "reason": "SuccessfullyConfigured",
    "status": "True",
    "type": "Available"
  },
  {
    "lastHeartbeatTime": "2024-09-06T14:52:57Z",
    "lastTransitionTime": "2024-09-06T14:52:57Z",
    "reason": "SuccessfullyConfigured",
    "status": "False",
    "type": "Degraded"
  },
  {
    "lastHeartbeatTime": "2024-09-06T14:52:57Z",
```

```
    "lastTransitionTime": "2024-09-06T14:52:57Z",
    "reason": "ConfigurationProgressing",
    "status": "False",
    "type": "Progressing"
  }
]
```

19. Now you can successfully remove the NNCP object entirely.

```
> oc get nncp
NAME                                STATUS      REASON
e810-policy-roderika               Available   SuccessfullyConfigured

> oc delete nncp e810-policy-roderika
nodenetworkconfigurationpolicy.nmstate.io "e810-policy-roderika" deleted
```

## Step 5 Installing SR-IOV

### Part I: Prerequisites for SR-IOV Operator (OpenShift 4.16.x)

20. Gather details about attached network interfaces. You can run this command “as-is” to return extremely useful information that we will reference several times throughout this and other guides. There’s an example of what should be returned below.

- Use the following link to gather network interface information (using `oc debug` commands). Be sure that your `KUBECONFIG` variable has been set first, and that you have debug privileges to the cluster (common for the `kubeadmin` user). These instructions and example output can be found [HERE](#).

**IMPORTANT:** You must validate the hardware (NICs) you’re using with the [SR-IOV compatibility list](#). There are options/methods to use SR-IOV operator with unsupported NICs, but this needs to be worked on directly with Red Hat.

Required: Deploy a Performance Profile

21. Follow the instructions for deploying a `PerformanceProfile`.

Optional: Install NMState Operator (OpenShift 4.17)

22. It is at this point where I usually recommend that you deploy the [NMState Operator](#) (it’s optional, but highly recommended).

### Part II: Install SI-IOV Operator (OpenShift 4.17.x)

23. Apply the following `Namespace`, `OperatorGroup`, and `Subscription`. This will install the [SR-IOV Operator](#). In later steps, you will deploy a couple of CRs which will configure the operator and thus, configure your SR-IOV-based NICs.

```
oc apply -f - <<EOF
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
```

```

openshift.io/cluster-monitoring: "true"
pod-security.kubernetes.io/audit: privileged
pod-security.kubernetes.io/warn: privileged
security.openshift.io/scc.podSecurityLabelSync: "true"
name: openshift-sriov-network-operator

```

---

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
  upgradeStrategy: Default

```

---

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/sriov-network-operator.openshift-sriov-network-operator: ""
  name: sriov-network-operator
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sriov-network-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

EOF

24. Now we need to install an `SriovOperatorConfig`, which is going to tell the operator important things like logging levels, how you want to treat reboots, etc. This should work for PoCs, but you want to pay attention to one option (`disableDrain`) to make sure you're *ok with the node rebooting*. If you need to really lock down reboots (i.e. similar to production environments), then change the setting accordingly.

```

oc apply -f - <<EOF
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: true

```

```
enableOperatorWebhook: true
disableDrain: true
logLevel: 2
EOF
```

## SR-IOV Deployment Overview

With the two operators out of the way, and configured at a global level, it's time to take the information that you completed in **Step 1** and apply them here; to the deployment. Deployments for SR-IOV-based resources are *namespaced* and that means that you can give a specific number of VFs to a given namespace. Namespaces can have their own resources. This is important when creating your deployments. I will use an example deployment, but you will need to edit some information to personalize these deployments. I will help you identify the changes needed as we go along.

1. Label the node(s) with the following label. This is required.

```
NODE_NAME=$(oc get no -o name)
```

```
oc label $NODE_NAME feature.node.kubernetes.io/network-sriov.capable=true
```

NOTE: \*The command above assumes that this is for a SNO deployment, however if you need this to be applied for a non-SNO deployment, simply replace the `$NODE_NAME` with the name of the node you want to enable SR-IOV on, like `node/<node-name>`.

2. With the nodes labeled, you will need to create the following manifest/deployment using the NIC details from our command earlier.

```
INTERFACE=ens8f0
VF_RANGE=0-31
VF_TOTAL=64
VLAN=3
VENDOR=8086
DEVICE_ID=158b
DEVICE_TYPE=vfio-pci
RDMA=false
PRIORITY=99
NAME=policy-sriov-$INTERFACE-0031
NAMESPACE=jinkit-vms
RESOURCE=${INTERFACE}_0031

oc apply -f - <<EOF | tee /dev/tty
---
```

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: $NAME
  namespace: openshift-sriov-network-operator
```

```
spec:
  deviceType: $DEVICE_TYPE
  isRdma: $RDMA
  nicSelector:
    deviceID: "$DEVICE_ID"
    vendor: "$VENDOR"
    pfNames:
      - $INTERFACE#$VF_RANGE
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: $VF_TOTAL
  priority: $PRIORITY
  resourceName: $RESOURCE
```

```
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: $INTERFACE-vlan3-d
  namespace: openshift-sriov-network-operator
spec:
  ipam: |-
    {
      "ipam": {
        "type": "dhcp"
      }
    }
  networkNamespace: $NAMESPACE
  resourceName: $RESOURCE
  vlan: $VLAN
EOF
```

### Example SR-IOV Deployment

```
oc apply -f - <<EOF
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: panos-vms
  name: panos-vms
spec: {}
---
apiVersion: sriovnetwork.openshift.io/v1
```

```
kind: SriovNetworkNodePolicy
metadata:
  name: policy-sriov-mcx4-enp4s0f1np1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  isRdma: false
  nicSelector:
    vendor: 15b3
    deviceID: '1015'
    pfNames:
      - enp4s0f1np1#0-5
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 6
  priority: 97
  resourceName: mcx4_enp4s0f1np1_003
```

```
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: panos-vms-vlan3-d
  namespace: openshift-sriov-network-operator
spec:
  ipam: |-
    {
      "ipam": {
        "type": "dhcp"
      }
    }
  logLevel: info
  networkNamespace: panos-vms
  resourceName: mcx4_enp4s0f1np1_003
  vlan: 3
```

```
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: panos-vms-vlan3-w
  namespace: openshift-sriov-network-operator
spec:
  ipam: |-
    {
```

```
"type": "whereabouts",
"range": "192.168.4.0/24",
"range_start": "192.168.4.160",
"range_end": "192.168.4.170",
"routes": [{"dst": "192.168.0.0/16", "gw": "192.168.4.1"}],
"dns": {"nameservers": ["8.8.8.8"]}
}
logLevel: info
networkNamespace: panos-vms
resourceName: mcx4_enp4s0f1np1_003
vlan: 3
EOF
```

## Installation Procedures

1. Now you can follow the instructions for setting up the Palo Alto VM-Series on OpenShift Virtualization. The documentation are located [HERE](#)

## Palo Alto Networks Configuration

### Start a VM in OpenShift Virtualization

To access the original GitHub Gist click [HERE](#). The goal of the Gist is to get a Palo Alto VM running on OpenShift Virtualization. These steps assumed you have performed the following steps first:

- You have set up OpenShift Storage, or some other CSI-based storage option that is supported from [OpenShift Virtualization](#).
- You have [SR-IOV supported device](#) NICs available and configured within your OpenShift cluster ([more on this below](#)).
- A private registry is required for PanOS/Panorama images (QCOW2 images stored in an OCI-compliant container image format). This guide will provide detailed instructions on how to use a free private registry on [Red Hat's Quay.io](#) ([below](#)); however, you can use your own private registry if you'd like. Please make sure you have completed all of the foregoing steps before proceeding to deploy the virtualized Palo Alto Networks NGFW.

### Create a New RHEL 8/9 or Fedora VM on OpenShift Virtualization

The goal of this document is to walk you through your [newly created OpenShift Virtualization environment](#) for engineering and management. Start a new RHEL or Fedora virtual machine within OpenShift Virtualization.

- Where you see (Fedora) at the beginning of a **Step** number below, you will use your Fedora or RHEL virtual machine in OCPV.

- Where you see a (Workstation) at the beginning of a **Step** number below, you will use your own PC or workstation. Please note that I am using a MacBook Pro for my *Workstation*, so some of my commands may vary just slightly from your own (although I tried to take some of this into account within my instructions below).

## Part I: Install Prerequisites

25. (Fedora) Update the system and install Git and ansible

```
sudo yum update -yy && sudo yum install -yy git podman tree
```

## Part II: Configuration Prerequisites

1. (Fedora) Set the global user/email for Git

```
git config --global user.name "Brandon B. Jozsa"
git config --global user.email "bjozsa@jinkit.com"
```

2. (Fedora) Create the following directories that we will use throughout this project

```
mkdir -p
{~/development/dockerfiles/panorama/,~/development/dockerfiles/panos/,~/development/p
anos-images/,~/Downloads/,~/ .kube/}
```


3. (Fedora) Download the follow Dockerfiles for Panorama and PanOS:

```
curl -L
https://gist.githubusercontent.com/v1k0d3n/ffbba6fd2908ecb1b30b87780ff6d8b1/raw/9938d
64e682bb9e84948d906c7d6704e21a63dc7/Dockerfile-panos -o
~/development/dockerfiles/panos/Dockerfile
```

```
curl -L
https://gist.githubusercontent.com/v1k0d3n/ffbba6fd2908ecb1b30b87780ff6d8b1/raw/9938d
64e682bb9e84948d906c7d6704e21a63dc7/Dockerfile-panorama -o
~/development/dockerfiles/panorama/Dockerfile
```

4. (Fedora) Make sure that the Fedora host has the **oc** and **virtctl** Linux binaries. First make sure that your Fedora environment has a `~/ .kube/` directory and then copy the **kubeconfig** into that directory, but as named **config**. If this is confusing, please review [this documentation](#).
  - **Details:** Copy the downloaded **kubeconfig** from your cluster and name it `~/ .kube/config` on the Fedora host.
5. (Fedora) Get the latest **oc** Linux binary for OpenShift:

```
curl -L
https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable-4.16/openshift-clien
t-linux-4.16.10.tar.gz -o ~/Downloads/openshift-client-linux-4.16.10.tar.gz
```



6. (Workstation) Download the `virtctl` binary for Linux by going to the OpenShift console and clicking on the top right-hand corner on the ? (question mark), and clicking on Command Line Tools. Download the one that says Download `virtctl` for Linux for `x86_64`.

7. (Workstation) Set a variable on your workstation that will make life easier later when we copy files from your workstation to your Fedora VM. You want to put YOUR FEDORA IP ADDRESS as the variable below:

```
primary_ip="PUT_YOUR_IP_HERE_AND_KEEP_QUOTES"
```

8. (Workstation > Fedora) Use `scp` or some other utility to the `virtctl.tar.gz` tarball to the Fedora virtual machine.

STOP: Change the information below to match your own environment. This includes the `$USER` and the variable you set earlier (`$primary_ip`), because these will be different for each environment.

```
scp ~/Downloads/virtctl.tar.gz
fedora@$primary_ip:/home/fedora/Downloads/virtctl.tar.gz
```

9. (Fedora) Back on the Fedora virtual machine, unpack both of these packages and copy the executables to `/usr/local/bin/`

```
# Go to the ~/Downloads folder, and untar the binaries for oc, kubectl, and virtctl:
cd ~/Downloads/
tar zxvf openshift-client-linux-4.16.10.tar.gz
tar zxvf virtctl.tar.gz
```

```
# Copy the binaries to /usr/local/bin/:
sudo cp virtctl /usr/local/bin/
sudo cp oc /usr/local/bin/
sudo cp kubectl /usr/local/bin/
```

10. (Fedora) Check that you can communicate with OpenShift:

```
[fedora@fedora-panos-devel-3 Downloads]$ oc get nodes
NAME          STATUS    ROLES                                AGE      VERSION
roderika     Ready    control-plane,master,worker        5d23h   v1.29.7+4510e9c
```

## Part III: Run the PanOS-Bootstrapper Utility (Optional)

26. (Fedora) Run the following podman command:

```
sudo podman run -d --network=host paloaltonetworks/panos-bootstrapper:latest
```

27. (Fedora) Get the IP address of your server, and then create following variable with that IP where the `PUT_YOUR_IP_HERE_AND_KEEP_QUOTES` notation is below:

IMPORTANT: Get the IP address of your Fedora VM first with the following command:

```
ip addr
```

When you find the IP address you need, enter it as a variable (`primary_ip`).



```
primary_ip="PUT_YOUR_IP_HERE_AND_KEEP_QUOTES"
```

28. (Fedora) Change into the following directory called `panos-images`:

```
cd ~/development/panos-images/
```

29. (Fedora) Look at the following sample. Configure your firewall variables to match your own deployment/environment:

**IMPORTANT:** The `auth_code` will be the authorization code for your Palo Alto VM-series firewall license. You may need to check your email for this code.

```
# Set variables to be used in the curl command below:
```

```
hostname="panos-01"
management_ip="192.168.3.105/24"
management_gateway="192.168.3.1"
dns_servers='["192.168.3.5", "192.168.3.5"]'
auth_code="REMOVED"
panorama_ip="192.168.3.95"
panorama_port="3978"
timezone="UTC"
```

```
# Use the following code to generate an ISO that will be used to configure your
firewall (ZTP):
```

```
curl -X POST http://$primary_ip:5000/bootstrap_kvm \
-H "Content-Type: application/json" \
-d '{
  "hostname": "$hostname",
  "archive_type": "iso",
  "management_ip": "$management_ip",
  "management_gateway": "$management_gateway",
  "dns_servers": "$dns_servers",
  "auth_code": "$auth_code",
  "panorama_ip": "$panorama_ip",
  "panorama_port": "$panorama_port",
  "timezone": "$timezone"
}' --output /home/fedora/development/panos-images/panos-bootstrap-$hostname.iso
```

30. (Fedora) Now go back to your CLI that is communicating with OpenShift Virtualization, and type the following commands. BE SURE to change with your own variables for `namespace`:

```
hostname="panos-01"
namespace="jinkit-vms"
```

```
# Be sure to use virtctl to create the ISO/pvc first:
```

```
virtctl image-upload pvc panos-bootstrap-$hostname-config-iso \
--size 1Gi \
--image-path=/home/fedora/development/panos-images/panos-bootstrap-$hostname.iso \
```

```
--namespace=$namespace \  
--force-bind \  
--volume-mode=block \  
--insecure
```

## Part IV: Stage PanOS Prerequisites

31. (Workstation) Download the PanOS and Panorama update from the [Palo Alto website \(11.2.0\)](#)

For this you will need to go to the following navigation path:

- PAN-OS: **Updates > Software Updates > PAN-OS for VM-Series KVM Base Images** and select **PA-VM-KVM-11.2.0.qcow2**.
  - Panorama: **Updates > Software Updates > Panorama Base Images** and select **Panorama-KVM-11.2.0.qcow2**.
32. (Workstation > Fedora) SCP The files from your local machine to the Fedora Development VM (these are example directories, so be sure to change for your environment):

**Important:** Be sure to use your own Workstation *PATH* and *\$USER*. The syntax is for example only, and will **not\*** match your environment exactly as shown below:\*

```
scp ~/Downloads/PA-VM-KVM-11.2.0.qcow2  
fedora@192.168.3.249:/home/fedora/development/dockerfiles/panos/PA-VM-KVM-11.2.0.qcow  
2
```

```
scp ~/Downloads/Panorama-KVM-11.2.0.qcow2  
fedora@192.168.3.249:/home/fedora/development/dockerfiles/panorama/Panorama-KVM-11.2.  
0.qcow2
```

## Part V: Build PanOS and Panorama Boot Images

1. Go to your registry and get your user credentials. I will show you how to do this for [Quay](#).
  - If you don't already have an account, create one.
  - Then when you are logged in, go to the very top right-hand corner and click on your username.
  - Click on **Account Settings** and on the left-hand menu click on the **Gear Icon**.
  - This pulls up a section called **\*\*Docked CLI Password\***.
  - For the **CLI Password** section click on **Generate Encrypted Password**.
  - It will ask you to verify your credentials. Enter them now.
  - This will now have an open window that says **Credentials for USER**.
  - On the left-hand side of this window, click on the **Podman Login** option.
  - Copy the podman login -u='USER' -p= string.
  - Paste this string into your Fedora CLI to authenticate Podman to your custom personal registry.

2. (Fedora) Run the following podman commands to build the correct containers. **MAKE SURE** to replace **USERNAME** with your actual username for [Quay](#).

**Important:** Set the following variable for your *QUAY USERNAME*.

```
quay_username=bjozsa-redhat
```

**Important:** Be sure to use your own Workstation *PATH* and *\$quay\_username* are set correctly. The syntax is for example only, and will likely not\* match your environment exactly as shown below:\*

```
cd /home/fedora/development/dockerfiles/panos/  
podman build -t quay.io/$quay_username/pa-vm:11.2.0 .
```

```
cd /home/fedora/development/dockerfiles/panorama/  
podman build -t quay.io/$quay_username/panorama:11.2.0 .
```

3. (Fedora) Now push these images to your Quay registry. Like the previous command, **MAKE SURE** to replace **USERNAME** with your actual username for [Quay](#).

**Important:** Be sure to use your own Workstation *PATH* and *\$USER*. The syntax is for example only, and will not\* match your environment exactly as shown below:\*

```
podman push quay.io/$quay_username/pa-vm:11.2.0  
podman push quay.io/$quay_username/panorama:11.2.0
```

4. (Fedora) Now create a secret for your private registry (where you pushed the images). By default, Quay will make your container images private. **DO NOT EXPOSE PALO ALTO IMAGES WITHOUT AUTHORIZATION!** In order to keep these images private but allow the VM to download what it needs, we need to create a secret for your registry.

*# Change the namespace to your own namespace:*

```
secret_name="quay-creds-registry-panos"  
namespace="jinkit-vms"
```

*# Do not change below this point:*

```
read -p "Enter Username: " username  
read -sp "Enter Password or Token: " password  
echo # This ensures a newline after password input
```

```
encoded_username=$(printf "%s" "$username" | base64 -w 0)  
encoded_password=$(printf "%s" "$password" | base64 -w 0)
```

```
oc apply -f - <<EOF  
apiVersion: v1  
kind: Secret  
metadata:  
  labels:  
    app: containerized-data-importer  
  name: $secret_name
```

```
namespace: $namespace
type: Opaque
data:
  accessKeyId: "$encoded_username"
  secretKey: "$encoded_password"
EOF
```

## Part VI: Configure SR-IOV

1. (Fedora) Make sure that you have the [SR-IOV operator](#) installed, and that you have configured two [SRIOVNetworks](#). Think of one network as the “trusted” network, and one network as the “untrusted” network. I will show you a couple of examples down below.

[INSERT SCREEN SHOOT]

## Part VII: Creating the Panorama VM

33. (Fedora) Before creating the PanOS VM-Series Firewall, first create the Panorama management instance (named `panorama-01`). Use the following deployment below simply as an example. Complete the variables according to your own environment, and the deployment should “just work” as-is:

```
# Variables
namespace=jinkit-vms
instance_name=panorama-01
secret_name="quay-creds-registry-panos"
nad_mgmt01=ens6f1np1-vlan3-d
nad_mgmt02=ens6f1np1-vlan3-d
mac_mgmt01="02:9b:48:00:18:51"
mac_mgmt02="02:9b:48:00:18:52"
registry_image="docker://quay.io/bjzsa-redhat/panorama:11.2.0"
disk_size="150Gi"
core_count="16"
memory_count="36G"

# Virtual Machine Deployment
oc apply -f - <<EOF
---
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: $instance_name
  namespace: $namespace
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
```

```

metadata:
  creationTimestamp: null
  name: $instance_name
spec:
  source:
    registry:
      secretRef: $secret_name
      url: "$registry_image"
    storage:
      resources:
        requests:
          storage: $disk_size
running: false
template:
  metadata:
    labels:
      kubevirt.io/size: small
      kubevirt.io/domain: $instance_name
  spec:
    domain:
      cpu:
        cores: $core_count
      devices:
        disks:
          - name: rootdisk
            disk:
              bus: virtio
        interfaces:
          - name: $instance_name-port1
            macAddress: $mac_mgmt01
            model: virtio
            sriov: {}
          - name: $instance_name-port2
            macAddress: $mac_mgmt02
            model: virtio
            sriov: {}
      resources:
        requests:
          memory: $memory_count
    networks:
      - name: $instance_name-port1
        multus:
          networkName: $nad_mgmt01
      - name: $instance_name-port2
        multus:

```

```

networkName: $nad_mgmt02
volumes:
  - name: rootdisk
    dataVolume:
      name: $instance_name

```

EOF

34. (OpenShift Console) In the OpenShift console, navigate to the VM you've just created (check the variables above, but if left default the name of the vm should be called **panorama-01**.
  - Click on the **Configuration** tab.
  - Next, click on the **Storage** tab.
  - Click on the blue box that says **Add disk**
  - In the new window (Add disk), use the **Name** storage-disk-01
  - **Source** should be Empty disk (blank)
  - **PersistentVolumeClaim size** should be 75 GiB or higher
  - **Type** should be set to Disk
  - **Interface** should be set to VirtIO
  - **StorageClass** should be set to lvms-vg1-immediate
  - Make sure the option **Apply optimized StorageProfile settings** is checked
  - Click the blue **Save** button when completed
35. Start the VM (you will notice that the VM was created earlier, but not started per the `running: false` option).
36. Complete the setup of Panorama as you would normally.

## Part VIII: Creating the PanOS VM

37. (Fedora) Now you can deploy the first PanOS VM-Series Firewall (named **panos-01**). Use the following deployment below simply as an example. Complete the variables according to your own environment, and the deployment should “just work” as-is - just like the Panorama instance above:

```

# Variables
namespace=jinkit-vmns
instance_name=panos-01
secret_name="quay-creds-registry-panos"
nad_mgmt01=ens6f1np1-vlan3-d
nad_mgmt02=ens6f1np1-vlan3-d
nad_trusted=ens6f1np1-vlan60-d
nad_untrusted=ens6f1np1-vlan90-d
mac_mgmt01="02:9b:48:00:18:11"
mac_mgmt02="02:9b:48:00:18:12"
mac_trusted="02:9b:48:00:18:13"
mac_untrusted="02:9b:48:00:18:14"
registry_image="docker://quay.io/bjzsa-redhat/pa-vm:11.2.0"
disk_size="80Gi"

```

```

core_count="8"
memory_count="16384M"

# Virtual Machine Deployment
oc apply -f - <<EOF
---
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: $instance_name
  namespace: $namespace
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      creationTimestamp: null
      name: $instance_name
    spec:
      source:
        registry:
          secretRef: $secret_name
          url: "$registry_image"
      storage:
        resources:
          requests:
            storage: $disk_size
  running: true
  template:
    metadata:
      labels:
        kubevirt.io/size: small
        kubevirt.io/domain: $instance_name
    spec:
      domain:
        cpu:
          cores: $core_count
        devices:
          disks:
            - name: rootdisk
              disk:
                bus: virtio
            - name: $instance_name-config-iso
              cdrom:
                bus: sata

```

```

interfaces:
  - name: $instance_name-port1
    macAddress: $mac_mgmt01
    model: virtio
    sriov: {}
  - name: $instance_name-port2
    macAddress: $mac_mgmt02
    model: virtio
    sriov: {}
  - name: $instance_name-port3
    macAddress: $mac_trusted
    model: virtio
    sriov: {}
  - name: $instance_name-port4
    macAddress: $mac_untrusted
    model: virtio
    sriov: {}
resources:
  requests:
    memory: $memory_count
networks:
  - name: $instance_name-port1
    multus:
      networkName: $nad_mgmt01
  - name: $instance_name-port2
    multus:
      networkName: $nad_mgmt02
  - name: $instance_name-port3
    multus:
      networkName: $nad_trusted
  - name: $instance_name-port4
    multus:
      networkName: $nad_untrusted
volumes:
  - name: rootdisk
    dataVolume:
      name: $instance_name
  - name: $instance_name-config-iso
    persistentVolumeClaim:
      claimName: panos-bootstrap-$instance_name-config-iso

```

EOF





## Troubleshooting

**NOTE:** Use cases that do not match the use case(s) as documented in this integration guide, using a version of PAN-OS or a version of the partner product not listed as tested and validated are **out of the scope** of the integration as documented by this integration guide. Any additional use cases or variation from those use cases documented in this integration guide are **out of the scope** of this integration guide document. It is not outside the realm of possibility that unanticipated issues (i.e. scalability, concurrent API session limits, interoperability, other incompatibilities, etc.) could be encountered if **out-of-scope** use cases for this integration guide document are deployed. Therefore, after familiarizing yourself with the use cases documented in this integration guide, if there are plans to deploy use cases that are **out-of-scope** for this integration guide, it is highly recommended that the initial deployment be performed in a pilot/proof-of-concept environment prior to deployment within production.

### Common troubleshooting steps

- ...

### Helpful Resources

#### Red Hat:

- [OpenShift Container Platform Tech Docs](#)
- [OpenShift Virtualization Tech Docs](#)
- [OpenShift Virtualization Release Notes](#)
- [Red Hat Customer Support](#)

#### Palo Alto Networks:

- [TechDocs Home - PAN-OS](#)
- [Palo Alto Networks Customer Support](#)
- [TechDocs Home - VM-Series](#)



## Contact Information for Support

### For Red Hat specific issues:

- [Red Hat Support Portal](#)

### For Palo Alto Networks specific issues:

- [Palo Alto Networks Live Community](#)
- [Palo Alto Networks Customer Support](#)



## Technical Details

- If applicable, list the names of API calls that are being leveraged to enable this integration. If none simply say so.
- If this is a syslog integration, list out the types of log(s) being used (traffic, threat, HIP Match, config, system, endpoint agent logs, etc.).
- List out any additional technical details on how the two technologies integrate.

**Preparing the Integration Guide (IG) and Joint Solution Brief (JSB). When your team begins to use the provided template(s) to prepare the Integration Guide(IG) and Joint Solution Brief (JSB), in the section of the template showing the generalized integration diagram, please ensure that you depict the Palo Alto Networks ML-Powered NGFW and/or Panorama (if applicable) using the approved iconology and the labeling below:**

Obviously it is ok to reduce the size of the icons and the font in order to accommodate available space on the generalized integration diagram as applicable. Only include Palo Alto Networks iconology that is part of your integration. **(NOTE: Once the IG DRAFT is completed, this page should be deleted).**



ML-Powered NGFW



Panorama



VM-Series



GlobalProtect