NOKIA

VIRTUALIZED SERVICE ROUTER

VSR INSTALLATION AND SETUP GUIDE RELEASE 20.2.R1

3HE 15837 AAAA TQZZA 01

Issue: 01

February 2020

Table of Contents

1 Getting Started

1.1 About This Guide

This guide describes how to install and set up the Nokia Virtualized Service Router (VSR).

The VSR software is designed to run on x86 virtual machines (VMs) deployed on industry standard Intel servers. The following applications (network functions) are supported by VSR software:

- Application Assurance (AA)
- BGP Route Reflection (RR)
- Broadband Network Gateway (BNG)
- Data Center Gateway (DCGW)
- L2TP Network Server (LNS)
- MAP-T Border Relay (MAP-T BR)
- Network Address Translation (NAT)
- Provide Edge (PE)
- Security Gateway (SeGW)
- WLAN Gateway (WLAN-GW)
- Virtualized Residential Gateway (vRGW)

Command outputs shown in this guide are examples only; actual outputs may differ depending on supported functionality and user configuration.

This guide is organized into functional chapters and includes:

- a functional overview of the VSR and a description of the VSR system architecture
- general requirements for the NFV infrastructure (NFVI) supporting VSR VMs
- procedures to instantiate VSR VMs on KVM compute hosts using libvirt or OpenStack
- procedures to instantiate VSR VMs on VMware ESXi compute hosts using vCloud Director or the vSphere Web client
- an overview of lifecycle management of VSR VMs by the CloudBand Application Manager (CBAM)
- troubleshooting procedures for common VSR problems

Note: This guide generically covers Release 20.*x*.R*x*. content and may contain some content that will be released in later maintenance loads. Not all information described in the SR OS guides listed in the *VSR Documentation Suite Overview* is applicable to the VSR. Please refer to the SR OS 20.*x*.R*x*. Software Release Notes, part number 3HE 16194 000*x* TQZZA, for information about features supported in each load of the Release 20.*x*.R*x*. software.

Note: For information about the virtualized 7750 SR and 7950 XRS simulator (vSIM), refer to the *vSIM Installation and Setup Guide*.

1.1.1   Audience

This guide is intended for network administrators who are responsible for the initial setup of the VSRs on a standard NFVI. It is assumed that the network administrators have an understanding of the following:

- x86 hardware architecture
- Linux system installation, configuration, and administration methods
- VMware basics
- OpenStack basics
- basic XML syntax
- SR OS CLI
- networking principles and configurations

1.1.2   VSR and SR Technical Publications

After the installation process of the required VSR product is completed, refer to the VSR and SR technical publications as listed in the *VSR Documentation Suite Overview Card*, part number 3HE 15075 AAA*x* TQZZA. These documents contain information about the software configuration and the command line interface (CLI) that is used to configure network parameters and services.

1.2   VSR Installation and Setup

This guide is presented in an overall logical configuration flow. Each section describes the tasks for a functional area.

Table 1 lists the general tasks and procedures necessary to install and set up a Nokia Virtualized Service Router (VSR) VM on either a Linux KVM or VMware ESXi host machine in the recommended order of execution.

Table 1   **VSR Installation and Configuration Workflow**

Task	Description	See
Installing the host machine	Set up and install the host machine, including the host operating system.	• Host OS and Hypervisor
Installing the virtualization packages	Install the necessary virtualization packages on the host machine.	• Linux KVM Compute Hosts • VMware ESXi
Optimizing BIOS and host OS	Optimize BIOS and host OS settings for VSR deployment.	• BIOS Settings • Kernel Parameters • Optimize BIOS and Host Settings
Configuring host networking	Configure host networking (NICs, network interfaces, vSwitch).	• VSR Networking • Linux vSwitch Implementations • Data Center Networking • Host Devices and PCI Passthrough • Network Interfaces • Guest vNIC Mapping in VSR VMs
Downloading the software image	Download the SR OS software image.	• VSR Software Packaging
Obtaining the license keys	Obtain the software license keys from Nokia.	• VSR Software Licensing
VM resource requirements	Determine resource requirements for the VM.	• Memory • Guest Memory Backing • vCPU
Creating configuration files	If required, create configuration files for the VM. The exact format of the configuration files depends on the method of installation.	• Deploying VSR on Linux KVM Hosts Using Libvirt or OpenStack

Table 1   **VSR Installation and Configuration Workflow (Continued)**

Task	Description	See
Launching the VM	Launch the VSR VM.	• Deploying VSR on Linux KVM Hosts Using Libvirt or OpenStack • Deploying VSR-I on VMware ESXi Hosts
Verifying the installation	Verify the VSR VM installation.	• Verifying VSR Installation on Linux KVM Hosts
Configuring VMs	Configure VM parameters as necessary.	• Virtual Machine Configuration Parameters
Managing the Lifecycle of the VM	Customize the VSR-I template package for a specific NFVI environment and manage the lifecycle of a VSR-I instance using CBAM.	• VSR-I Lifecycle Management Using CBAM

2   VSR Overview

2.1   VSR Overview

The Nokia Virtualized Service Router (VSR) is a carrier grade network function virtualization (NFV) platform based on the industry-leading SR OS software that powers the 7750 SR and 7950 XRS routers. NFV enables network functions that previously depended on custom hardware to be deployed on commodity hardware using standard IT virtualization technologies. For network operators, the benefits of NFV include:

- reduced CAPEX by using industry-standard hardware that is potentially easier to upgrade
- reduced OPEX (space, power, cooling) by consolidation of multiple functions on fewer physical platforms
- faster and simpler testing and rollout of new services
- more flexibility to scale capacity up or down, as needed
- ability to move or add network functions to a location without necessarily needing new equipment

2.1.1   VSR Concept

The VSR software is designed for a standard virtualization environment in which the hypervisor software running on a host machine (compute server) creates and manages one or more VMs that consume a subset of the host machine resources. Each VM is an abstraction of a physical machine with its own CPU, memory, storage, and interconnect devices. Each VSR system forms a Virtual Network Function (VNF) running an x86-optimized version of the SR OS software, and made up of one or more VNF components (VNF-C) spanning one or more compute servers.

In virtualization terms, SR OS is the guest operating system of each VSR VM. VSR VMs can be deployed in combination with other VMs on the same server, including VMs that run guest operating systems other than SR OS.

Note: Care must be taken not to over-subscribe host resources; VSR VMs must have dedicated CPU cores and dedicated vRAM memory to ensure good stability and performance. In addition, combining VSR VMs with other VMs that have intensive memory access requirements on the same CPU socket should be generally avoided for performance and stability reasons.

Figure 1 shows the general concept of a VSR.

Figure 1   **VSR Concept**
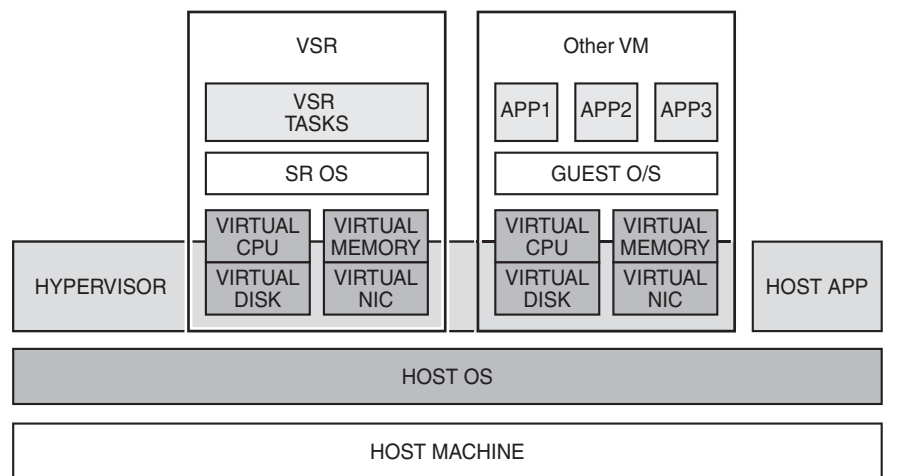
The host machine supporting a VSR VM is a generalized compute x86-based server such as the Nokia Airframe line of servers.

The host machine must run the Linux KVM or ESXi hypervisor software, which support all VSR applications and are compatible with the VSR software.

See NFV Infrastructure Requirements for detailed information about the minimum requirements of the host server and the supported hypervisors for the VSR.

2.2 VSR Network Functions

VSR currently supports the following network functions:

- BGP Route Reflector
- Broadband Network Gateway
- L2TP Network Server
- Network Address Translation
- MAP-T Border Relay
- Provider Edge Router
- Data Center Gateway
- Security Gateway
- Application Assurance
- WLAN Gateway
- Virtualized Residential Gateway

The VSR is a single product consisting of a single software image. The NFV functions described in this section can be deployed in isolation or in combination with each other on one VSR system. The allowed functions are determined by the software licenses described in VSR Software Licensing.

2.2.1 BGP Route Reflector

When configured as a BGP Route Reflector (RR), a VSR system supports the following features:

- IGP protocols
- static routes
- iBGP client-to-client, client-to-nonclient, and nonclient-to-nonclient reflection
- multi-protocol BGP (all address families supported by SR OS)
- route policies
- disable route table install
- convergence optimizations with SMP
- optimal route reflection

When used as a dedicated RR, the VSR must be deployed as an integrated model; see Integrated Model for more information.

2.2.2   Broadband Network Gateway

When configured as a Broadband Network Gateway (BNG), a VSR system supports the following features:

- routed CO model of Enhanced Subscriber Management (ESM) on numbered subscriber-interface and group-interface
- dual-stack IPoE subscriber management (DHCPv4/v6, SLAAC)
- dual-stack PPPoE sessions
- Static SAP and MSAP
- 1:1 and N:1 VLANs
- managed routes (IPv4/IPv6)
- subscriber authentication using LUDB and RADIUS
- dynamic QoS overrides
- dynamic filter overrides
- accounting per session/host/SPI
- LAG for subscriber access
- HTTP redirect
- H-QoS for subscriber hosts
- subscriber LI using UDP or GRE encapsulation
- data-triggered SAPs and ESM hosts (stateless redundancy)
- L2TP LAC
- credit control by way of category-maps

2.2.3   L2TP Network Server

When configured as a L2TP Network Server (LNS), a VSR system provides L2TP tunnel functionality and subscriber management features. The following features are supported:

- L2TP tunneling
- L2TP tunnel accounting
- IPv4 and IPv6 subscriber management
- subscriber accounting
- VPRN support

2.2.4   Network Address Translation

When configured for Network Address Translation (NAT), a VSR system supports
key features including:

- LSN44 (deterministic and non-deterministic)
- NAT64
- DS-Lite
- L2-Aware NAT in conjunction with BNG functionality
- UPnP for L2-aware NAT
- LI for NAT

2.2.5   MAP-T Border Relay

When configured as a MAP-T Border Relay, a VSR system supports the following
features:

- hub-and-Spoke model
- hull routing support (IS-IS, BGP, OSPF, RIP)
- upstream MAP-T anti-spoof
- multiple MAP-T domains in the same routing context
- upstream and downstream fragmentation
- MSS adjust and MTU support per domain
- forward/drop statistics collection per domain
    - fragmentation statistics
    - logging

2.2.6   Provider Edge Router

When configured as a Provider Edge router (PE), a VSR system can deliver Layer-
2 and Layer-3 VPNs, as well as Internet access.

As a PE router, the VSR supports the following features:

- IPv4 and IPv6 routing protocols and routing policies: Static, RIP, RIPng,
  OSPFv2, OSPFv3, BGP, MP-BGP, IS-IS

- IPv4 and IPv6 multicast protocols: IGMP, MLD, PIM, and MSDP
- Ethernet Virtual Private Wire Services (VPWS)
- Ethernet Virtual Private LAN Services (VPLS), including routed VPLS (R-VPLS)
- IPv4 and IPv6 unicast VPNs (RFC 4364), including inter-AS models A and B
- IPv6 over IPv4 MPLS LSPs (6PE)
- point-to-point MPLS LSPs, signaled using LDP, RSVP, or BGP
- segment routing
- OAM tools: ping, trace, BFD
- IPv4 and IPv6 interface filters (ingress and egress)
- QoS classification
- ingress and egress traffic policing, including support for H-Pol
- egress traffic queuing and scheduling
- traffic mirroring to and from SAP and spoke-SDP interfaces
- Network Group Encryption (NGE) for SDPs, VPRNs, and router interfaces

2.2.7   Data Center Gateway

When configured as a Data Center Gateway (DCGW), a VSR system can interconnect EVPN-signaled VPNs in the data center to WAN services.

As a DCGW, the VSR supports the following features:

- VLL and VPLS services using BGP-EVPN signaling and VXLAN/IPv4 transport
- VLL and VPLS services using BGP-EVPN signaling and MPLS transport (LDP, RSVP, MPLSoUDP)
- VLL service using static VXLAN/IPv4 transport
- Nuage VSD integration using XMPP (fully dynamic model and static-dynamic model)

2.2.8   Security Gateway

When configured as a Security Gateway (SeGW) and with the application of appropriate software licenses (ASLs), a VSR system supports the following features:

- IKEv1 static/dynamic LAN-to-LAN tunnel with pre-shared key authentication
- IKEv1 remote-access tunnel with plain-xauth-psk authentication

- IKEv2 static/dynamic LAN-to-LAN tunnel and remote-access tunnel
- IKEv2 tunnel authentication method: psk/psk-radius/cert-auth/cert-radius/eap/ autoeap/auto-eap-radius
- IKEv2 remote-access tunnel internal address assignment methods: RADIUS local address pool/external DHCPv4/v6 server
- encryption algorithm: DES/3DES/AES
- authentication algorithm: MD5/SHA1/SHA256/SHA384/SHA512
- Diffie-Hellman Group: 1/2/5/14/15
- Perfect Forward Secrecy
- NAT-T support
- IPv4 and IPv6 support
- multi-chassis (using a SAP or network interface to shunt traffic)
- IKEv2 fragmentation
    - client lockout
    - auto CRL update
    - TCP MSS Adjust

2.2.9   Application Assurance

The Application Assurance (AA) SR OS feature set is enabled on the VSR with appropriate software licenses (ASLs). The AA is offered as an enhancement option to the VSR roles of PE, BNG, LNS, and SeGW.

When the AA feature set is enabled, the VSR uses AA (on a base PE configuration) to allow deployment as a standalone transit-AA DPI VNF. The following AA features are supported by the VSR in the current release:

- AA use cases
    - DNS-Ip-Cache
    - AA policers (all types)
    - AA http-redirect
    - http-notification (IBN)
    - http-enrichment
    - url-filter url-list for local URL filtering
- AA group partition features
    - AA radius-accounting
    - AA event-log (syslog export)

- – stateful FW (**gtp**, **gtp-filter**, **sctp-filter**, **session-filter**)
- – AA policy application/AG/ASO/app-filter, signatures
- – AA policy charging-groups
- – AA transit-ip (IPv4/IPv6)
- – AA transit prefix lists
- statistics
  - – All AA XML and IPfix stats records export
- AA Hi/Lo resource watermark alarms
- AARP - local protection
- AA support on the following services:
  - – Epipe SAP and spoke-SDP (MPLS+GRE)
  - – VPLS SAP and spoke-SDP
  - – IES/VPRN SAP
  - – IES/VPRN ESM (in BNG or WLGW application)
  - – IES/VPRN IPSec Private SAP (in PE or SeGW application)
  - – AA tunnel support (DS-Lite, 6RD/6to4, Teredo, GRE)
- AA signatures configuration-only upgrade without VSR system upgrade

2.2.10   WLAN Gateway

When configured as a WLAN Gateway (WLGW), a VSR system supports the following features:

- access over soft-GRE, soft-L2TPv3, and L2-AP
- dual-stack sessions (DSM and ESM)
- central and distributed RADIUS-Proxy for EAP
- L2-aware NAT
- HTTP-redirect (vFP and ISA based)
- migrant user support
- data-triggered mobility
- data-triggered UE creation (IPv4 ESM, IPv4/IPv6 DSM)
- L2-wholesale
- control plane triggered mobility
- inter WLAN-GW redundancy

2.2.11   Virtualized Residential Gateway

When configured as a Virtualized Residential Gateway (vRGW), a VSR system supports the following features:

- access over Soft-GRE, soft-L2TPv3 and L2-AP using WLAN-GW group interfaces
  - per-home DHCP pool allocation, with support for sticky and static hosts
  - implicit and explicit per home (BRG) authentication
  - L2-aware NAT with UPnP support and IPv6 SLAAC/IA_NA support
  - data-triggered host creation (IPv4)

2.3   VSR Deployment Models

The VSR can be deployed only as an integrated model (VSR-I). In releases prior to Release 19.5.R1, VSR could also be deployed as a distributed model (VSR-D).

2.3.1   Integrated Model

The integrated VSR model uses a single VM to represent a network element. All functions and processing tasks of a network element, including control, management and data plane are performed by the resources of the single VM.

An integrated VSR model supports vertical scale-up or scale-down (by adding or removing VM resources). Increase the scale of an integrated system by adding virtual CPUs and virtual memory to the VM. However, such changes require the VM to be shut down and restarted.

It may be necessary to deploy multiple integrated model VSR systems to achieve the necessary scale or redundancy for a specific application.

From a configuration perspective, an integrated VSR is modeled as a chassis with one slot. The slot is equipped with the cpm-v card type that maps one-to-one with the VM. The VSR shows a chassis type of VSR-I.

When a cpm-v card is installed in the VSR-I chassis, it loads the **both.tim** software image, which presents the view that the VSR-I system has two slots:

- an "A" slot running control plane and management tasks
- a "1" slot running datapath tasks

In a VSR-I system, Slot "1" has four MDA slots that are numbered 1 through 4. Each MDA slot can be equipped with any of the supported VSR MDAs described in MDA Types subject to the following limitations:

- maximum of four 20-port I/O MDAs
- maximum of one virtualized ISA-AA
- maximum of one virtualized ISA-BB
- maximum of one virtualized ISA-tunnel

Figure 2 shows the general concept of a VSR integrated model. In this example, one server is running a hypervisor that supports three VMs; two of these VMs are allocated to VSR-I instances and the VMs appear as two distinct systems externally.

Figure 2   **VSR Integrated Model**

VM 1 VSR-I system N1 Guest O/S = SR OS vCPUs = 4 vMem = 4GB	VM 2 VSR-I system N2 Guest O/S = SR OS vCPUs = 6 vMem = 8GB	VM 3 Guest O/S = Other vCPUs = 1 vMem = 2GB
Hypervisor		
Host O/S		
Server		
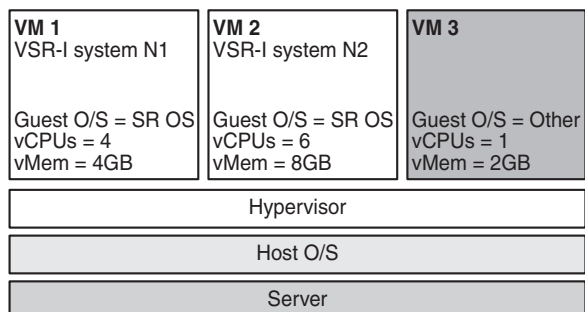
25777

2.4 VSR Card and MDA Types

This section describes the types of cards and MDAs modeled by the VSR.

2.4.1 Card Types

The VSR supports the **cpm-v** card type. This is the only allowed card type in a VSR-I system. When it is installed, the card loads the **both.tim** image and executes control, management, and datapath tasks. The cpm-v card provides the VSR-I system with 4 MDA slots in a virtual slot 1. The MDA slots can be configured as described in Integrated Model.

2.4.2 MDA Types

The following MDA types are supported in a VSR-I system.

- **m20-v**

  The m20-v MDA provides 20 I/O ports. Each port is configurable as a network, access, or hybrid port from the SR OS perspective.

  Each port of the m20-v maps to one vNIC interface of the VSR-I. The default speed of each m20-v MDA port (from the SR OS guest perspective) is 40 Gb/s; however, the speed can be changed through **config**>**port**>**ethernet**>**speed**. Allowable values are 1 Gb/s, 10 Gb/s, 25 Gb/s, 40 Gb/s, 50 Gb/s and 100 Gb/s. Auto-negotiation of the speed with the remote end is not supported. The operational speed affects only the rate of egress traffic on the port, not the rate of ingress traffic. A maximum of four m20-v MDAs are supported on a VSR-I.

- **isa-aa-v**

  The isa-aa-v MDA adds Application Assurance processing capabilities to the VSR-I.

  A maximum of one isa-aa-v MDA is supported by the VSR-I.

- **isa-bb-v**

  The isa-bb-v MDA adds NAT, LNS, WLGW and other processing capabilities to the VSR-I.

  A maximum of one isa-bb-v MDA is supported by VSR-I.

  WLGW features require the isa-bb-v MDA to be configured in the mda/1 slot of the VSR-I.

- **isa-tunnel-v**

The isa-tunnel-v MDA adds IPSec and IP/GRE tunnel termination capabilities to the VSR.

A maximum of one isa-tunnel-v MDA is supported by the VSR-I.

2.5   VSR Architecture

The VSR architecture is similar to the physical Nokia SR-series routers. Table 2 summarizes the similarities and important differences between a physical SR-series router and a VSR system.

➡ **Note:** The VSR VMs support both control plane processing and datapath functions; this behavior is similar to the CPM and IOM functions supported by a single card in some physical SR routers.

Table 2   **Function Comparison between SR-series Routers and VSR**

Function in SR-series router	Function in VSR
CPMs manage the system in an SR-series router and run control plane protocols. SMP distributes the workload over multiple CPU cores, if available.	The VMs manage the VSR system and run control plane protocols such as OSPF and BGP. SMP distributes the workload over multiple CPU cores, if available.
The CPMs of some SR-series routers use P-chips or Q-chips, which allow support of specific features at higher scale or performance.	The VSR does not have an equivalent of P-chips or Q-chips. Specific features (for example, centralized BFD) are supported with lower scale and performance, or not supported on the VSR (for example, CPM filtering, and Ethernet OAM). Refer to the *SR OS 20.x.Rx. Software Release Notes* for a complete list of supported features.
CPMs download configuration and state information to the IOMs. The messaging is reliable and uses the fabric.	The VMs download configuration and state information.
The IOMs support local control plane (LCP) software tasks to accomplish the following: • programming the fastpath (for example, FIB updates) • collecting and sending statistics • controlling ISA applications	The VSR supports LCP software tasks to accomplish the following: • programming the fastpath (for example, FIB updates) • collecting and sending statistics • controlling ISA applications
The P network processor and Q queuing chips of the IOM (or equivalent card type) handle fastpath forwarding and packet queuing.	The VSR supports software tasks that handle fastpath forwarding and packet queuing. See Virtual Forwarding Path for information about the VSR software tasks.

Table 2　　　　**Function Comparison between SR-series Routers and VSR**

Function in SR-series router	Function in VSR
Packet forwarding fastpath is implemented by the P-chip and Q-chip of the IOM, and the ISA function fastpath is implemented by software running on the ISA hardware.	The VSR supports ISA-related processing in-line in the fastpath. In other words, ISA-related processing is performed as part of the packet-forwarding pipeline.

2.5.1　Virtual Forwarding Path

The forwarding-related fastpath functions on the hardware-based SR-series routers are implemented by the P-chip and the Q-chip. The P-chip network processor handles functions such as filtering, classification, policing, header field lookups, and manipulation. The Q-chip handles real-time queuing and scheduling decisions. The P-chip and Q-chip are programmed by the LCP control code that runs on the general-purpose CPU of the IOM.

Because the VSR does not have a P-chip or Q chip, the forwarding-related fastpath functions are implemented in software. The virtual forwarding path (vFP) of the VSR is a rewrite of the P-chip and Q-chip code to take advantage of x86 CPU instructions and memory architecture. The vFP uses the same programming APIs as the P-chip and Q-chip to maximize feature portability.

2.5.2　Control and Management Plane

2.5.2.1　Allocation of vCPUs for Control and Management Tasks

The number of vCPUs available for control and management plane tasks (such as OSPF, BGP, and SNMP) depends on the system configuration.

In a VSR-I system, the number of cores available for both CPM control and IOM control functions usually defaults to one. In cases where an ISA is installed and the VM has a total of three or more vCPUs, the default number of control cores is two.

The default number of control cores does not provide sufficient computational power for some VSR applications. More vCPUs can be reserved for the control plane by using the **control-cpu-cores** SMBIOS parameter. See Sysinfo for more information about **control-cpu-cores** and other SMBIOS parameters.

2.6   VSR Networking

A VSR VM can have one or more virtual NIC ports. Depending on the hypervisor, each VSR vNIC port can be configured to use one of the following virtualized I/O technologies:

- SR-IOV, supported with both Linux KVM and VMware ESXi
- PCI passthrough, supported with both Linux KVM and VMware ESXi
- VirtIO, supported with Linux KVM only
- VMXNET3, supported with VMware ESXi only
- E1000, supported with both Linux KVM and VMware ESXi

In the VirtIO, VMXNET3, and E1000 models, the virtual NIC port is internally connected to a logical interface within the host. The logical host interface may map directly to a physical NIC port/VLAN or it may connect to a vSwitch within the host. If a vNIC port is connected to a vSwitch, a physical NIC port/VLAN must be added as a bridge port of the vSwitch to enable traffic to reach other hosts.

In the SR-IOV and PCI passthrough models, the guest directly connects its virtual NIC interface to a host PCI device corresponding to an entire physical NIC port or a slice of a physical NIC port (SR-IOV virtual function). This mostly bypasses the hypervisor and host OS networking stack and enables very fast data transfer with the help of Intel Virtualization Technology for Directed I/O (VT-d) or I/O Memory Management Unit (IOMMU) technology.

The number of virtual NIC ports supported by each VSR VM and the constraints on the virtualized I/O model for each vNIC port depend on the VSR VM type and hypervisor.

In a VSR-I, the VM must be assigned a minimum of 1 and a maximum of either 10 vNIC ports (with VMware ESXi) or 16 vNIC ports (with Linux KVM). The first vNIC port (by lowest PCI bus/device/function address in the guest) must use either VirtIO or E1000 depending on whether the hypervisor in KVM or VMware ESXi. The remaining vNIC ports can be any combination of the supported technologies for the hypervisor.

2.7 VSR Software Packaging

The VSR software is available for download from OLCS as a ZIP file with a name such as Nokia-VSR-VM-20.*x*.zip. The ZIP archive file contains two OVA archive files and a QCOW2 disk image file.

The sros-vsr.ova archive file is used for onboarding a VSR-I VM into a VMware environment. This OVA contains an OVF descriptor file and a VMDK disk image containing the VSR software. The OVA file can be used to instantiate a VSR-I VM using either vCloud Director or the vSphere Web Client interacting with a vCenter Server.

➡ **Note:** Do not use the sros-vm.ova file. This OVA should only be used for vSIM (virtualized 7750 SR and 7950 XRS simulator) deployments. Refer to the *vSIM Installation and Setup Guide* for more information.

➡ **Note:** TIMOS content is created under the cf3:/TIMOS directory. This location cannot be edited.

The sros-vm.qcow2 disk image should be used when creating VSR-I VMs on Linux KVM machines using libvirt or OpenStack.

3HE 15837 AAAA TQZZA 01

3 NFV Infrastructure Requirements

3.1 Overview

This chapter describes the network functions virtualization (NFV) infrastructure that must be in place to support VSR VMs. The NFV infrastructure includes compute servers (host machines), storage solutions, networking devices, and the software that runs on these components to support virtualization.

The NFV infrastructure is typically deployed in a data center and may be managed by a cloud management platform such as OpenStack, but this is not required.

Note: For recommendations about VSR deployment in an OpenStack environment, see Deploying VSR on Linux KVM Hosts Using Libvirt or OpenStack.

Information about other cloud management platforms is beyond the scope of this guide.

3.2 Compute Server Hardware Requirements

This section describes the compute server hardware requirements.

3.2.1 CPU and DRAM

Nokia recommends the deployment of VSR VMs using Airframe servers. However, VSR VMs can be deployed on any server that is powered by one or two of the following CPU models:

- Intel Xeon E5-26xx-v2 (Intel Ivy Bridge)
- Intel Xeon E5-26xx-v3 (Intel Haswell)
- Intel Xeon E5-26xx-v4 (Intel Broadwell-EP)
- Intel Xeon 5xxx/6xxx/8xxx Gold or Platinum (Intel Skylake-SP)

The server should be equipped with sufficient DRAM memory to meet the memory requirement of the host and have adequate resources to back the memory of each guest VM without oversubscription. See Memory and Guest Memory Backing for more information.

➡️ **Note:** VSR deployment is not supported on servers powered by AMD or ARM CPUs.

3.2.2 Intel QuickAssist Support

VSR supports IPsec fastpath offloading using Intel QuickAssist (QAT) hardware. The system automatically detects and utilizes the hardware if it is made available to the VSR; when enabled, the system uses the QAT hardware for ESP packet encryption or decryption.

VSR supports following QAT hardware on KVM hypervisor via SR-IOV:

- Intel PCH chipset C627/C628
- Intel QuickAssist Adapter 8970

Using QAT offloading is optional; if QAT is not provisioned, VSR uses CPU for IPsec fastpath processing.

3.2.3   Storage

Under normal circumstances (without extensive storage of log and accounting files), each VSR VM needs less than 10 Gbytes of persistent storage in total across all virtual disks (CF1, CF2 and CF3).

Each virtual disk allocated to a VSR VM must be backed by either a disk image stored on the local host machine or a network-attached block storage device. Each virtual disk must be made to appear to the VSR VM as an IDE hard drive.

Note: CF3 has a maximum of 1.2 GB, even if the volume size is increased. CF1 or CF2, which can be of larger sizes, must be used to add more storage.

3.2.4   NICs

When the VSR VM uses a VirtIO, E1000, or VMXNET3 driver for one of its vNIC interfaces (ports), the abstraction provided by the hypervisor allows any type of physical NIC to be used to transport the traffic associated with the vNIC interface.

To use the SR-IOV or PCI passthrough models, ensure that the physical NIC is supported by VSR software for the type of hypervisor that is used. Refer to the *SR OS 20.x.Rx* Software Release Notes for a list of compatible NICs.

3.2.4.1   Using SR-IOV

Single root I/O virtualization (SR-IOV) is a PCI-SIG standard that allows a single root function (a single physical Ethernet port) to appear as multiple separate physical devices; each device is associated with its own PCIe function called a Virtual Function (VF).

In an NFV host, the hypervisor can assign the VFs to VMs so that they appear as vNIC interfaces to the guests. SR-IOV enables almost bare-metal I/O performance because data is transferred directly using Direct Memory Access (DMA) between the NIC hardware and guest memory (with address translation provided by Intel VT-d).

To use SR-IOV, the following prerequisites apply.

- The physical NIC must support SR-IOV.
- Both SR-IOV and Intel VT-d must be enabled in the BIOS (on a Linux KVM host, use **dmesg** to check for **dmar: kernel** messages).

- The IOMMU must be enabled in the host OS.

  On a Linux KVM host, this requires **intel_iommu=on** and **iommu=pt** to be specified as kernel boot parameters (**iommu=pt** causes the DMA remapping to be bypassed in the Linux kernel, improving host performance).

- SR-IOV must be activated on Linux Kernel (Intel) or driver levels (Mellanox).

> **Note:** When SR-IOV is used with Ethernet NICs, the technology can be very restrictive about the types of Ethernet frames that can be delivered to the guest on receive or accepted from the guest on transmit.

The SR-IOV filtering rules implemented by the NIC depend on the hypervisor, the physical NIC model, its firmware revision and the version of the host driver software. All NICs should allow untagged Ethernet frames with a unicast MAC DA matching the guest vNIC interface MAC address to pass through. However, more advanced cases may not work. For example, in SR-IOV mode it may not be possible to send or accept:

- Ethernet frames with a multicast MAC DA
- tagged Ethernet frames with the VLAN tag added or removed by the guest
- Ethernet frames with a MAC DA not matching the guest vNIC interface MAC address (such as VRRP packets addressed to a virtual MAC address)

> **Note:** If the internal MAC address is changed in a VM, traffic may become unidirectional.

If any of these SR-IOV restrictions prove to be too limiting, consider PCI passthrough as an alternative technology.

3.2.4.1.1 MTU Configuration

For SR-IOV, all capabilities are closely linked to the NIC and driver. The SR-IOV Virtual Function (VF) inherits the MTU value from the Physical Function (PF) associated with the NIC port.

When setting the MTU, perform the following steps in the order shown.

> **Note:** Performing these steps in a different order or changing the MTU on the PF, where the VFs are already enabled, can result in issues that are difficult to troubleshoot. For example, traffic flow may become unidirectional, independent of the size of transported packets.

Step 1. Set the MTU on the PF (NIC).

Step 2. Configure the number of VFs on the PF.

Step 3. Start the VM.

3.2.4.1.2 Trusted Mode

Trusted mode can be enabled to allow a VM to change VF parameters, such as MTU and VLAN. By default, the VF is in untrusted mode.

Trusted mode of a VF can be configured using the **ip link set dev** *interface-name* **vf** *VF-number* **trust on** command for both Intel and Mellanox cards. Consider that this command is not persistent.

3.2.4.1.3 Setting up Linux KVM Hosts to Use SR-IOV with Intel NICs

To enable SR-IOV for supported Intel 10GE NICs on a Linux KVM host, perform the following steps.

1. Use the `sysfs` tool to set the number of VFs per physical port (PF). For example, to create 30 VFs on the physical port corresponding to eth1, enter the following command:

   ```
   echo 30 > /sys/class/net/eth1/device/sriov_numvfs
   ```

   The preceding `sysfs` configuration is not persistent across reboots on most Linux distributions.

2. To make the configuration persistent, run a script after each reboot. It is beyond the scope of this guide to describe all the scripting options, but some of the more common methods include using systemd, crontab, and rc.local, as described later in this section.

> **Note:** As there is no single script or configuration file where all network options, such as systemd, crontab, rc.local, or ifcfg-* are configured; these options could be used in a system at the same time and could conflict. Use as few tools as possible to avoid such conflict.

3. Use the **lspci** command to verify the creation of the VFs.

4. Prevent the host from binding its VF driver to the new VF devices as follows:

   - Create or modify the `/etc/modprobe.d/blacklist.conf` file.
   - Add the following lines to the `blacklist.conf` file:

     ```
     blacklist ixgbevf
     ```

Using systemd

The following is a sample script with three commands (any commands can be used).
This script is executed for three interfaces.

```
[root@vsr /]# more /root/activate_sriov.sh
#!/bin/bash
sriovinterfaces='ens6f0 em50 p1p1'
#------------- Tuning for selected interfaces
activate_sriov () {
    for i in $sriovinterfaces
        do
            #------- Configure SR-IOV
          echo 4 > "/sys/class/net/$i/device/sriov_numvfs"
            #------- Configure Rx/Tx Ring Parameters
            ethtool -G $i rx 4096 >/dev/null 2>&1
            ethtool -G $i tx 4096 >/dev/null 2>&1
            #------- Configure Tx Queue Length
            ip link set dev $i  txqueuelen 20000
        done
}
#------------- __MAIN__
activate_sriov
```

To use systemd, perform the following steps.

Step 1. Create a new systemd unit file in the `/etc/systemd/system`
directory.

Step 2. In the `[Service]` section of the systemd unit file, invoke a bash script that
loops through all the SR-IOV physical NIC ports and executes the `echo
<number-of-VFs>/sys/class/net/<physical-port>/device/
sriov_numvfs` command.

Step 3. Ensure that the script is executable.

```
chmod +x /root/activate_sriov.sh
```

Step 4. Enable the systemd service using the **systemctl enable** *new-service-
name* command. An example systemd unit file is shown below.

```
[root@vsr ~]# more /etc/systemd/system/autostart_nokia.service
[Unit]
Description=Autostart Service
After=network.target
After=libvirtd.service
[Service]
Type=oneshot
User=root
ExecStart=/usr/bin/bash /root/activate_sriov.sh
Restart=no
[Install]
WantedBy=multi-user.target
```

Step 5. Activate the systemd service.

```
systemctl enable autostart_nokia.service
```

```
Created symlink from /etc/systemd/system/multi-user.target.wants/
autostart_nokia.service to /etc/systemd/system/autostart_nokia.service.
```

Using crontab

Use `crontab` to execute a bash script at reboot. Add a line that calls a bash script at each reboot.

Using rc.local

Edit `/etc/rc.d/rc.local` to call a bash script. The bash script can loop through all the SR-IOV physical NIC ports and execute the `echo <number-of-VFs>/sys/class/net/<physical-port>/device/sriov_numvfs` command.

➡ **Note:** While using the `/etc/rc.local` script, which is executed at boot time, is a common solution, it is not recommended. If you open such file on your system, the following message may be displayed.

```
# It is highly advisable to create own systemd services or udev
rules# to run scripts during boot instead of using this file.
```

Using ifcfg-*

These ifcfg-* files can be edited to configure interfaces. These files are found in the `/etc/sysconfig/network-scripts/` directory.

3.2.4.1.4 Setting up Linux KVM Hosts to Use SR-IOV with Mellanox ConnectX-4 and ConnectX-5 NICs

To support SR-IOV with supported Mellanox ConnectX-4 or ConnectX-5 NICs installed in a Linux KVM host:

1. If necessary, download the latest MLNX_OFED software package from the Mellanox website. Run the `mlnxofedinstall` installation script using an appropriate set of options as guided by the Mellanox documentation.
2. Load the new driver as instructed by the installation script.

➡️ **Note:** Consider that the server may need to be rebooted during this step.

3. Run `mlxfwmanager` and record the PSID for the installed NIC card. This ID is needed to search for the appropriate firmware on the Mellanox web site.

4. Download the firmware version recommended by Nokia; refer to the *SR OS 20.x.Rx. Software Release Notes*. Install the new firmware using a command similar to the following:

```
mlxfwmanager_pci -i fw-ConnectX4-rel-12_16_1020-
0NHYP5_0XR0K2_Ax-FlexBoot-3.4.812.bin -u
```

5. Enable SR-IOV in the NIC firmware.

   i. Load Mellanox Software Tools by running the **mst start** command.

   ii. From the output of the preceding command, identify the correct device for the NIC port that you are configuring.

   For example, `/dev/mst/mt4115_pciconf0`.

   iii. Query the status of the identified device:

   ```
   mlxconfig -d /dev/mst/4115_pciconf0
   ```

   iv. Enable SR-IOV on this device and set the maximum number of VFs for this device in firmware by using the **mlxconfig** command. For example, to set a maximum of 16 VFs, enter:

   ```
   mlxconfig -d /dev/mst/4115_pciconf0 set SRIOV_EN=1
   NUM_OF_VFS=16
   ```

   v. To apply the new firmware settings, reset the card without reboot using the **mlxfwreset** command. For example:

   ```
   mlxfwreset --device /dev/mst/mt4115_pciconf0 reset
   ```

6. Enable SR-IOV in the MLNX_OFED driver.

   i. Set the number of VFs per PF by using the **sysfs** tool.

   For example, use one of the following commands to create VFs on the physical port:

   - `echo 16 > /sys/class/infiniband/mlx5_0/device/ sriov_numvfs`

     This command creates 16 VFs on the physical port corresponding to mlx5_0.

   - `echo 1 > /sys/class/net/ens6f0/device/sriov_numvfs`

     This command creates one VF on the physical port corresponding to ens6f0.

The preceding **sysfs** configuration is not persistent across reboots on most Linux distributions; see Setting up Linux KVM Hosts to Use SR-IOV with Intel NICs, for more information about making the configuration persistent.

    ii. Verify the creation of the VFs using the **lspci** command.

    iii. Verify the creation of the VFs using the **ip link show** command.

7. To bypass some of the SR-IOV restrictions mentioned in Using SR-IOV, it is highly recommended to enable the "trust" setting for each VF, as shown in the following examples:

- ```
  echo ON >/sys/bus/pci/devices/0000:03:00.0/sriov/1/
  trust
  ```

- ```
  ip link set dev ens6f0 vf 1 trust on
  ```

Otherwise, the incoming and outgoing frames may be blocked if they do not match the MAC address and VLAN ID specified in the configuration of the vNIC interface.

3.2.4.2 Using PCI Passthrough

PCI passthrough is a virtualization technology that allows a PCI device of the host to be assigned directly to a VM. When the assigned PCI device is a physical NIC port, the guest controls the port using its own equivalent of the bare-metal NIC driver. In "managed" mode, the PCI device is automatically detached from the host OS drivers when the guest is started, then re-attached when the guest shuts down. Alternatively, the host OS may be configured to blacklist the PCI devices used by the guest so that they never get attached to host OS drivers.

With PCI passthrough, the physical port is fully managed by a VM, and the host (hypervisor) is not involved. The MTU value of the physical port is overwritten after the VM starts. MTU can be checked in the VM using the **show port** *port-name* **detail** command.

→ **Note:** As the VM controls the hardware, there are no tools to determine statistics on a hypervisor level and as a result, SR OS debugging commands must be used.

To use PCI passthrough, the following requirements apply:

- Intel VT-d must be enabled in the BIOS. On a Linux KVM host, use **dmesg** to check for **dmar: kernel** messages.

- IOMMU must be enabled in the host OS. On a Linux KVM host, this requires the **intel_iommu=on** and **iommu=pt** to be specified as kernel boot parameters (**iommu=pt** causes DMA remapping to be bypassed in the Linux kernel, improving host performance).

3.3 Compute Server Software Requirements

This section describes the requirements for host OS and virtualization software that runs on compute servers to support VSR VMs.

3.3.1 BIOS Settings

The BIOS of the server is responsible for initializing the system and loading the OS. Each time the system boots, there is an opportunity for the user to change various BIOS settings. Some of these settings are important or even critical to the operation of the VSR VMs on the server.

Important BIOS settings on host machines that support VSR VMs are listed below. Some of these settings are mandatory and others are recommended for better performance.

➡ **Note:** Generic terms are used for the BIOS settings; actual parameter names may differ, depending on the manufacturer of the server.

The mandatory settings are:

- SR-IOV must be enabled if you plan to use this technology. The actual BIOS setting may be called "SR-IOV Global Enable", depending on your BIOS vendor.
- Intel VT-x must be enabled in all cases
- Intel VT-d must be enabled to use the SR-IOV or PCI passthrough functionality (see NICs for more information). The actual BIOS setting may be called "I/OAT DMA Engine" depending on your BIOS vendor.
- x2APIC must be enabled in all cases
- Non-Uniform Memory Access (NUMA) must be enabled if it is applicable to the host and if it is disabled by default in the BIOS

The following settings are highly recommended on compute hosts intended to have VSR VM with high packet-per-second forwarding requirements:

- disable the following:
  - hardware prefetcher
  - IO Non Posted Prefetching

This parameter is relevant to Intel Haswell-based hosts and onwards, and should be disabled on those systems. It is not exposed on all BIOS versions.

– adjacent cache line prefetching

– PCIe Active State Power Management (ASPM) support; on Linux KVM hosts, this can be disabled by a kernel boot parameter

– Advanced Configuration and Power Interface (ACPI) states:

• P-State

If enabled, the CPU (all cores on specific NUMA) enters "sleep" mode in case there is no activity. This mode is similar to C-State but for the whole NUMA node. In most cases, it saves power in idle times. However, for performance-oriented systems, when power consumption is not an issue, it is recommended that P-State is disabled.

• C-State

For energy saving, it is possible to lower the CPU power when it is idle. Each CPU has several power modes called "C-states" or "C-modes." Energy-saving C-states are not suitable in high performance configurations, therefore, it should be disabled.

– NUMA node interleaving

Enabling Node Interleaving means that memory is interleaved between memory nodes, and there is no NUMA presentation to the operating system. For performance reasons, it is recommended to disable interleaving (and enable NUMA), thus ensuring that memory is always allocated to the local NUMA node for any given logical processor. See Hyper-Threading for more details.

• enable Turbo boost

Turbo Mode—(Intel) Turbo Boost Technology (TBT) automatically runs the processor core faster than the base frequency. The processor must be working in the power, temperature, and specification limits of the thermal design power (TDP). Both single and multi-threaded application performance is increased.

• set the Power Management option to either maximum or high performance

• set the CPU Frequency to maximum speed for maximum performance

• set the Memory Speed to maximum speed for maximum performance

3.3.2   NUMA

Most server motherboards with multiple physical CPU sockets use a NUMA
architecture. With NUMA, system memory is divided into multiple NUMA nodes,
typically one per CPU socket, to improve performance and system expandability.
When a CPU or I/O device needs to access a memory location, the latency and
memory bandwidth depends on whether the memory is part of the local NUMA node.
Access to non-local memory is slower than access to local memory.

➡️ **Note:** If NUMA is not configured correctly, performance issues may arise, including:

- excessive CPU usage
- random CPU spikes

See Numatune for information about correct NUMA configuration on Linux.

See Set NUMA Node Affinity for information about correct NUMA configuration on VMware.

3.3.2.1   NUMA Topology

Figure 3 shows the NUMA topology. This diagram is also applicable to an Intel
Haswell or Broadwell-based system, which supports up to four memory channels per
CPU, and uses the QPI bus for inter-socket communications.

Figure 3   **NUMA Topology**
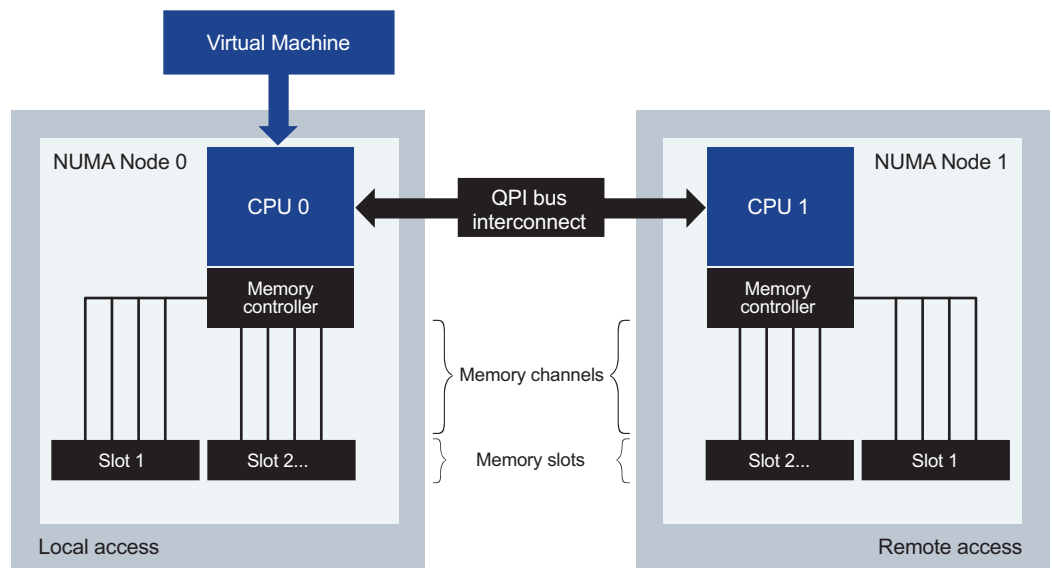
➡️ **Note:** All resources (such as CPU, memory, and NICs) must reside on the same NUMA, otherwise border violations may occur.

Several tools and commands are available to help you understand the NUMA topology of the host machine.

The **numactl** command can be used to:

- control processes and memory
- identify the number of NUMA nodes per server (for example, 1, 2, 4, and so on)
- view the memory per NUMA
- see the CPU core ID per NUMA, which is information required for a correct VSR configuration

The **numactl** command is not present by default, so must be installed using the **yum install** or **apt install** commands.

```
[root@vsr vsr-ws]# yum install numactl
root@sc-03:~# apt install numactl
```

The **numactl --hardware** command shows the memory size of each NUMA node and the "distance" of the NUMA nodes from each other. The following example shows command output:

```
$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 12 13 14 15 16 17
node 0 size: 128910 MB
node 0 free: 669 MB
node 1 cpus: 6 7 8 9 10 11 18 19 20 21 22 23
node 1 size: 129022 MB
node 1 free: 4014 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

The **virsh capabilities** command is another way to view the NUMA topology of the host. The following output shows an excerpt of the command output:

```
<topology>
        <cells num='2'>
                <cell id='0'>
                        <cpus num='12'>
                                <cpu id='0'/>
                                <cpu id='1'/>
                                <cpu id='2'/>
                                <cpu id='3'/>
```

```
                                        <cpu id='4'/>
                                        <cpu id='5'/>
                                        <cpu id='12'/>
                                        <cpu id='13'/>
                                        <cpu id='14'/>
                                        <cpu id='15'/>
                                        <cpu id='16'/>
                                        <cpu id='17'/>
                                </cpus>
                        </cell>
                        <cell id='1'>
                                <cpus num='12'>
                                        <cpu id='6'/>
                                        <cpu id='7'/>
                                        <cpu id='8'/>
                                        <cpu id='9'/>
                                        <cpu id='10'/>
                                        <cpu id='11'/>
                                        <cpu id='18'/>
                                        <cpu id='19'/>
                                        <cpu id='20'/>
                                        <cpu id='21'/>
                                        <cpu id='22'/>
                                        <cpu id='23'/>
                                </cpus>
                        </cell>
                </cells>
</topology>
```

The preceding sample output indicates that the host machine has two NUMA nodes (cells in virsh terminology) and each node is associated with 12 logical CPUs. The **virsh capabilities** command output does not indicate the free memory associated with each NUMA node; use the **virsh freecell** command to show this information.

To retrieve the affiliation of a network device (for example eth0) with a NUMA node, show the output of **/sys/class/net/eth0/device/numa_node**. Alternatively, if you know the bus number, slot number, and function number of the PCI device (for example, 82:00.0), you can show the output of **/sys/bus/pci/devices/ 0000\:82:\00.0/numa_node**.

The **lstopo** and l**stopo-no-graphics** commands provide another method of visualizing the NUMA topology of the system. The **hwloc** package must be installed to access these commands.

3.3.2.2   Assessing NUMA Layout and Processes

The efficiency of the NUMA layout can be assessed with the **numastat** command (which is part of the **numactl** package).

The **numastat** command can be used to detect NUMA border violations.

The **numastat -c qemu-kvm** command (on Red Hat/Centos) is useful to show the amount of memory each QEMU-KVM process (virtual machine) is using from each NUMA node.

3.3.2.3 Prepare VMs for Using NUMA

When the details of the host machine NUMA topology are understood, perform the following steps to improve performance.

Step 1. Ensure that NUMA is enabled in the BIOS. Some versions of BIOS have this functionality disabled by default. In Figure 4, NUMA is disabled.

Figure 4 **Status of NUMA in the BIOS**
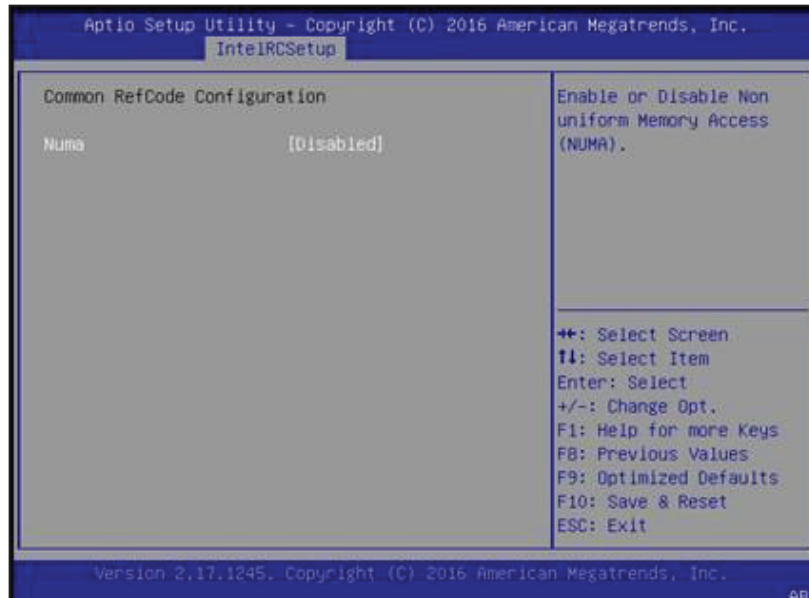
sc0069

Step 2. Allocate the vCPUs of the VM from one NUMA node, *x*. Use the <**cputune**> element or the **vcpu placement='auto'** described in Cputune.

Step 3. Allocate the guest memory (hugepages) of a VM from the same NUMA node, *x*. Use the <**numatune**> element described in Numatune.

Step 4. To use PCI passthrough or SR-IOV for specific network interfaces of the VM, choose PCI devices that are associated with NUMA node x.

3.3.3 Hyper-Threading

The hyper-threading feature on Intel CPUs allows each physical CPU core to appear as two logical processors to the operating system. Hyper-threading works by interleaving two execution threads on the same core. The two threads (siblings) share the same L1 or L2 cache, so that one thread executes while the other is waiting on data.

If a VSR VM is deployed on a host machine with hyper-threading enabled in the BIOS, then it can optimize its own software task placement to achieve the maximum performance benefit from the use of hyper-threading. This optimization is dependent on two conditions:

1. The VSR must be aware that hyper-threading is enabled on the host. The methods for conveying this status are hypervisor-dependent and are described in later sections of this guide.
2. The VSR must be aware how its vCPUs map to physical CPU threads. If the VSR detects that the host has hyper-threading enabled, it always assumes that its first two vCPUs are siblings of the same pCPU core, that the next two vCPUs are siblings of some other pCPU core, and so on. In general, this requires CPU pinning directives in the configuration of the VM. Methods for CPU pinning are hypervisor-dependent and are explained in later sections of this guide.

If the conditions outlined above cannot be fulfilled, then the performance of a VSR VM on a hyper-threaded host may actually be worse than the performance on the same machine with hyper-threading disabled. In fact, in these circumstances, Nokia strongly recommends that hyper-threading is disabled in the BIOS of the host machine.

The following techniques can be used to check whether a Linux host machine has hyper-threading disabled.

1. From the output of the **lscpu** command, if the Thread(s) per Core field in the output equals 1, this indicates that hyper-threading is disabled.
2. Show **/proc/cpuinfo** and check whether the displayed number of siblings is the same as the displayed number of CPU cores; if they are equal, this indicates that hyper-threading is disabled.

3.3.4   CPU Isolation

Caution: If the host OS schedules its own tasks to the CPU cores assigned to a VSR VM, VSR performance and stability could be compromised.

On Linux KVM hosts this can be avoided by using one of two methods: using the **isolcpus** kernel boot parameter or **systemd** CPU affinity.

The **isolcpus** kernel parameter specifies a list of CPU cores that should be avoided by the host scheduler, and this should match the list of CPUs to which VSR VMs are pinned. See Kernel Parameters for more details.

The CPUAffinity setting in `/etc/systemd/system.conf` specifies the subset of CPU cores that must be used by systemd for its tasks; this list should not overlap with the list of CPUs to which VSR VMs are pinned.

On VMware ESXi hosts CPU isolation can be achieved using one of two methods: by setting the **sched.cpu.latencySensitivity** property to **high** or by editing the VMX file with to include certain vCPU affinity directives. Further details are provided in the VMware section of this guide.

3.3.5   Host OS and Hypervisor

VSR VMs can be deployed on compute hosts that use either the Linux KVM hypervisor or the VMware ESXi hypervisor that is part of the VMware vSphere suite.

3.3.5.1   Linux KVM Compute Hosts

The Linux KVM hypervisor is supported for all VSR applications.

The KVM hypervisor requires a Linux operating system. The following Linux host operating systems are all qualified for use with VSR VMs running Release 16.0.R4 or later SR OS software:

- CentOS 7.0-1406 with 3.10.0-123 kernel
- CentOS 7.2-1511 with 3.10.0-327 kernel
- CentOS 7.4-1708 with 3.10.0-693 kernel
- Centos 7.5-1804 with 3.10.0-862 kernel

- Red Hat Enterprise Linux 7.1 with 3.10.0-229 kernel
- Red Hat Enterprise Linux 7.2 with 3.10.0-327 kernel
- Red Hat Enterprise Linux 7.4 with 3.10.0-693 kernel
- Red Hat Enterprise Linux 7.5 with 3.10.0-862 kernel
- Ubuntu 14.04 LTS with 3.13 kernel
- Ubuntu 16.04 LTS with 4.4 kernel

VSR VMs can be deployed on compute hosts running the KVM hypervisor using tools provided by the Linux libvirt software package (see Deploying and Managing VSR VMs Using Libvirt) or using the OpenStack cloud management software (see OpenStack Overview for more information).

3.3.5.1.1   Kernel Parameters

When it is started, the Linux kernel accepts certain command-line options or boot parameters. The Centos, RHEL, and Ubuntu installations include the GNU Grand Unified Boot loader version 2 (GRUB2) that allows the user to pass boot parameters to the kernel. The method of specifying and updating the boot parameters depends on the specific Linux distribution.

RHEL and Centos

When the legacy boot mode is used (non-UEFI) on RHEL and Centos hosts, GRUB2 reads its configuration from the **/boot/grub2/grub.cfg** file. The **grub.cfg** file is generated during installation.

➡ **Note:** The **grub.cfg** file should never be edited directly. Use the **grub2-mkconfig** utility to manually regenerate the file.

The **grub2-mkconfig** utility regenerates the **grub.cfg** file using the template files in the **/etc/grub.d/** directory, and the custom settings in the **/etc/default/grub** file.

Perform the following steps on RHEL and Centos hosts to pass a series of boot parameters (param1, param2) to the kernel at the next boot:

1. Add or edit the following line in the **/etc/default/grub** file to prepare the boot template:

```
GRUB_CMDLINE_LINUX="param1[=value_1][,=value_2]…[,=value_1
0] <space> param2[=value_1][,=value_2]…[,=value_10]
<space>..."
```

2. Use one of the following commands, depending on the mode of server or Linux boot:
   - for legacy, use **grub2-mkconfig -o /boot/grub2/grub.cfg**
   - for EFI, use **grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg**

Ubuntu

Perform the following steps on Ubuntu hosts to pass a series of boot parameters (param1, param2) to the kernel at the next boot:

1. Add or edit the following line in the **/etc/default/grub** file to prepare the boot template:

```
GRUB_CMDLINE_LINUX="param1[=value_1][,=value_2]…[,=value_1
0] <space> param2[=value_1][,=value_2]…[,=value_10]
<space>..."
```

2. Use the **sudo update-grub** command to convert the template to a boot record.

Recommended Kernel Boot Parameters for VSR Deployment

The following sections describe the recommended kernel boot parameters for:

- SR-IOV and PCI Passthrough
- Huge Pages
- SELinux Extensions
- Spin-lock Loops
- Isolcpus Kernel Boot Parameter
- Kernel Boot Parameter Example

SR-IOV and PCI Passthrough

If the VSR host server will run one or more VSR VMs that use SR-IOV or PCI passthrough, the **GRUB_CMDLINE_LINUX** string must include the following kernel settings:

- `pci=realloc`

This setting enables kernel reallocation of PCI bridge resources if the BIOS allocations are too small.

- `pcie_aspm=off`

  This setting disables the PCIe Active State Power Management.

- `iommu=pt`

  This setting bypasses DMA remapping in the Linux kernel.

- `nopat`

  This setting disables the page attribute table extensions.

- `intel_iommu=on`

  This setting enables IOMMU on Intel servers.

Huge Pages

The memory of the VMs should be backed by explicit 1 GByte huge pages to optimize VSR data path performance. The following huge pages mechanisms are available to a system:

- Explicit HugePages

  These huge pages are reported as HugePages_Total in /proc/memory.
- Transparent Huge Pages (THP)

  These huge pages are reported as AnonHugePages in /proc/memory.

Due to its required interactions with the **hugetlbfs** filesystem, applications must be written to use the Explicit HugePages. However, THP has no such dependencies because the kernel (**khugepaged** daemon) automatically aggregates default-sized pages into huge pages.

Explicit huge pages work best for NFV applications because they are never swapped to disk and the necessary support is built-in to many hypervisors. See Guest Memory Backing for information about how to back the memory of a KVM VM using Explicit HugePages.

Use the following commands to reserve a specified number of 1Gbyte huge pages at runtime:

- To allocate 20 huge pages of the default size (not persistent):

  `echo 20 > /proc/sys/vm/nr_hugepages`
- To allocate 20 huge pages of the default size (persistent):

  `sysctl -w vm.nr_hugepages=20`
- To allocate four 1Gbyte huge pages from NUMA node 1:

```
echo 4 > /sys/devices/system/node/node1/hugepages/
hugepages-1048576kB/nr_hugepages
```

Runtime huge page allocation can sometimes fail because memory has become too fragmented. For this reason, it is recommended to allocate huge pages at boot time, by including the following kernel boot parameters in the **GRUB_CMDLINE_LINUX** string:

```
hugepagesz=1G, hugepages=n, default_hugepagesz=1G
```

> **Note:** Set the value of **n** carefully; do not set the value so high that all host memory is allocated in 1 Gbyte huge pages. It is essential to leave enough host memory using the normal 4 K page size. For example, set **n** to the amount of memory in GBytes less the reserved host memory. If a system has 64 GBytes RAM and 8GBytes is reserved host memory, then **n**=56 in this example.

> **Note:** In a NUMA system, the number of huge pages requested in the kernel boot option is the total number across all NUMA nodes. If you specify hugepages=*n* and there are M NUMA nodes, then *n*/M huge pages are allocated from each NUMA node; that is, the system attempts to allocate an equal number of huge pages from each NUMA node.

SELinux Extensions

For VSR deployment, Security Enhanced Linux (SELinux) must be disabled, or set to permissive mode. Kernel audit should also be disabled.

Disable the Security Enhanced Linux (SELinux) extensions and the kernel audit procedures by adding the following settings to the **GRUB_CMDLINE_LINUX** string:

```
selinux=0
```

```
audit=0
```

You can also disable SELinux by editing the SELINUX setting in the **/etc/selinux/config** file as follows:

```
SELINUX=disabled
```

After you have updated the configuration file, reboot the system and use the **sestatus** command to verify the change.

Spin-lock Loops

In some cases, spin-lock issues can cause poor VSR performance. You can address these issues by using the Pause Loop Exiting (PLE) Intel VT-x feature. When the PLE feature is used, a VM-Exit is triggered if an excessive number of PAUSE instructions are issued by a vCPU. In such cases, the PLE infers that the vCPU is probably waiting on another vCPU to release a lock, and without the VM-Exit, there is no opportunity for the other vCPU to be scheduled so that it can actually release the lock.

To exit immediately from a spin-lock loop, add the following setting to the GRUB_CMDLINE_LINUX string:

kvm_intel.ple_gap=0

Isolcpus Kernel Boot Parameter

The **isolcpus** kernel boot parameter is one of the supported methods of achieving CPU isolation on a Linux KVM compute server; see CPU Isolation for more information. The **isolcpus** parameter provides a list of CPUs (specified as ranges and/or comma-separated values) that the host Linux scheduler should bypass when scheduling tasks to the cores. The only workload on these isolated cores is work-assigned to run there using CPU pinning (see Cputune). By pinning the vCPUs of the VSR VMs onto some subset of the host isolated cores, resource contention for these cores is avoided and VSR performance is maximized.

The following shows an example **isolcpus** setting in the GRUB_CMDLINE_LINUX string:

```
isolcpus=1-13
```

Unsupported SFPs

Some Intel NICs may, by default, not support non-Intel SFP optics. This situation may be remedied by adding a kernel boot parameter setting such as the following:

```
ixgbe.allow_unsupported_sfp=1,1,1,1,1,1,1,1,1,1,1,1
```

➡ **Note:** Some driver versions do not support the above parameter; in this case, non-Intel SFPs do not function correctly. Nokia recommends first testing this parameter in a lab environment.

Kernel Boot Parameter Example

The following shows an example kernel boot parameter setting for a hyper-threaded system.

➡ **Note:** When configuring the kernel boot parameters, ensure the use of correct values for the number of huge pages and the number of CPUs to isolate from the host scheduler. These values may differ from those used in this example.

```
GRUB_CMDLINE_LINUX="pci=realloc pcie_aspm=off iommu=pt
intel_iommu=on nopat hugepagesz=1G default_hugepagesz=1G
hugepages=50 isolcpus=1-9,11-19 selinux=0 audit=0
kvm_intel.ple_gap=0"
```

3.3.5.1.2 Linux vSwitch Implementations

A virtual switch (vSwitch) is a software implementation of a Layer 2 bridge or Layer 2-3 switch in the host OS software stack. When the host has one or more VMs, the vNIC interfaces (or some subset) can be logically connected to a vSwitch to enable the following:

- vNIC-to-vNIC communication within the same host without relying on the NIC or other switching hardware in the data center
- multiple vNICs to share the same vSwitch "uplink"

The following vSwitch implementation options are available on KVM Linux hosts:

- Linux Bridge
- Open vSwitch
- Open vSwitch with DPDK
- Nuage Networks VRS (refer to the *Nuage VSP Installation Guide* for more information)

Linux Bridge

The Linux bridge is a software implementation of an IEEE 802.1D bridge that forwards Ethernet frames based on learned MACs. It is part of the **bridge-utils** package. The Linux bridge datapath is implemented in the kernel (specifically, the **bridge** kernel module), and it is controlled by the **brctl** userspace program.

On Centos and RHEL hosts, a Linux bridge can be created by adding the **ifcfg-brN** (where **N** is a number) file in the **/etc/sysconfig/network-scripts/** directory. The contents of this file contain the following directives:

- **DEVICE=brN** (with **N** correctly substituted)
- **TYPE=Bridge** (**Bridge** is case-sensitive)

The following output shows an **ifcfg** file:

```
TYPE=Bridge
DEVICE=br0
IPADDR=192.0.2.1
PREFIX=24
GATEWAY=192.0.2.254
DNS1=8.8.8.8
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
DELAY=0
```

To add another interface as a bridge port of **brN**, add the **BRIDGE=brN** directive to the **ifcfg** network-script file for that other interface.

On Ubuntu hosts, a Linux bridge is created by adding an **auto brN** stanza followed by an **iface brN** stanza to the **/etc/network/interfaces** file. The **iface brN** stanza can include several attributes, including the **bridge_ports** attribute, which lists the other interfaces that are ports of the Linux bridge.

The following output shows an **/etc/network/interfaces** file that creates a **bridge br0** with **eth0** as a bridge port:

```
auto lo
iface lo inet loopback
auto br0
iface br0 inet dhcp
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

→ **Note:** Use the **systemctl restart network** command to restart networking after making changes to network configuration files.

By default, the Linux bridge is VLAN unaware and it does not take VLAN tags into consideration, nor does it modify them when forwarding the frames. If the bridge is configured to have VLAN sub-interfaces, frames without a matching VID are dropped or filtered. If a VLAN sub-interface of a port is added as a bridge port, then frames with the matching VID are presented to the bridge with the VLAN tag stripped. When the bridge forwards an untagged frame to this bridge port, a VLAN tag with a matching VID is automatically added.

The following methods can be used to configure MTU on a Linux bridge.

- If the Linux bridge has a physical port connected to it, ensure that the correct MTU is assigned to a physical port.
- If the Linux bridge does not have a physical interface connected, add a "dummy" interface to it.

The following configuration example for dummy interfaces is not persistent after reboot; a boot script must be prepared to automate it.

i. Create three dummy interfaces to be used by three different Linux bridges using the **modprobe dummy numdummies=3** command.

ii. Verify the MTU of dummy interfaces using the **ip link | grep dummy** command.

```
ip link | grep dummy
                17: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DE
FAULT group default qlen 1000
                18: dummy1: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DE
FAULT group default qlen 1000
                19: dummy2: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DE
FAULT group default qlen 1000
```

iii. Change the MTU for all dummy interfaces to a specified value using the **ip link set dev** *dummy-interface* **mtu** *mtu-value* command.

```
ip link set dev dummy0 mtu 9212
ip link set dev dummy1 mtu 9212
ip link set dev dummy2 mtu 9212
```

iv. Verify the MTU of dummy interfaces using the **ip link | grep dummy** command.

```
ip link | grep dummy
                17: dummy0: <BROADCAST,NOARP> mtu 9212 qdisc noop state DOWN mode DE
FAULT group default qlen 1000
                18: dummy1: <BROADCAST,NOARP> mtu 9212 qdisc noop state DOWN mode DE
FAULT group default qlen 1000
                 19: dummy2: <BROADCAST,NOARP> mtu 9212 qdisc noop state DOWN mode D
EFAULT group default qlen 1000
```

v. Add a dummy interface to a Linux bridge using the **ip link set dev** *dummy-interface* **master br-test** and **ip link set dev** *LB-bridge* **up** commands.

```
ip link set dev dummy0 master br-test
ip link set dev br-test up
```

vi. Verify the MTU of the Linux bridge using the **ip link show** *LB-bridge* command.

```
ip link show br-test
                20: br-test: <NO-
CARRIER,BROADCAST,MULTICAST,UP> mtu 9212 qdisc noqueue state DOWN mode DEFAULT group
 default qlen 1000
                                link/ether de:95:59:17:2e:12 brd ff:ff:ff:ff:ff:ff
```

This Linux bridge can now be used to serve traffic.

Open vSwitch

Open vSwitch (OVS) is an open-source switching stack for virtualization that offers significantly more functionality than the Linux bridge. Key features of OVS include the following:

- programmability via the OpenFlow and OVSDB protocols
- Layer 2 and Layer 3 forwarding (IPv4/IPv6)
- kernel or user-space forwarding (using DPDK)
- flow-caching
- tunneling (GRE, VXLAN, STT, and Geneve)
- QoS and policing
- support for networking protocols including STP, RSTP, BFD, and LACP
- traffic monitoring via NetFlow, sFlow, IPFIX, SPAN, RSPAN, and GRE tunneled mirrors
- persistency over reboot

OVS is supported by most virtualization platforms, including KVM. OVS and OVS-DPDK packages are available for Ubuntu, Centos, and RHEL (in addition to other Linux distributions). OVS is the most popular networking plug-in for OpenStack.

OVS comprises the following three components:

- **ovs-vswitchd**—a userspace daemon

  The **ovs-vswitchd** daemon uses OpenFlow to interact directly with controllers and uses the NETLINK protocol to communicate with the kernel module.

- **ovsdb-server**—a database daemon

  The **ovsdb-server** daemon maintains a persistent database of the switch configuration. The daemon uses the OVSDB protocol (RFC 7047) to interact with external controllers and the **ovs-vswitchd** daemon.

- **openvswitch.ko**—a kernel module

In a native OVS (non-DPDK) scenario, the first packet of each new flow is directed to **ovs-vswitchd**, which decides how that packet (and others in the same flow) should be forwarded. The forwarding decision is encoded in a flow cache entry programmed by **ovs-vswitchd** into the kernel module. When future packets hit the flow cache entry, they are forwarded entirely by the kernel module and do not need to be directed to the **ovs-vswitchd** "slow path".

Consider the following when configuring OVS:

- By default, OVS operates in MAC learning mode; however, it can program flow using OpenFlow or Manual flow configuration. As a result, unusual forwarding decisions may be made by OVS, which can be difficult to debug.
- MTU considerations:
  - The tunneling techniques for OVS add additional overhead, and OVS does not support fragmentation. As a result of additional overhead used by OVS tunneling techniques, some traffic could be silently dropped along the path.
  - MTU can be set using the **ovs-vsctl set int** *OVS-name* **mtu_request=9212** command.
  - MTU must be set before the VMs are started.

> **➡** **Note:** Commands may differ depending on the OpenStack version. Refer to OpenStack documentation for applicable commands to each version.

Open vSwitch with DPDK

DPDK is an open-source toolkit for fast packet processing. When OVS is compiled to use DPDK libraries and DPDK NIC drivers, the result is a higher performance vSwitch, which is referred to as OVS-DPDK (in this document). OVS-DPDK is considerably faster (7x to 10x) than native OVS due to the following reasons:

- The OVS-DPDK fast path moves from the openvswitch.ko kernel module to a user-space implementation (the **dpif-netdev** component of the **ovs-vswitchd** daemon).
- OVS-DPDK communicates with VM vNIC ports (that use a VirtIO driver) using user-space vHost drivers (**vhostuser**).
- Poll-mode-driver (PMD) threads of the user space **ovs-vswitchd** process send and receive packets over the OVS switch ports.

Refer to the *SR OS 20.x.Rx*. Software Release Notes for information about the currently supported versions of OVS-DPDK.

3.3.5.2 VMware ESXi

A VSR-I VM supporting any application can be deployed on a VMware ESXi compute host using VMware vCloud Director (vCD) or the vSphere Web Client interface to a vSphere vCenter Server. Release 16.0.R4 supports the following deployment options:

- ESXi 6.0 Update 2, vCD 8.10 and vCenter Server 6.0 (vCloud NFV 1.5)
- ESXi 6.5 Update 1, vCD 8.20 and vCenter Server 6.5 (vCloud NFV 2.0)
- ESXi 6.7 and vCenter Server 6.7

In addition, for the RR application only, a VSR-I can also be deployed on a VMware ESXi 5.5 compute host, but in this case, only the vSphere Web Client interface is supported.

The following vSphere features are supported with the VSR-I, regardless of application or workload:

- Distributed Resource Scheduler (DRS)—but not fully-automated mode
- High Availability
- vSphere standard switch—connected to the guest using an E1000 or VMXNET3 driver
- vSphere distributed switch (vDS)—connected to the guest using an E1000 or VMXNET3 driver
- SR-IOV and PCI passthrough (NIC model dependent)

The following vSphere features are unsupported:

- DRS fully-automated mode
- vMotion
- Storage vMotion
- Fault Tolerance

3.3.6 Data Center Networking

A typical data center has many racks of servers, each with a TOR switch, such as the Nuage Networks 7850 VSG. Each compute server is cabled to its TOR switch (typically nx10GE), and each TOR may be connected (at 40GE or higher speeds) into a CLOS-type topology of leaf and spine switches. A gateway router, such as the 7750 SR or 7950 XRS, may connect the data center network to the wide-area or metro network.

The interconnection of TOR, leaf, and spine switches, and DC gateway forms the underlay network of the data center. Network virtualization using VXLAN or GRE encapsulation allows per-tenant overlay networks to be created on top of the common underlay. Overlay networks provide several advantages, including:

- Provide security and isolation between VMs that should not be able to communicate.
- Allow different tenants to use the same overlapping address space.
- Keep state out of the underlay network, allowing for higher scale.
- Facilitate live migration of VMs so that VMs can retain their current IP addresses while being moved across IP subnet boundaries in the underlay network.

In the VSR context, overlay networks are well-suited for creating an out-of-band management network connecting multiple VSRs in the data center to internal management systems.

Overlay networks are not as well-suited for carrying user plane traffic into and out of VSR VMs. One complication is the presumed nature of the packets encapsulated by VXLAN: VXLAN expects untagged Ethernet frames as the payload but much of the user plane traffic that is sent to a VSR-PE or VSR-BNG could be MPLS encapsulated if it originates or terminates in the WAN.

4 VSR Software Licensing

4.1 Overview

A software license key must be installed on every VSR system, allowing the system to load the valid license file at bootup, in order for it to be fully operational. The license file encodes the allowed capabilities and features of the VSR system. Contact your Nokia account representative to obtain license files associated with a purchase order or trial request.

A VSR system can be booted up without a license key but a forced reboot will occur after 60 minutes and during that time window no system configuration is supported; the available commands are restricted to a minimum set of operational commands.

The license file for a VSR system can be stored on a local storage device of the VM or on a remote FTP server. The license file location is configurable as a BOF parameter or it can be passed as an SMBIOS value. The license file is read at boot up time. If the license file is used or changed after the system is up, the new license file can be re-read and re-activated if there is no change to the software version, system type, or the set of licensed features. The **admin system license validate** command reads a license file to determine whether a valid license can be found inside the referenced file (or inside the BOF referenced license-file if no URL is provided). The **admin system license activate** command is used to proceed with activating the new license.

4.2   VSR-I License Keys

When you purchase software licenses for one or more VSR-I systems, your Nokia account representative will provide you with corresponding VSR-I license key files, which could be one license file for all the VSR-I systems or a separate license file for each one.

If you are given a common "wildcard UUID" license file for your VSR-I system, there is no restriction on the UUIDs of the VSR-I VMs; they can have any value. When each VSR-I has its own "UUID-locked" license file, all the license records in that license file are locked to a specific UUID value (readable as cleartext), and this license file is intended for the specific VSR-I system that runs in a VM with that UUID value.

To associate a VSR-I with its license file, you must correctly set the **license-file** boot option (BOF) parameter on the VSR-I. The **license-file** parameter can be specified by editing the BOF file (before or after bootup), or by including it in the configuration data of the VM and passing it to the guest (VSR) as SMBIOS information. The **license-file** parameter can reference a file stored on a local disk (for example, CF3:) or a file stored on an FTP server. See Sysinfo for more information about SMBIOS parameters.

When the VSR software starts booting and determines that the system type (chassis) is VSR-I, it attempts to read and parse the referenced license file. If a valid license key is not found or the one found is corrupt, the system is allowed to complete its bootup procedures but only a limited number of non-configuration-related commands are available in this state, and the system is forced to reboot after 60 minutes.

A valid license key for the VSR-I system must meet the following criteria:

- the license is for a VSR (not vSIM)
- the license is for a VSR-I system

   VSR-I and VSR-D licenses are not interchangeable.

- the UUID of the VM matches the one encoded in the "UUID-locked" license key (if applicable)
- the VSR software version (the major release number) matches the one encoded in the license key
- the license file is not expired

If VSR reports that a valid license record was not found, first ensure that the license meets the above criteria. If all of the above reasons are ruled out, it is possible that the license file became corrupted. Re-downloading or re-installing the license-file may help in this case. To verify access to the license file and view it at the same time, use the **file type** *license-url* command.

4.3 Feature Licenses

In addition to allowing the system to boot into a fully operational state, the license file also encodes the set of value-added features that have been purchased for use on the VSR-I systems covered by the license.

Table 3 lists the features enabled by purchasable Application Specific Licenses (ASLs) in the current release. The table indicates whether some form of CLI enforcement is currently in place for the associated functionality. If an ASL has CLI enforcement, an error message will be displayed when you attempt to configure the associated functionality. In most cases, the command triggering the error (and its sub-trees) is not added to the configuration of the system.

Table 3   **VSR ASL Support**

Category	ASL Feature	ASL Key Enforcement
Platform	Adv DCGW and Svc Chain	No
	Advanced QoS BW	No
	BGP Route Reflection	No
	BNG	Yes
	Hybrid OpenFlow Switch	No
	IPSec	Yes
	IP Tunnels	No
	Legal Intercept	No
	LNS	Yes
	NAT	Yes
	NGE	Yes
	Telemetry	No
	VPN	No
	vRGW	Yes
	WLAN Gateway	Yes

Table 3   **VSR ASL Support (Continued)**

Category	ASL Feature	ASL Key Enforcement
Application Assurance	AA Identification	Yes
	AA Control	Yes
	AA Policing	Yes
	AA Stateful Firewall	Yes
	AA RTP Performance	Yes
	AA ICAP Control	Yes
	AA In-Browser Notification	Yes
	AA Local List URL Filtering	Yes
	AA Dynamic Experience Management	Yes
IPSec	IPSec Geo Redundancy	Yes
NAT	NAT Geo Redundancy	Yes
	UPnP	Yes
	L2-Aware NAT	Yes
	LSN	Yes
	MAP-T	Yes
WLAN GW	Multiple SSID	Yes
	Inter-AP Mobility	Yes
	WLGW Geo Redundancy	Yes

4.4   Checking the License Status

Once the VSR is operational, check the license status of the system, including the set of licensed features. At the prompt, type the following:

show system license ↵

The following is sample output for a VSR-I with a valid license:

```
A:Dut-A# show system license
===============================================================================
Current License
===============================================================================
License status : monitoring, valid license record
Time remaining : 137 days 7 hours
-------------------------------------------------------------------------------
License name   : sr-regress@list.nokia.com
License uuid   : 00000000-0000-0000-0000-000000000000
Machine uuid   : 3ffaf6a4-edce-45ab-bde6-c7d1587103f9
License desc   : Virtual SR [Integrated] [ALL]
License prod   : Virtual-SR
License sros   : TiMOS-B-16.0.*
Current date   : WED MAY 30 17:33:59 UTC 2018
Issue  date    : MON APR 16 23:34:58 UTC 2018
Start  date    : SUN APR 15 00:00:00 UTC 2018
End    date    : MON OCT 15 00:00:00 UTC 2018
-------------------------------------------------------------------------------
vChassis       : VSR-I
vSR CPMs       : limit: 1
vSR IOMs       : limit: 1
-------------------------------------------------------------------------------
AA_RTU         : AA Identification
                 AA Control
                 AA Policing
                 AA Stateful Firewall
                 AA RTP Performance
                 AA ICAP Control
                 AA In-Browser Notification
                 AA Local List URL Filtering
                 AA Dynamic Experience Management
IPSEC_RTU      : IPSec Geo Redundancy
NAT_RTU        : NAT Geo Redundancy
                 UPnP
                 L2-Aware NAT
                 LSN
                 MAP
VSR_RTU        : Legal Intercept
                 Advanced QoS BW
                 VPN
                 BNG
                 LNS
                 WLGW
                 IPSec
                 vRGW
                 Adv DCGW and Svc Chain
                 Hybrid OpenFlow Switch
```

```
                        IP Tunnels
                        NGE
WLGW_RTU          : Multiple SSID
                        Inter-AP Mobility
                        WLGW Geo Redundancy
===============================================================================
A:Dut-A#
```

5 Deploying VSR on Linux KVM Hosts Using Libvirt or OpenStack

5.1 Introduction

There are several methods commonly used to define the properties of a VSR VM and start it under the control of the Linux-KVM hypervisor, including:

- specifying the VM parameters in a domain XML file read by **virsh**, the **libvirt** command shell
- using the **virt-manager** GUI application available as part of the **libvirt** package
- using the **qemu-kvm** (RedHat/Centos) or **qemu-system-x86_64** (Ubuntu) commands
- using OpenStack or other cloud management platforms to create the VM

This chapter provides an overview of the first and last methods.

The Linux **libvirt** package provides the Virtual Shell (**virsh**) command-line application to facilitate the administration of VMs. The **virsh** application provides commands to create and start a VM using the information contained in a domain XML file. It also provides commands to shut down a VM, list all the VMs running on a host, and output specific information about the host or a VM.

OpenStack is open-source cloud management software that performs the role of a Virtualized Infrastructure Manager (VIM) in the ETSI NFV reference architecture. The VIM is responsible for controlling and managing the NFVI compute, storage, and network resources within a data center. The OpenStack software is written in Python and available freely under an Apache 2.0 license. There are multiple OpenStack distributions, some of which come with technical support and other services. The modular architecture of OpenStack allows the installation of different components as needed. In this chapter, most of the guidelines will apply to the OpenStack Nova component, which is responsible for compute management.

5.2   Deploying and Managing VSR VMs Using Libvirt

Libvirt is open source software that provides a set of APIs for creating and managing VMs on a host machine, independent of the hypervisor. Libvirt uses XML files to define the properties of VMs and virtual networks. It also provides a convenient **virsh** command line tool.

The **libvirt** domain XML file for a VSR VM defines the important properties of the VM. Use any text editor to create the domain XML file, then pass the filename as a parameter of the **virsh create** command to start up the VSR VM; for example, **virsh create domain1.xml**.

Use **virsh** commands to show information about the VM or change specific properties. Table 4 lists the basic virsh commands, where **VM_name** is the value that you configured for the **name** element in the XML configuration file. Refer to http://libvirt.org/virshcmdref.html for more information.

Table 4   **Basic virsh Commands**

Command	Example	Result
capabilities | grep cpu	**virsh capabilities | grep cpu** ↵	Shows the number of cores on the physical machine
console	**virsh console *VM_name*** ↵	Connects the serial console of the VM if using the serial PTY port
define	**virsh define *VM_name*.xml** ↵	Reads the XML configuration file and creates a domain
destroy	**virsh destroy *VM_name*** ↵	Stop and power down a VM (domain). The terminated VM is still available on the host and can be started again. The system status is "shut off".
dumpxml	**virsh dumpxml *VM_name*** ↵	Shows the XML configuration information for the specified VM, including properties added automatically by **libvirt**
list	**virsh list [--all | --inactive]** ↵	The "--all" argument shows all active and inactive VMs that have been configured and their state The "--inactive" argument shows all VMs that are defined but inactive
nodeinfo	**virsh nodeinfo** ↵	Shows the memory and CPU information, including the number of CPU cores
start	**virsh start *VM_name*** ↵	Starts the VM domain

Table 4   **Basic virsh Commands (Continued)**

Command	Example	Result
undefine	**virsh undefine *VM_name*** ↵	Deletes a specified VM from the system
vcpuinfo	**virsh vcpuinfo *VM_name*** ↵	Shows information about each vCPU of the VM

➡️ **Note:** The **virsh shutdown** and **virsh reboot** commands do not affect VSR VMs because the VSR software does not respond to the associated ACPI signals.

Some VM property changes made from the **virsh** command line do not take immediate effect because the VSR does not recognize and apply these changes until the VM is destroyed and restarted. Examples of these changes include:

- modifying the vCPU allocation with the **virsh setvcpus** command
- modifying the vRAM allocation with the **virsh setmem** command
- adding or removing a disk with the **virsh attach-disk**, **virsh attach-device**, **virsh detach-disk,** or **virsh detach-device** commands
- adding or removing a vNIC with the **virsh attach-interface**, **virsh attach-device**, **virsh detach-interface**, or **virsh detach-device** commands

5.2.1   Libvirt Domain XML Structure

The **libvirt** domain XML file describes the configuration of a VSR VM. The file begins with a **<domain type='kvm'>** line and ends with a **</domain>** line. In XML syntax, **domain** is an *element* and **type='kvm'** is an *attribute* of the **domain** element. VSR VMs must have the **type='kvm'** attribute because KVM acceleration is mandatory. Other domain types, including **type='qemu'**, are not valid.

The **libvirt** domain XML file structure can conceptually be interpreted as a tree, where the **domain** element is the root element and contains all the sub-elements (child elements) in the file. All sub-elements can contain their own child elements, and so on. The following **domain** child elements must be configured for VSR VMs:

- name, Domain Name and UUID
- uuid, see Domain Name and UUID
- memory, see Memory
- memoryBacking, see Guest Memory Backing
- vcpu, see vCPU

- cputune, see Cputune
- numatune, see Numatune
- cpu, see CPU
- sysinfo, see Sysinfo
- os, see OS
- features, see Hypervisor Features
- clock, see Clock
- devices, see Devices
- seclabel, see Seclabel

5.2.1.1 Domain Name and UUID

Use the **<name>** element to assign each VM a meaningful name. The name should
be composed of alphanumeric characters (spaces should be avoided) and must be
unique within the scope of the host machine. Use the **virsh list** command to show
the VM name. The following is an example of a <name> element:

```
<name>vsr-i</name>
```

Each VM has a globally unique UUID identifier. The UUID format is described in RFC
4122. If you do not include a **<uuid>** element in the domain XML file, **libvirt** auto
generates a value that can be displayed after the VM is created using the **virsh
dumpxml** command. Setting the UUID value explicitly ensures that it matches the
UUID specified in the software license. See VSR Software Licensing, for information
about VSR software licenses. The following is an example of a **<uuid>** element,
using the correct RFC 4122 syntax:

```
<uuid>ab9711d2-f725-4e27-8a52-ffe1873c102f</uuid>
```

5.2.1.2 Memory

The maximum memory (vRAM) allocated to a VM at boot time is defined in the
<memory> element. The '**unit**' attribute is used to specify the unit to count the vRAM
size.

Note: The unit value is specified in kibibytes (2^10 bytes) by default. However, all memory
recommendations in this document are expressed in units of gibibytes (2^30 bytes), unless
otherwise stated.

To express a memory requirement in gibibytes, include a **type='G'** (or **type='GiB'**) attribute, as shown in the following example:

```
<memory unit='G'>6</memory>
```

The amount of vRAM needed for a VSR VM depends on the VSR system type, VSR card type, and the MDAs installed in the system or card. Refer to the *SR OS 20.x.Rx. Software Release Notes* for the minimum memory requirement for VSR-I VMs.

5.2.1.3   Guest Memory Backing

Include the <**memoryBacking**> element to disable Kernel SamePage Merging (KSM) and to back the vRAM memory of a VM with hugepages. KSM is disabled by the <**nosharepages/**> element described in Kernel SamePage Sharing.

Hugepages boost performance by minimizing Translation Lookaside Buffer (TLB) cache misses when the guest code needs to map a virtual memory address to a physical memory address. All the vRAM of a VSR-I VM must be backed by 1G hugepages.

Note: CPU support for 1G hugepages is indicated by the **pdpe1gb cpu** flag. Use the **cat /proc/cpuinfo** command to show the CPU flags for each CPU in the host.

The following shows an example <**memoryBacking**> configuration suitable for VSR-I VMs.

```
<memoryBacking>
<hugepages>
      <page size='1' unit='G' nodeset='0'/>
</hugepages>
<nosharepages/>
</memoryBacking>
```

In the <**hugepages**> configuration, the **page size** and **unit** refer to the size of hugepages allocated by the host, and **nodeset** specifies the guest NUMA node to which the hugepages will be assigned.

This XML block prevents consumption of memory from the non-hugepages block, which is reserved for Linux tasks. Errors could break a hypervisor, resulting in loss of remote access to a server.

5.2.1.3.1   Kernel SamePage Sharing

The KSM feature allows KVM VMs to share identical memory pages. When KSM is
enabled and it finds two or more guests with an identical memory page, it reduces
the duplicates to a single page and marks the page for copy-on-write. If the contents
of the page are modified by a guest, a new page is created for that guest.

KSM has a performance overhead and minimal benefit on a host that is running only
one or several VSR VMs. To disable KSM for a VM, include **<nosharepages/>** as a
child element of the **<memoryBacking>** element.

5.2.1.4   vCPU

The number of vCPUs allocated to a VM is defined in the **<vcpu>** element. In the
current release, a minimum of two or three vCPUs can be allocated to a VSR VM,
depending on the supported applications. The maximum number of vCPUs that can
be used by any VSR VM is 56.

The <vcpu> element contains the following attributes:

- **cpuset**

  The **cpuset** attribute provides a comma-separated list of physical CPU numbers
  or ranges, where ^ indicates exclusion. Any vCPU or vhost-net thread
  associated with the VM that is not explicitly pinned by the **<cputune>**
  configuration is pinned to one of the physical CPUs allowed by the **cpuset**
  attribute.

  When using the **cpuset** attribute, ensure that all the allowed physical CPUs
  belong to one NUMA node; see NUMA for more information.

- **current**

  The **current** attribute allows fewer than the maximum vCPUs to be allocated to
  the VM at boot up. This attribute is not required for VSR VMs because in-service
  changes to the vCPU allocation are not allowed.

- **placement**

  The **placement** attribute accepts a value of either **static** or **auto**. Use **static**
  when specifying a **cpuset**. When **auto** is used, **libvirt** ignores the **cpuset**
  attribute and maps vCPUs to physical CPUs in a NUMA-optimized manner
  based on input from the **numad** process. The **placement** attribute defaults to
  the placement mode of **<numatune>**, or to **static** if a **cpuset** is specified.

The following example **<vcpu>** configuration for a VSR VM allocates four vCPUs.
There are no **placement** or **cpuset** attributes in this example because it is assumed
that CPU pinning requirements are handled by the **<cputune>** element.

```
<vcpu>4</vcpu>
```

Use **<cputune>** to meet the CPU pinning requirements of VSR-I VMs. See Cputune for more information.

5.2.1.5   Cputune

The **<cputune>** element provides fine-grained control of the physical CPU resources used by the VM. The **<vcpupin>** child elements enable individual guest vCPUs to be mapped to specific physical CPUs using the **cpuset** attribute. The **cpuset** attribute has the same syntax as the **<vcpu> cpuset** attribute.

The **<cputune>** element overrides the **cpuset** policy specified in the **<vcpu>** element. See vCPU for more information.

The **<emulatorpin>** child element allows you to restrict the set of physical CPUs used by the vhost-net threads associated with the VirtIO interfaces of the VM.

For VSR-I VMs, use **<vcpupin>** to meet the following CPU pinning guidelines.

CPU Pinning Guidelines
- Each vCPU should be pinned.
- All the vCPUs of a single VM should be pinned to the physical CPUs of a single NUMA node.
- If hyper-threading is enabled on the host machine, then the first two vCPUs must be pinned to sibling threads of the same pCPU core, the next two vCPUs must be pinned to sibling threads of some other pCPU core, and so on.

It is also recommended to include the **<emulatorpin>** element in **<cputune>**. The **<emulatorpin> cpuset** attribute should be set to an available CPU outside of the CPU set allocated to the VSR VM.

The following output shows a **<cputune>** element for a VSR-I with 4 vCPUs, pinned to four physical CPUs that are associated with NUMA node 1.

```
<cputune>
        <vcpupin vcpu='0' cpuset='6'/>
        <vcpupin vcpu='1' cpuset='7'/>
        <vcpupin vcpu='2' cpuset='8'/>
        <vcpupin vcpu='3' cpuset='9'/>
        <emulatorpin cpuset='5'/>
</cputune>
```

5.2.1.6   Numatune

Include the **<numatune>** element to tune the performance of a NUMA host. The
<memory> child element allows you to specify the host NUMA nodes from which to
allocate memory for the VM. The **<numatune>** element provides the following
optional attributes:

- **nodeset**

  The **nodeset** attribute allows you to specify a comma-separated list of numbers
  or ranges, with ^ indicating exclusion. The numbers refer to host NUMA nodes.

- **placement**

  The placement attribute accepts a value of either **static** or **auto**. Use **static**
  when specifying a nodeset. When **auto** is specified, **libvirt** ignores the **nodeset**
  attribute and allocates memory in a NUMA-optimized manner based on input
  from the **numad** process.

  The **placement** attribute defaults to the same placement mode of **<vcpu>**, or to
  static if **nodeset** is specified.

- **mode**

  The **mode** attribute indicates the allocation policy.

  When **mode='strict'** is specified, **libvirt** attempts to allocate memory for the VM
  from the NUMA nodes allowed by **nodeset**, but if it cannot do so, the memory
  allocation fails.

  When **mode='preferred'** is specified, **libvirt** attempts to allocate memory for the
  VM from the NUMA nodes allowed by **nodeset**, but if it cannot do so, the
  memory is allocated from other NUMA nodes.

The following output shows a **<numatune>** configuration for a VSR VM that ensures
all memory for the VM will be allocated from NUMA node 1.

```
<numatune>
        <memory mode='strict' nodeset='1'/>
</numatune>
```

5.2.1.7   CPU

The **<cpu>** element specifies CPU capabilities and topology presented to the guest.
The **mode** attribute of **<cpu>** supports the following values:

- **custom**

  In the **custom** mode, specify all the capabilities of the CPU that will be presented
  to the guest.

- **host-model**

  In the **host-model** mode, the model and features of the host CPU are read by **libvirt** just before the VM is started and the guest is presented with almost identical CPU and features.

  If the exact host model cannot be supported by the hypervisor, **libvirt** falls back to the next closest supported model that has the same CPU features. This fallback is permitted by the **<model fallback='allow'/>** element).

- **host-passthrough**

  In the **host-passthrough** mode, the guest CPU is represented as exactly the same as the host CPU, even for features that **libvirt** does not understand.

> **Note:** The **mode='host-model'** with **<model fallback='allow'/>** is recommended for VSR VMs.

The **<topology>** child element specifies three values for the guest CPU topology: the number of CPU sockets, the number of CPU cores per socket, and the number of threads per CPU core.

If the host machine has hyper-threading enabled in the BIOS, then the **<topology>** element for a VSR-I VM must be specified as follows:

- the **sockets** attribute value must be set to "1"
- the **cores** attribute value must be equal to half the number of vCPUs assigned to the VSR VM
- the **threads** attribute value must be set to "2"

> **Caution:** The number of vCPUs allocated to the VM using the **<vcpu>** element (see vCPU) must equal the number of sockets × cores × threads, as specified in the **<topology>** element. If they are not equal, then the hypervisor may not create the VM and, even if the VM is created, no hyper-threading optimizations are applied by VSR.

The **<numa>** child element in the **<cpu>** element creates specific guest NUMA topology. However, this is not applicable to the VSR because the VSR software is not NUMA-aware.

The following output shows a **<cpu>** configuration suitable for a VSR-I VM on a hyper-threaded host machine. The configuration provides the VSR-I with 12 vCPUs.

```
<cpu mode='host-model'>
    <model fallback='allow'/>
    <topology sockets='1' cores='6' threads='2'/>
</cpu>
```

➡️ **Note:** For the VSR VM to properly detect the vCPU layout specified by the **<topology>** element, the ACPI feature must be enabled in the libvirt domain XML file. See Hypervisor Features.

5.2.1.8   Sysinfo

The **<sysinfo>** element presents SMBIOS information to the guest. SMBIOS is divided into three blocks of information (blocks 0 to 2); each block consists of multiple entries. SMBIOS **system** block 1 is most important for the VSR. The SMBIOS **system** block contains entries for the manufacturer, product, version, serial number, UUID, SKU number, and family.

SMBIOS provides a convenient way to pass VSR-specific configuration information from the host to the guest so that it is available to VSR software when it boots. When a VSR VM is started, the VSR software reads the product entry of the SMBIOS system block. If the product entry begins with **TIMOS:** (case insensitive), VSR recognizes the string that follows as containing important initialization information. The string following the **TIMOS:** characters contains one or more attribute-value pairs formatted as follows:

```
attribute1=value1 attribute2=value2 attribute3=value3
```

This pattern continues until all attributes have been specified.

The supported attribute-value pairs and their uses are summarized in Table 5.

Table 5   **VSR Boot Parameters in SMBIOS Product Entry**

Attribute Name	Valid Values	Description
address	<ip-prefix>/<ip-prefix-length>@active where: • <ip-prefix>: an IPv4 or IPv6 prefix • <ip-prefix-length>: 1-128	Sets a management IP address.
static-route	<ip-prefix>/<ip-prefix-length>@<next-hop-ip> where: • <ip-prefix>: an IPv4 or IPv6 prefix • <next-hop-ip>: an IPv4 or IPv6 address	Adds a static route for management connectivity. Static default routes (0/0) are not supported.

Table 5   **VSR Boot Parameters in SMBIOS Product Entry (Continued)**

Attribute Name	Valid Values	Description
license-file	<file-url> where: • <file-url>: <cflash-id/><file-path> or • ftp:// <login>:<password>@<remote-host/><file-path> or • tftp:// <login>:<password>@<remote-host/><file-path> <cflash-id>: cf1: | cf1-A: | cf1-B: | cf2: | cf2-A: | cf2-B: | cf3: | cf3-A: | cf3-B:	Specifies the local disk or remote FTP/TFTP location of the license file.
primary-config	<file-url> where: • <file-url>: <cflash-id/><file-path> or • ftp:// <login>:<password>@<remote-host/><file-path> or • tftp:// <login>:<password>@<remote-host/><file-path> <cflash-id>: cf1: | cf1-A: | cf1-B: | cf2: | cf2-A: | cf2-B: | cf3: | cf3-A: | cf3-B:	Specifies the local disk or remote FTP/TFTP location of the primary configuration file.
chassis	VSR-I	Specifies the logical chassis type. For an integrated model system, the chassis type is VSR-I.
slot	A	Specifies the logical slot number of the VSR card indicated by the card attribute.
card	cpm-v	Specifies the logical card type. In combination with the chassis value, the card value indicates the type of VSR VM.

Table 5   **VSR Boot Parameters in SMBIOS Product Entry (Continued)**

Attribute Name	Valid Values	Description
mda/n n=1..4	m20-v, isa-aa-v, isa-bb-v, isa-tunnel-v	Specifies the logical MDA types that are logically equipped in the indicated card. Up to four MDAs are supported per card, depending on the chassis or card type combination.
control-cpu-cores	1 to 16	Specifies the number of physical CPU (pCPU) cores to allocate to the control plane. These pCPU cores are reserved for control and management tasks. By default, only one control CPU core is allocated to a VM. See Allocation of vCPUs for Control and Management Tasks for more information.
system-base-mac	hh:hh:hh:hh:hh:hh	Specifies the first MAC address in a range of 1024 contiguous values to use as chassis MACs. The default is the same for all VSRs and should be changed so that each VSR has a unique, non-overlapping range.
vsr-deployment-model	• route-reflector • queue-scale • high-packet-touch	Specifies an application profile for the VSR VM so that optimizations specific to that profile can be applied. The **route-reflector** profile is recommended when deploying the VSR as a control-plane BGP RR. If N is the number of vCPUs allocated to the VM, then this SMBIOS directive is equivalent to manually assigning the **control-cpu-cores** SMBIOS attribute the value N-1 (if the host is not hyper-threaded) or N/2-1 (if the host is hyper-threaded). The **queue-scale** profile is recommended when a VSR is used in a system configured as a virtualized BNG or other similar system type that requires deep buffering. In this profile the maximum supported MTU size is 2048 bytes. The **high-packet-touch** profile is recommended when a VSR is used in a system configured as an IPSec security gateway or for Application Assurance. This profile has an effect only if the VSR VM detects that it is running on a hyper-threaded host machine. In this case, it runs two vFP worker tasks on the same pCPU core rather than just one vFP worker task per pCPU core (using just one hyper-thread and leaving the other idle).

Table 5   **VSR Boot Parameters in SMBIOS Product Entry (Continued)**

Attribute Name	Valid Values	Description
hyperthreading	1	By setting this attribute value to "1", the VSR VM is forced to assume that it is running on a hyper-threaded host machine. This setting is not required nor advised on Linux KVM hosts. It should be used only when the VSR is known to be deployed on a VMware ESXi host that has hyper-threading enabled. VSR VMs cannot detect the hyper-threading state of ESXi host machines, so a manual setting, using this attribute, is required.

The following **<sysinfo>** output shows a configuration suitable for a VSR-I VM. Replace the attribute values in the SMBIOS **product** entry string with values appropriate for your deployment.

```
<sysinfo type='smbios'>
   <system>
     <entry name='product'>TIMOS:slot=A chassis=VSR-I card=cpm-v mda/1=m20
      v control-cpu-cores=2 system-base-mac=de:ad:be:ef:00:01 address=192.0.2.1
      24@active primary-config=ftp://user01:pass@10.0.0.1/home/user01/vsr-i
      config.cfg  license-file=ftp://user01:pass@10.0.0.1/home/user01/license.txt<
      entry>
   </system>
</sysinfo>
```

5.2.1.9   OS

The **<os>** element provides information about the guest OS to the hypervisor. It contains a **<type>** element that specifies the guest operating system type. For VSR VMs, the **<type>** element must specify **hvm**, which means that the guest OS is designed to run on bare metal and requires full virtualization.

The **arch** attribute of the **<type>** element specifies the CPU architecture that is presented to the guest. For VSR VMs, specify **arch=x86_64** to allow the VSR software to take advantage of 64-bit instructions and addressing.

The **machine** attribute of the **<type>** element specifies how QEMU should model the motherboard chipset in the guest system. For VSR VMs, specify **machine='pc'**, which is an alias for the latest I440FX/PIIX4 architecture supported by the hypervisor when the VM is created. The I440FX is a (1996 era) motherboard chipset that combines both Northbridge (memory controller) and Southbridge (IO devices) functionality.

If **machine='q35'** is specified, QEMU-KVM can also emulate a Q35 chipset. Q35 is a relatively modern (2009 era) chipset design; it separates the Northbridge controller (MCH) from the Southbridge controller (ICH9) and provides the guest with advanced capabilities such as IOMMU and PCI-E.

Although the I440FX emulation is the older machine type, it is the more mature and hardened option and is recommended by Nokia.

The **<os>** element also contains the **<smbios>** child element that you must include in the configuration of VSR VMs. Set the **mode** attribute to "**sysinfo**", which allows you to pass the information specified in the **<sysinfo>** element (including the **product** entry) to the VSR guest. If you are using configDrive, set the **mode** attribute to "**emulate**".

The **<os>** element can also include one or more **<boot>** child elements. The **dev** attribute of each **<boot>** element specifies a device such as 'hd' (hard drive), 'fd' (floppy disk), 'cdrom', or 'network', which indicates that the guest should load its OS from this device. The order of multiple **<boot>** elements determines the boot order. For VSR VMs, always boot from the '**hd**' device that VSR translates to its CF3 disk.

The following **<os>** output shows element configuration suitable for VSR VMs of all types.

```
<os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
</os>
```

5.2.1.10 Hypervisor Features

The **<features>** element allows certain CPU and machine features to be toggled on or off.

The **<acpi>** child element of **<features>** toggles support of ACPI. The ACPI feature must be enabled for all VSR VMs. It is critical when the VSR VM runs on any host machine with a physical CPU having 18 or more cores, and it is important for VSR-I VMs because it provides the method for the VSR software to learn about the CPU topology of the KVM host machine so that it can apply hyper-threading optimizations if needed.

The following output shows the required configuration of the **<features>** element.

```
<features>
 <acpi/>
</features>
```

5.2.1.11   Clock

The **\<clock\>** element controls specific aspects of timekeeping within the guest. Each guest must initialize its clock to the correct time-of-day when booting and update its clock accurately as time passes.

The **offset** attribute of **\<clock\>** controls how the of the time-of-day clock of the guest is initialized at bootup. For VSR VMs, the **offset** attribute value should be set to **utc**, which enable the host and guest to belong to different time zones, if required.

The VSR and other guests update the time-of-day clock by counting ticks of virtual timer devices. The hypervisor injects ticks to the guest in a manner that emulates traditional hardware devices, for example, the Programmable Interrupt Timer (PIT) and CMOS Real Time Clock (RTC). Each virtual timer presented to the guest is defined by a **\<timer\>** sub-element of **\<clock\>**. The **name** attribute of **\<timer\>** specifies the device name (for example, 'pit', 'rtc' or 'hpet'), the **present** attribute indicates whether the timer should be made available to the guest, and the **tickpolicy** attribute controls the action taken when the hypervisor (QEMU) discovers that it has missed a deadline for injecting a tick to the guest. A **tickpolicy** value set to 'delay' means the hypervisor should continue to delay ticks at the normal rate, with a resulting slip in guest time relative to host time. A **tickpolicy** value set to 'catchup' means the hypervisor should deliver ticks at a higher rate to compensate for the missed tick.

The following **\<clock\>** output shows element configuration suitable for VSR VMs.

```
<clock offset='utc'>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='rtc' tickpolicy='catchup'/>
    <time name='hpet' present='no'/>
</clock>
```

5.2.1.12   Devices

Use the **\<devices\>** element to add various devices to the VM, including hard drives, CD-ROMs, network interfaces, and serial console ports.

The **\<devices\>** element requires that the file path of the program used to emulate the devices must be specified in the **\<emulator\>** child element. On Centos and Red Hat hosts the emulator is a binary called **qemu-kvm**. On Ubuntu hosts, the emulator is called **qemu-system-x86_64**.

VSR VM configuration information is provided about the following device types:

- Disk Devices

- Host Devices and PCI Passthrough
- Network Interfaces
- Guest vNIC Mapping in VSR VMs
- Console and Serial Ports

Note: A NIC swap may result in Linux interface name changes, MAC address changes and/
or PCI bus address changes.

5.2.1.12.1   Disk Devices

The **<disk>** child element of the **<devices>** element allows you to add up to four
disks to a VSR VM.

The **type** attribute of the **<disk>** element specifies the underlying source for each
disk. The only supported value for VSR VMs is **type='file'**, which indicates that the
disk is a file residing on the host machine.

The **device** attribute of the **<disk>** element configures the representation of the disk
to the guest OS. The supported values for VSR VMs are **device='disk'** and
device='cdrom'. When **device='disk'** is specified, QEMU-KVM attaches a hard
drive to the guest VM and VSR interprets this as a Compact Flash (CF) storage
device. When **device='cdrom'** is specified, QEMU-KVM attaches a CD-ROM to the
guest VM, and VSR mounts it as a read-only configDrive if the volume label is config-
2 (case insensitive).

The optional **<driver>** child element of the **<disk>** element provides additional
details about the back-end driver. For VSR VMs, set the **name** attribute to **'qemu'**
and the **type** attribute to **'qcow2'**. These two attributes specify that the disk image
has the QCOW2 format.

When you download the VSR software, the zip file contains a QCOW2 disk image,
which is a file that represents the VSR software on a hard disk; any VSR VM can be
booted from this disk image. QCOW2 is a disk image format for QEMU-KVM VMs
that uses thin provisioning (that is, the file size starts small and increases in size only
as more data is written to disk). It supports snapshots, compression, encryption, and
other features.

The optional **cache** attribute of the **<driver>** element controls the caching mechanism of the hypervisor. A value set to '**writeback**' offers high performance but risks data loss (for example, if the host crashes but the guest believes the data was written). For VSR VMs, it is recommended to set **cache='none'** (no caching) or **cache='writethrough'** (writing to cache and to the permanent storage at the same time).

The mandatory **<source>** child element of the **<disk>** element indicates the path (for disks where **type='file'**) to the QCOW2 file used to represent the disk.

Note: The recommended storage location for QCOW2 disk image files is the **/var/lib/libvirt/images** directory; storing disk images in other locations may cause permission issues.

The mandatory **<target>** child element of the **<disk>** element controls how the disk appears to the guest in terms of bus and device. The **dev** attribute should be set to a value of **'hda'**, **'hdb'** or **'hdc'**. A value of **'hda'** is the first IDE hard drive; it maps to CF3 on VSR-I VMs. A value of **'hdb'** is the second IDE hard drive; it maps to CF1 on VSR-I VMs. A value of **'hdc'** is the third IDE hard drive; it maps to CF2 on VSR-I VMs. The **bus** attribute of the **<target>** element should be set to **'virtio'** for VSR virtual disks.

Each VSR VM must be provided with a "hda" hard disk that contains the VSR software images. Each virtual disk of each VSR VM should have be provided with its own, independent QCOW2 file.

The following **<disk>** element configuration output provides a VM with a CF3 device.

```
<disk type='file' device='disk'>
     <driver name='qemu' type='qcow2' cache='none'/>
     <source file='/var/lib/libvirt/images/vsr-i.qcow2'/>
     <target dev='hda' bus='virtio'/>
</disk>
```

ConfigDrive

A ConfigDrive is a CD-ROM disk attached to the guest. It contains user-data and meta-data files that are read by the guest at bootup to initialize settings (for example, IP address). The ConfigDrive can be built manually (using the **genisoimage** command), but it is typically generated automatically by a VIM such as OpenStack. The following conditions must be true for the VSR to recognize a CD-ROM as a ConfigDrive:

1. The volume label must be **config-2** (case insensitive).

2. The file system must be ISO9660 or VFAT

3. An **/openstack/2013_10_17/user_data** file must be present

When a VSR VM detects a ConfigDrive, it reads the **user_data** file in the **openstack/2013_10_17** directory. If the file contains a string beginning with "TIMOS:" (without the quotes, case insensitive), the VSR handles the string in the same way it would handle an identical string in an SMBIOS product entry. ConfigDrive provides an alternate method of passing the boot parameters to the VSR software when creating a VSR VM using OpenStack (Nova); see Table 5 for more information about VSR boot parameters.

The following ConfigDrive configuration shows XML formats for ISO and VFAT.

```
<disk type='file' device='cdrom'>
    <driver name='qemu' type='raw' cache='none'/>
    <source file='/<TESTBED>/images/flexibed/<VM_NAME>/configDrive.iso'/>
    <target dev='hdd' bus='ide'/>
</disk>
<disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none'/>
    <source file='/<TESTBED>/images/flexibed/<VM_NAME>/configDrive.qcow2'/>
    <target dev='hdd' bus='virtio'/>
</disk>
```

5.2.1.12.2 Host Devices and PCI Passthrough

The **<hostdev>** child element of the **<devices>** element allows host USB, PCI, and SCSI devices to be passed through directly to the guest. For VSR VMs, the main benefit of the **<hostdev>** element is that it supports PCI passthrough for physical NIC ports. See Using PCI Passthrough for information about the benefits of PCI passthrough and the host prerequisites.

For PCI passthrough, the **<hostdev>** element includes **mode='subsystem'** and **type='pci'** attributes.

In addition, the **managed** attribute should be set to **managed='yes'**, which ensures that the PCI device (physical NIC port) is automatically detached from the host before it is attached to the guest at bootup, and then re-attached to the host when the guest is powered down. If **managed='yes'** is not set, you must use the **virsh nodedev-detach** and **virsh nodedev-reattach** commands when the guest is started and stopped.

For PCI passthrough, the **<hostdev>** element must contain a **<source>** sub-element, that contains an **<address>** sub-element. Each attribute of the **<address>** element (**domain**, **bus**, **slot**, and **function**) takes a hexadecimal value (preceded by 0x) and together they indicate the exact PCI device (in the host PCI addressing domain) to pass through to the guest.

The **<hostdev>** element should also contain a **<rom>** element, with the **bar** attribute set to '**off**'. The **<rom bar='off'/>** element prevents QEMU from presenting the option ROM of the physical NIC to the guest. If the option ROM is not blocked, it can result in delays and degraded performance in the VSR guest.

The following methods are available to obtain PCI addresses of NIC ports:

1. sudo lshw -class network -businfo
2. ethtool -i <interface-name>
3. readlink -f /sys/class/net/<interface-name>/device

The following **<hostdev>** configuration shows PCI passthrough of the NIC port associated with the PCI address 0000:02:01.0.

```
<hostdev mode=subsystem' type='pci' managed='yes'>
    <source>
        <address domain='0x0000' bus='0x02' slot='0x01' function='0x0'/>
    </source>
    <rom bar='off'/>
</hostdev>
```

5.2.1.12.3   Network Interfaces

The **<interface>** sub-element of the **<devices>** element allows you to add up to 16 virtual NIC ports to a VSR VM. The **type** attribute of **<interface>** supports several values, including:

- type='direct'
- type='bridge'
- type='hostdev'
- type='network'
- type='vhostuser'

The following child elements of **<interface>** are common to most interface types:

- **<mac>**: Contains an **address** attribute that indicates the MAC address of the guest vNIC port.
- **<model>**: Contains a **type** attribute that indicates the NIC model presented to the guest.

  The default value for **type** is '**virtio**', which indicates that the guest should use its VirtIO driver for the network interface.
- **<driver>**: Contains several attributes corresponding to tunable driver settings.

The **queues** attribute, when used in conjunction with the **<model type='virtio'/>** element, enables multi-queue VirtIO in the guest.

➡️ **Note:** The VSR does not support multi-queue VirtIO.

- **<address>**: Specifies the guest PCI address of the vNIC interface when the **type='pci'** attribute is included.

  The other attributes required to specify a PCI address are: **domain** (0x0000), **bus** (0x00-0xff), **slot** (0x00-0x1f), and **function** (0x0-0x7).

  If the **<address>** element is not included, the hypervisor assigns an address automatically as follows: the first interface defined in the **libvirt** domain XML has the lowest PCI address, the next one has the next-lowest PCI address, and so on.

  The VSR maps vNIC interfaces to its own set of interfaces based on the order of the vNIC interfaces, from lowest to highest PCI address; this should be considered when you change the PCI address of a vNIC interface. See Guest vNIC Mapping in VSR VMs for information about how the VSR maps vNIC interfaces.

- **<target>**: Specifies the name of the Linux tun device representing the vNIC interface in the host.

➡️ **Note:** The **<target>** element does not need to be configured with VSR VMs.

type='direct'

The **<interface>** element with **type='direct'** allows you to create a direct connection between the guest vNIC port and a host physical NIC port or SR-IOV virtual function. The interconnection uses a MACVTAP driver in the Linux host. The I/O performance is significantly less than bare-metal, but live migration can be supported when the interconnection is used with an SR-IOV virtual function.

Specifying **type='hostdev'** with an SR-IOV virtual function does not permit live migration, as described in type='hostdev'.

To connect a guest vNIC port to an SR-IOV virtual function using the MACVTAP driver, include a **<source>** sub-element with the **dev** attribute that indicates the interface name of the host VF interface and **mode='passthrough'**. The following configuration shows 'enp133s0f0' as the host VF interface name.

```
<interface type='direct'>
      <source dev='enp133s0f0' mode='passthrough'/>
      <model type='virtio'>
</interface>
```

type='bridge'

The **<interface>** element with **type='bridge'** specifies that the guest vNIC port should be connected to a vSwitch or Linux bridge in the host. The interconnection uses the Vhost-Net back end driver when the **<model type='virtio'/>** element is included. The I/O performance of this configuration is significantly less than bare-metal.

To use a Linux bridge named brX, include a **<source>** sub-element with a **bridge='brX'** attribute, as shown in the following configuration.

```
<interface type='bridge'>
        <source bridge='br0'/>
        <model type='virtio'>
</interface>
```

To use a standard (non-DPDK) OpenvSwitch bridge named brX, include a **<source>** sub-element with a **bridge='brX'** attribute. In addition, include a **<virtualport type='openvswitch'/>** element, as shown in the following configuration.

```
<interface type='bridge'>
      <source bridge='br1'/>
      <virtualport type='openvswitch'/>
      <model type='virtio'>
</interface>
```

type='hostdev'

The **<interface>** element with **type='hostdev'** is the recommended method of connecting a guest vNIC interface to a virtual function of an SR-IOV NIC port (PF). This method offers bare-metal performance levels and allows network parameters (such the MAC address or VLAN tag) to be assigned to the VF prior to PCI assignment to the guest.

➡ **Note:** To use SR-IOV, the host must have the correct BIOS and kernel boot settings. See Using SR-IOV for more information.

To connect a guest vNIC port to an SR-IOV virtual function using the **<interface type=hostdev>** element, include a **<source>** sub-element containing an **<address type='pci'>** element. The **<address>** element indicates the host PCI address of the VF in the form of **domain** (0x0000), **bus** (0x00-0xff), **slot** (0x00-0x1f), and **function** (0x0-0x7) attributes. To obtain the PCI addresses of all virtual functions associated with a physical NIC port, use the following command:

ls -l /sys/class/net/<interface-name>/device/virtfn*

The following configuration shows an **<interface type='hostdev'>** element that connects a guest vNIC interface to the VF with host PCI address 0000:81:10.1. The VF is associated with both a MAC address and a VLAN ID in this example.

```
<interface type='hostdev' managed='yes'>
    <mac address='52:54:00:81:10:01'/>
    <source>
        <address type='pci' domain='0x0000' bus='0x81' slot='0x10' function='0x1'/>
    </source>
    <vlan>
        <tag id='100'/>
    </vlan>
</interface>
```

type='network'

In addition to the XML definition of domains (or VMs), **libvirt** also supports XML definition of networks, which are managed the same way as domains. To define networks, create an XML file for each network, then start each network with a **virsh net-create <***filename.xml***>** command. The **virsh net-autostart <***network-name***>** command ensures that the network is always available at startup. The **virsh net-list** command shows all of the available networks. The **virsh net-destroy** <*network-name*> command removes a network.

To attach a guest vNIC interface to a libvirt-defined network, provide the VM an **<interface type='network'>** element. The **<source>** sub-element is required with its **network** attribute is set to the name of the libvirt-defined network.

The **type='network'** interface type is useful when you want to assign an SR-IOV virtual function to a guest vNIC. However, **libvirt** should be allowed to choose the next available VF from a pool instead of binding the guest vNIC to a specific VF with a specific host PCI address. The pool method makes it simpler to manage the VF resources in one host and to move guests between hosts. When a guest moves from one host to another, a hard-coded PCI address may not correspond to an available VF. Libvirt allows you to define a network that conceptually contains a pool of VF resources.

The following configuration shows a network XML file that assigns all VFs associated with the eth3 physical function (PF) to a pool.

```
<network>
    <name>eth3-vf-pool</name>
    <forward mode='hostdev' managed='yes'>
          <pf dev='eth3'/>
    </forward>
</network>
```

The following configuration shows an **<interface>** used to bind a guest vNIC interface to any available VF of eth3.

```
 <interface type='network'>
     <source network='eth3-vf-pool'/>
</interface>
```

type='vhostuser'

The **<interface>** element with **type='vhostuser'** specifies that the guest vNIC port should be connected to an DPDK-accelerated OVS bridge (an OVS bridge with **datapath_type=netdev**). The guest uses a VirtIO driver and the host implements the back-end Vhost driver in userspace. The I/O performance of this configuration is significantly better than non-DPDK OVS.

The following output shows an <i**nterface type='vhostuser'>**configuration that binds a guest vNIC interface to a dpdkvhostuser interface 'vhost-user0' of a OVS-DPDK bridge.

```
<interface type='vhostuser'>
    <source type='unix' path='/usr/local/var/run/openvswitch/vhost
     user0' mode='client'/>
    <model type='virtio'/>
</interface>
```

> **Note:** The **<model type='virtio'/>** element is mandatory for a '**vhostuser**' interface. Change the name of the dpdkvhostuser interface and the path to the associated socket, if they differ in your configuration.

5.2.1.12.4   Guest vNIC Mapping in VSR VMs

This section describes the relationship between a network interface defined in the **libvirt** XML for a VSR VM and its use by the VSR software.

In the current release, each VSR VM supports a maximum of 16 vNIC interfaces. The
VSR software puts the defined interfaces in ascending order of (guest) PCI address,
which is also the order they typically appear (top to bottom) in the domain XML file.

The order of the defined interfaces and the VSR VM type determines the use of each
interface by the VSR software. The VSR interface mapping information is
summarized in Table 6.

Table 6   **VSR-I Interface Mapping**

Order (By Guest PCI Address)	VSR Software Use	Supported Interface Types
1	Management port (A/1)	**type='direct'** with **<model type='virtio'/>** **type='bridge'** with **<model type='virtio'/>**
2	MDA port (1/1/1)	See Host Devices and PCI Passthrough and Network Interfaces for more information about the following interface types: • type='direct' • type='bridge' • type='hostdev' • type='network' • type='vhostuser'
3	MDA port (1/1/2)	See Host Devices and PCI Passthrough and Network Interfaces for more information about the following interface types: • type='direct' • type='bridge' • type='hostdev' • type='network' • type='vhostuser'
...	-	-

Table 6   **VSR-I Interface Mapping (Continued)**

| Order
(By Guest PCI Address)	VSR Software Use	Supported Interface Types
16	MDA port (1/1/15)	See Network Interfaces for more information about the following interface types: • type='direct' • type='bridge' • type='hostdev' • type='network' • type='vhostuser'

5.2.1.12.5   Console and Serial Ports

The **<console>** sub-element in the **<devices>** element allows you to add a console port to a VSR VM. As it does on physical routers, the console port on a VSR VM provides interactive access to CLI.

There are several methods for creating and accessing a VSR console port. The first method is to bind the console port to a TCP socket opened by the host. To access the console, establish a Telnet session with the host, using the port number of the TCP socket. The following example shows a configuration for this method:

```
<console type='tcp'>
      <source mode='bind' host='0.0.0.0' service='4000'/>
      <protocol type='telnet'/>
      <target type='virtio' port='0'/>
</console>
```

The second method is to bind the console port to an emulated serial port. In this case, the **virsh console <***domain-name***>** command is used to access the console. The following example shows a configuration for this method:

```
<serial type='pty'>
      <source path='/dev/pts/1'/>
      <target port='0'/>
      <alias name='serial0'/>
</serial>
<console type='pty' tty='/dev/pts/1'>
      <source path='/dev/pts/1'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
</console>
```

5.2.1.13 Seclabel

The **<seclabel>** element controls the generation of security labels required by
security drivers such as SELinux or AppArmor. These are not supported with VSR
VMs and therefore you must specify <seclabel type='none'> in the domain XML.

5.2.2 Example Libvirt Domain XML

The following example shows a libvirt domain XML configuration for a VSR-I VM.
Substitute the correct values for your configuration.

```
<domain type='kvm'>
      <name>vsr-i-01</name>
      <uuid>ab9711d2-f725-4e27-8a52-ffe1873c102f</uuid>
      <memory unit='G'>6</memory>
      <memoryBacking>
          <hugepages>
              <page size='1' unit='G' nodeset='0'/>
          </hugepages>
          <nosharepages/>
      </memoryBacking>
      <vcpu>4</vcpu>
      <cputune>
          <vcpupin vcpu='0' cpuset='6'/>
          <vcpupin vcpu='1' cpuset='7'/>
          <vcpupin vcpu='2' cpuset='8'/>
          <vcpupin vcpu='3' cpuset='9'/>
          <emulatorpin cpuset='5'/>
      </cputune>
      <numatune>
          <memory mode='strict' nodeset='1'/>
      </numatune>
          <cpu mode='host-model'>
              <model fallback='allow'/>
              <topology sockets='1' cores='4' threads='1'/>
      </cpu>
      <sysinfo type='smbios'>
          <system>
              <entry name='product'>TIMOS:slot=A chassis=VSR-I card=cpm-v mda/1=m20
               v control-cpu-cores=2 system-base
               mac=de:ad:be:ef:00:01 address=192.0.2.1/24@active primary-config=ftp:
               /user01:pass@10.0.0.1/home/user01/vsr-i/config.cfg  license-file=ftp:
               /user01:pass@10.0.0.1/home/user01/license.txt</entry>
          </system>
      </sysinfo>
      <os>
          <type arch='x86_64' machine='pc'>hvm</type>
          <boot dev='hd'/>
          <smbios mode='sysinfo'/>
      </os>
      <features>
          <acpi/>
      </features>
```

```
<clock offset='utc'>
   <timer name='pit' tickpolicy='delay'/>
   <timer name='rtc' tickpolicy='catchup'/>
   <time name='hpet' present='no'/>
</clock>
<devices>
   <emulator>/usr/bin/qemu-kvm</emulator>
   <disk type='file' device='disk'>
       <driver name='qemu' type='qcow2' cache='none'/>
       <source file='/var/lib/libvirt/images/vsr-i-01.qcow2'/>
       <target dev='hda' bus='virtio'/>
   </disk>
   <interface type='bridge'>
       <source bridge='br0'/>
       <virtualport type='openvswitch'/>
       <model type='virtio'>
   </interface>
   <interface type='hostdev' managed='yes'>
       <mac address='52:54:00:81:10:01'/>
       <source>
           <address type='pci' domain='0x0000' bus='0x81' slot='0x10' function
            ='0x1'/>
       </source>
       <vlan>
           <tag id='100'/>
       </vlan>
   </interface>
   <serial type='pty'>
       <source path='/dev/pts/1'/>
       <target port='0'/>
       <alias name='serial0'/>
   </serial>
   <console type='pty' tty='/dev/pts/1'>
       <source path='/dev/pts/1'/>
       <target type='serial' port='0'/>
       <alias name='serial0'/>
   </console>
</devices>
<seclabel type='none'/>
</domain>
```

5.2.3   Verifying VSR Installation on Linux KVM Hosts

5.2.3.1   Overview

This section describes the basic procedures for verifying the VSR VM (VM)
installation on a Linux KVM host. Common problems that you may encounter are
highlighted and possible solutions to resolve these issues are provided.

Note: This section assumes that libvirt tools were used to deploy the VSR.

5.2.3.2 Verifying Host Details

Successful installation of a VSR VM requires the host machine to be set up properly. Use the commands described in this section to show host information for Linux systems (running Centos or Red Hat).

5.2.3.2.1 General System Information

To show the Linux kernel version, enter the following:

```
uname -a ↵
```

To verify that the Linux kernel is 64-bit, enter the following:

```
uname -m ↵
```

The command output should be x86_64.

5.2.3.2.2 Linux Distribution Type

To show the type of Linux distribution and version, enter the following:

```
lsb_release -a ↵
```

Note: Depending on your Linux distribution, you may have to install a package such as **redhat-lsb-core** to use this command.

5.2.3.2.3 PCI Devices

To view all PCI devices, enter the following:

```
lspci ↵
```

A partial sample output of the command is as follows:

```
[user@host ~]# lspci
04:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
05:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
06:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
07:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection
```

The first entry indicates that there is a PCI device attached to bus 04, with device ID
00 and function 0 (04:00.0) and that it is an 82574L Gigabit Ethernet controller made
by Intel Corporation.

To view PCI device details, including capabilities such as the maximum bus speed
and the number of lanes (for example x4), enter the following:

```
lspci -vvv ↵
```

5.2.3.2.4   CPU Processor Information

To view details about all the CPU processors available to the host, enter the
following:

```
cat /proc/cpuinfo ↵
```

When hyper-threading is enabled on Intel CPUs, every hyper-thread appears as a
separate processor, as shown in the following partial sample output:

```
[user@host ~]# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz
stepping        : 4
cpu MHz         : 2593.614
cache size      : 15360 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse
6 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm cons
ant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulq
q dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 sse4_2 x2ap
c popcnt aes xsave avx f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dts tpr_sh
dow vnmi flexpriority ept vpid fsgsbase smep erms
bogomips        : 5187.22
clflush size    : 64
```

```
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

➡ **Note:** Similar output can be obtained using the **lscpu** and **lscpu --all --extended** commands.

To run VSR VMs, the **cpu family** value must be 6 (Intel) and the **model** should be greater than or equal to 42 (in most cases). In addition, several CPU flags are critical for the VSR and must be passed through to the guest. These include:

- x2apic—support for an advanced interrupt controller introduced with Intel Nehalem processors

  Support for the x2APIC is mandatory; the flag must not be emulated
- lm—long mode, indicating a 64-bit CPU, which is necessary to support 64-bit guests
- vmx—support for Intel virtualization technologies such as VT-d/VT-x

5.2.3.2.5   Host Memory

To view details about the host memory, enter the following:

```
cat /proc/meminfo ↵
```

The following is a partial sample output of this command:

```
[user@host ~]# cat /proc/meminfo
MemTotal:      16406144 kB
MemFree:        9442676 kB
MemAvailable:  11272708 kB
Buffers:         648220 kB
Cached:         1352744 kB
SwapCached:           0 kB
Active:         4898888 kB
Inactive:       1723664 kB
```

The **MemFree** value must be at least 4194304 kB if you want to create another VSR VM on this host.

5.2.3.2.6   Host Capability

To show summary information about the host and its virtualization capabilities, enter the following:

```
virsh capabilities ↵
```

Note: The **libvirt** package must be installed to use this command on the host.

The command output must confirm that the system is capable of supporting guests with the x86_64 architecture (64-bit guests).

5.2.3.2.7   QEMU and libvirt Information

To show **libvirt** and QEMU version information, enter the following:

```
virsh version ↵
```

Note: The **libvirt** package must be installed to use this command on the host.

5.2.3.2.8   Loaded Modules

To list all the kernel modules installed on the host, enter the following:

```
lsmod ↵
```

Some key modules are: **bridge**, **kvm**, **kvm_intel**, **vhost_net**, **tun**, **macvtap** and **openvswitch**.

5.2.3.2.9   Host Virtualization Setup

To check that dependencies for virtualization are installed correctly on the host, enter the following:

```
virt-host-validate ↵
```

The following is a sample output of the command:

```
[user@host ~]# virt-host-validate
  QEMU: Checking for hardware virtualization                                  : PASS
  QEMU: Checking for device /dev/kvm                                          : PASS
  QEMU: Checking for device /dev/vhost-net                                    : PASS
  QEMU: Checking for device /dev/net/tun                                      : PASS
```

```
        LXC: Checking for Linux >= 2.6.26                                    : PASS
[user@host ~]#
```

5.2.3.3   Verifying the Creation of VMs

Before attempting to log in to a VSR system and to check for successful boot of its
VMs, ensure that the VMs have been created as expected on all the host machines.

If **libvirt** is used, view the list of VMs on a specific host by entering the following
command:

virsh list ↵

The following is a sample output of this command:

```
[user@host ~]# virsh list --all
 Id    Name                                State
--------------------------------------------------
 1     VSR1                                running
 2     VSR2                                running
 3     IOM1                                running
```

Because each QEMU-KVM VM is a process with two or more threads, you can also
use a sequence of commands, such as the following, to get more details about a
running VM:

```
[user@host ~]# ps -ef | grep VSR1
qemu      6304      1  5 Sep10 ?        05:03:50 /usr/libexec/qemu-kvm.real -
name VSR1 -S -machine rhel6.0.0, accel=kvm, usb=off -
cpu SandyBridge, +erms, +smep, +fsgsbase, +rdrand, +f16c, +osxsave, +pcid, +pdcm, +x
tpr, +tm2, +est, +smx, +vmx, +ds_cpl, +monitor, +dtes64, +pbe, +tm, +ht, +ss, + acpi
, + ds, +vme -m 6144 -realtime mlock=off -smp 2, sockets=2, cores=1, threads=1 -
uuid nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn -nographic -no-user-config -nodefaults -
chardev socket, id=charmonitor, path=/var/lib/libvirt/qemu/
VSR1.monitor, server, nowait -mon chardev=charmonitor, id=monitor, mode=control -
rtc base=utc -no-kvm-pit-reinjection -no-shutdown -no-acpi -boot strict=on -
device piix3-usb-uhci, id=usb, bus=pci.0, addr=0x1.0x2 -device virtio-serial-
pci, id=virtio-serial0, bus=pci.0, addr=0x7 -drive file=/path/
disk1.qcow2, if=none, id=drive-virtio-disk0, format=qcow2, cache=none -
device virtio-blk-pci, scsi=off, bus=pci.0, addr=0x8, drive=drive-virtio-
disk0, id=virtio-disk0 -netdev tap, fd=23, id=hostnet0, vhost=on, vhostfd=24 -
device virtio-net-
pci, netdev=hostnet0, id=net0, mac=nn:nn:nn:nn:nn:nn, bus=pci.0, addr=0x3, bootindex
=1 -netdev tap, fd=25, id=hostnet1, vhost=on, vhostfd=26 -device virtio-net-
pci, netdev=hostnet1, id=net1, mac=nn:nn:nn:nn:nn:nn, bus=pci.0, addr=0x4 -
chardev socket, id=charconsole0, host=0.0.0.0, port=2500, telnet, server, nowait -
device virtconsole, chardev=charconsole0, id=console0 -device virtio-balloon-
pci, id=balloon0, bus=pci.0, addr=0x6

[user@host ~]# ps -T 6304
root@vsr_hyp]# ps -T 6304
```

```
  PID  SPID TTY        STAT    TIME COMMAND
 6304  6304 ?          Sl     0:19 /usr/libexec/qemu-kvm.real -name VSR1 -S -
machine rhel6.0.0,accel=k
 6304  6310 ?          Sl   169:47 /usr/libexec/qemu-kvm.real -name VSR1 -S -
machine rhel6.0.0,accel=k
 6304  6311 ?          Sl   134:52 /usr/libexec/qemu-kvm.real -name VSR1 -S -
machine rhel6.0.0,accel=k
```

These sample command outputs indicate that the VM called VSR1 is running as
process ID 6304 in the host machine. There are three threads associated with this
process.

Obtain a real-time view of the host system impact of all running VMs by entering the
following commands:

top ↵

htop ↵

The following is a sample output of the command:

```
[root@vsr_hyp]# top
top - 14:00:10 up 5 days,  4:44,  2 users,  load average: 4.09, 4.08, 4.09
Tasks: 184 total,   2 running, 182 sleeping,   0 stopped,   0 zombie
%Cpu(s): 51.2 us,  0.1 sy,  0.0 ni, 48.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  16406144 total,  6970448 used,  9435696 free,   649488 buffers
KiB Swap:        0 total,        0 used,        0 free.  1328612 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM    TIME+ COMMAND
 6349 qemu      20   0 4860828 2.445g   5912 S 402.1 15.6  24522:22 qemu-
kvm.real
 6304 qemu      20   0 6736452 0.981g   5916 S   4.7  6.3 305:44.16 qemu-
kvm.real
 6321 qemu      20   0 6736424 0.980g   5916 S   4.3  6.3 280:32.31 qemu-
kvm.real
 6308 root     20  0       0       0      0 S   0.3  0.0   4:45.88 vhost-
6304
 6324 root    20  0     0     0     0 S  0.3  0.0  1:23.12 vhost-6321
```

5.2.3.4   Verifying Host Networking

Different methods can be used to provide network connectivity between the VSR
VMs and external destinations.

Table 7 lists some useful commands to help troubleshoot networking at the host
level.

Table 7 **Host Network Troubleshooting**

Command Syntax	Description
ip -d link show	Shows details of all host network interfaces, including physical NIC ports and logical interfaces, such as vNIC interface constructs on the host
ip link set dev <interface-name> **mtu** <value>	Explicitly sets the MTU (Maximum Transmit Unit) of a host interface
ip addr show	Shows the IP addresses associated with host network interfaces **Note**: In a virtualization environment, many interfaces will not have any IP addresses assigned to them
ip route show	Shows the IP routing table of the host
tcpdump -i <interface name>	Captures packets on the selected interface and outputs them for analysis
brctl show	Shows all the Linux bridges
ovs-vsctl show	Shows all the Open vSwitch bridges
ethtool -S <interface name>	Shows statistics collected by the physical NIC for a selected interface

5.2.3.5   Verifying VSR Installation

After verifying that the VSR VMs have been created successfully on the respective hosts, check the SR OS operating system to verify that it has booted up properly on each VM and that each VSR system is functional. Access to the VSR guests is often required to perform these checks. This is the main reason for adding a serial console port to VSR VMs. With console access, log in to each VSR and perform the checks described in this section.

5.2.3.5.1   Check the Status of the System BOF

The output of this check depends on the SMBIOS text string you used for the VM and the saved BOF configuration. At the prompt, type the following:

```
A:VSR#  show bof ↵
```

The following is a sample output of this command:

```
A:Dut-A# show bof
===============================================================================
BOF (Memory)
===============================================================================
    primary-image    cf3:\timos\
    primary-config   ftp://*:*@[135.121.85.54]/./images/dut-a.cfg
    license-file     ftp://*:*@[135.121.85.54]/./images/vmLICENSE/timos.vsr-all.txt
    address          135.121.82.216/21 active
    address          3000::8779:52d8/117 active
    static-route     128.251.10.0/24 next-hop 135.121.80.1
    static-route     135.0.0.0/8 next-hop 135.121.80.1
    static-route     138.0.0.0/8 next-hop 135.121.80.1
    static-route     172.20.0.0/14 next-hop 135.121.80.1
    static-route     172.31.0.0/16 next-hop 135.121.80.1
    static-route     192.168.85.54/32 next-hop 135.121.85.54
    autonegotiate
    duplex           full
    speed            100
    wait             3
    persist          off
    no li-local-save
    no li-separate
    console-speed    115200
    system-base-mac  fa:ac:ff:ff:10:00
===============================================================================
A:Dut-A#
```

5.2.3.5.2   Check the Chassis Type

Verify that the VSR VM chassis type is set correctly.

➡ **Note:** The **Chassis Topology** field is used to differentiate a 7950 XRS-20 from a 7950 XRS-40, not to differentiate a VSR-I from a VSR-D.

If the chassis type does not match the one encoded in the SMBIOS text string, assume there is an error in the SMBIOS text string. To view the chassis information, type the following at the prompt:

```
A:VSR#   show chassis ↵
```

The following is a sample output of this command:

```
A:Dut-A# show chassis
===============================================================================
System Information
===============================================================================
  Name                            : Dut-A
  Type                            : VSR-I
  Chassis Topology                : Standalone
  Location                        : (Not Specified)
  Coordinates                     : (Not Specified)
```

```
    CLLI code                       :
    Number of slots                 : 2
    Oper number of slots            : 2
    Num of faceplate ports/connectors : 32
    Num of physical ports           : 32
    Base MAC address                : fa:ac:ff:ff:10:00
    FP Generations                  : VFP
    System Profile                  : none
===============================================================================
Chassis Summary
===============================================================================
Chassis    Role                Status
-------------------------------------------------------------------------------
1          Standalone          up
===============================================================================
A:Dut-A#
```

5.2.3.5.3   Check the Card Types Equipped in the System

Verify that correct (virtualized) card types are equipped in the system.

If a card type does not match the one encoded in the SMBIOS text string of the corresponding VM, assume there is an error in that SMBIOS text string. To view the card information, type the following at the prompt:

```
A:VSR#   show card ↵
```

The following is a sample output of this command:

```
A:Dut-A# show card
===============================================================================
Card Summary
===============================================================================
Slot      Provisioned Type                     Admin Operational   Comments
          Equipped Type (if different)         State State
-------------------------------------------------------------------------------
1         iom-v                                up    up
A         cpm-v                                up    up/active
===============================================================================
A:Dut-A#
```

5.2.3.5.4   Check the VSR System Licenses

Verify that the VSR system has valid licenses. To view the system license information, type the following at the prompt:

```
A:VSR# show system license ↵
```

The following is a sample of the command output:

```
A:Dut-A# show system license
===============================================================================
Current License
===============================================================================
License status : monitoring, valid license record
Time remaining : 90 days 6 hours
-------------------------------------------------------------------------------
License name   : sr-regress@list.nokia.com
License uuid   : 00000000-0000-0000-0000-000000000000
Machine uuid   : 3ffaf6a4-edce-45ab-bde6-c7d1587103f9
License desc   : Virtual SR [Integrated] [ALL]
License prod   : Virtual-SR
License sros   : TiMOS-B-20.#.*
Current date   : THU MAY 02 18:51:51 UTC 2019
Issue  date    : SAT MAR 02 02:08:03 UTC 2019
Start  date    : FRI MAR 01 00:00:00 UTC 2019
End    date    : THU AUG 01 00:00:00 UTC 2019
-------------------------------------------------------------------------------
vChassis       : VSR-I
vSR CPMs       : limit: 1
vSR IOMs       : limit: 1
-------------------------------------------------------------------------------
AA_RTU         : AA Identification
                 AA Control
                 AA Policing
                 AA Stateful Firewall
                 AA RTP Performance
                 AA ICAP Control
                 AA In-Browser Notification
                 AA Local List URL Filtering
                 AA Dynamic Experience Management
                 AA TCP Optimization
                 AA Multi-Path TCP Proxy
                 AA Flow Attributes
IPSEC_RTU      : IPSec Geo Redundancy
NAT_RTU        : NAT Geo Redundancy
                 UPnP
                 L2-Aware NAT
                 LSN
                 MAP
VSR_RTU        : Legal Intercept
                 Advanced QoS BW
                 VPN
                 BNG
                 LNS
                 WLGW
                 IPSec
                 vRGW
                 Adv DCGW and Svc Chain
                 Hybrid OpenFlow Switch
                 IP Tunnels
                 NGE
WLGW_RTU       : Multiple SSID
                 Inter-AP Mobility
                 Selective Offload
                 WLGW Geo Redundancy
                 3G/4G Interworking
===============================================================================
A:Dut-A#
```

5.3 Deploying and Managing VSR VMs using OpenStack

5.3.1 OpenStack Overview

This section provides high-level guidance for the deployment of VSR VMs using OpenStack.

➡ **Note:** This chapter builds on information presented in previous sections of this installation guide. Review the concepts and procedures described in the preceding chapters before performing OpenStack deployment.

Before performing the procedures in this chapter, install the recommended server hardware and server OS, as described in NFV Infrastructure Requirements.

OpenStack is an open-source cloud management software that performs the role of a Virtualized Infrastructure Manager (VIM) in the ETSI NFV reference architecture. The VIM is responsible for controlling and managing the NFVI compute, storage, and network resources within a data center.

The OpenStack software is written in Python and is available freely under an Apache 2.0 license. There are multiple OpenStack distributions, some of which are provided with technical support and other services. The following OpenStack versions are supported for VSR deployment:

- RDO OpenStack "Liberty"
- RDO OpenStack "Mitaka"
- RDO OpenStack "Newton"
- RDO OpenStack "Ocata"
- RDO OpenStack "Pike"
- Red Hat OpenStack Platform 8 (OSP8)
- Red Hat OpenStack Platform 9 (OSP9)
- Red Hat OpenStack Platform 10 (OSP10)
- Red Hat OpenStack Platform 11 (OSP 11)
- Red Hat OpenStack Platform 12 (OSP 12)
- Mirantis OpenStack 9.0

The modular architecture of OpenStack allows the installation of different components as needed. The components most commonly used with VSR VMs include:

- keystone—authenticates and authorizes users and administrators
- horizon—provides a GUI dashboard for managing other OpenStack components
- nova—manages compute resources, determines the placement of VMs, and supports simplified networking
- neutron—manages network connectivity between VMs; applies security rules, supports DHCP/static IP addressing, NAT (floating IP), metadata services, intersubnet routing, and so on
- cinder—manages block storage devices that VMs can use for persistent storage
- glance—manages disk images that can be used to create new VMs
- heat—supports orchestration of composite applications using templates

5.3.2   Basic OpenStack Installation

The OpenStack components that are required in your environment should be installed according to the instructions provided with the OpenStack distribution.

The remaining steps in this section assume that you have an operational OpenStack deployment with at least one controller node and at least one compute node, deployed to physically separate servers.

The controller node runs:

- an SQL server with databases necessary for each OpenStack service
- a message queue service
- the OpenStack Identity (keystone) service
- the OpenStack Image Service (glance)
- control and scheduling actions for the OpenStack Compute (nova) service
- the OpenStack networking (neutron) service and ML2 plug-in

Each compute node runs the Linux KVM hypervisor, the OpenStack Compute (nova) service, and various OpenStack networking (neutron) agents.

If you plan to use the command line for managing OpenStack infrastructure Nokia recommends that you set the OpenStack environment values OS_USERNAME, OS_PASSWORD, OS_TENANT_NAME, and OS_AUTH_URL to the values you specified during installation. These environment variables can be saved in a keystonerc_admin file and sourced when necessary.

5.3.3   Preparing the OpenStack Environment for VSR VMs

Perform the procedures described in this section to customize the OpenStack setup so that it suitable for the deployment of VSR VMs.

5.3.3.1   Optimize BIOS and Linux Kernel Settings of Compute Nodes

The recommendations presented in BIOS Settings and Linux KVM Compute Hosts should be followed to ensure that the BIOS settings and Linux kernel settings of each compute node are appropriate for the VSR VMs that will be deployed to that host.

5.3.3.2   Adjust Compute Node Resource Allocation

The CPU and RAM must not be oversubscribed on any compute node that will be eligible for hosting VSR VMs.

On each compute node, edit the `/etc/nova/nova.conf` file to include the following lines:

- cpu_allocation_ratio = 1.0
- disk_allocation_ratio = 1.0
- ram_allocation_ratio = 1.0

In addition, provide a value for **vcpu_pin_set** that reserves a list of CPUs for the guests; the list of CPUs should be isolated from the host as described in CPU Isolation. For example:

```
vcpu_pin_set=1-17,19-35
```

5.3.3.3   Adjust Nova Scheduler Parameters

To adjust the Nova Scheduler Parameters, perform the following steps.

Step 1. On the controller node, edit the `/etc/nova/nova.conf` configuration file to ensure that the Nova scheduler takes all necessary constraints into account.

Step 2. In the configuration file, the following line may be disabled (commented out). Enable it as follows:

```
scheduler_available_filters=nova.scheduler.filters.all
_filters
```

Step 3. Add another line for the available filters, as follows:

```
scheduler_available_filters=nova.scheduler.filters.pci
_passthrough_filter.PciPassthroughFilter
```

Step 4. To ensure that VSR VMs are deployed to hosts, update the line for `scheduler_default_filters` with additional keywords so that it reads as follows (note the keyword in bold):

```
scheduler_default_filters=RetryFilter,AvailabilityZone
Filter,RamFilter,ComputeFilter,ComputeCapabilitiesFilt
er,ImagePropertiesFilter,CoreFilter,PciPassthroughFilt
er,NUMATopologyFilter,AggregateInstanceExtraSpecsFilte
r
```

5.3.3.4   (Optional) Enable SR-IOV on OpenStack Controller and Compute Nodes

To use SR-IOV with VSR VMs, enable SR-IOV on the OpenStack controller and also on each compute node with VSR VMs that will have SR-IOV interfaces. Follow these steps:

5.3.3.4.1   Enable SR-IOV on OpenStack Controller and Compute Node

Follow the steps to enable SR-IOV on the OpenStack controller and the compute node.

Step 1. Edit the ML2 plugin file on the OpenStack controller node by editing the following file:

```
/etc/neutron/plugins/ml2/ml2_conf.ini
```

Step 2. Change the `mechanism_drivers` line to
`mechanism_drivers=openvswitch,sriovnicswitch`

Step 3. Check the SR-IOV NIC type from the compute node.

Use the following command on the compute node:

`#lspci -nn | grep -i "Virtual Function"`

Sample output:

```
09:10.0 Ethernet controller [0200]: Intel Corporation 82599 Ethernet Controller Virt
ual Function [8086:10ed] (rev 01)
```

In this output, the SR-IOV NIC type is `8086:10ed`.

Step 4. On the OpenStack controller node, add the SR-IOV NIC type in the `/etc/neutron/plugins/ml2/ml2_conf_sriov.ini` file by adding the following parameter:

`supported_pci_vendor_devs = 8086:10ed`

Step 5. Update file `/usr/lib/systemd/system/neutron-server.service` to include the SR-IOV service by adding the following to `ExecStart`:

`--config-file /etc/neutron/plugins/ml2/`
`ml2_conf_sriov.ini`

Step 6. Update the `/etc/nova/nova.conf` file on each compute node with the PCI whitelist.

For example, the output of `virsh nodedev-dumpxml pci_0000_$id` for each SR-IOV PF shows a list of the SR-IOV virtual functions (VF). This includes a list of the VFs in the PCI whitelist.

```
pci_passthrough_whitelist={"address":"*:09:10.0","physical_network":"sriovNet1"}
pci_passthrough_whitelist={"address":"*:09:10.1","physical_network":"sriovNet1"}
pci_passthrough_whitelist={"address":"*:09:10.2","physical_network":"sriovNet1"}
pci_passthrough_whitelist={"address":"*:09:10.3","physical_network":"sriovNet1"}
pci_passthrough_whitelist={"address":"*:09:11.0","physical_network":"sriovNet2"}
pci_passthrough_whitelist={"address":"*:09:11.1","physical_network":"sriovNet2"}
pci_passthrough_whitelist={"address":"*:09:11.2","physical_network":"sriovNet2"}
pci_passthrough_whitelist={"address":"*:09:11.3","physical_network":"sriovNet2"}
```

Step 7. For Liberty, update the MTU value on each compute node in file `/etc/nova/nova.conf`, by updating the following parameter:

`network_device_mtu=9000`

For Mitaka, update the MTU value on each compute node in file `/etc/neutron/neutron.conf`, by updating the following parameter:

`global_physnet_mtu = 9000`

Step 8. Reboot the compute node and check the status of the OpenStack services.

5.3.3.5   (Optional) Create Volume Drives using OpenStack Cinder

Up to three Cinder volumes may be attached (as block devices) to a VSR-I. The "hda" device corresponds to CF3, the "hdb" device corresponds to CF1 and the "hdc" device corresponds to CF2. If a Cinder volume is created from the QCOW2 disk image provided by Nokia, the size will be limited to 1189 MB even if the requested volume size is greater.

Use this procedure to create a volume using OpenStack. The newly created volume drive is attached to the VSR-I during VSR instantiation.

5.3.3.5.1   Create a Volume Drive using OpenStack Cinder

Use this procedure to create a volume using OpenStack. The newly created volume drive is attached to the VSR-I during instantiation.

FTP or TFTP server is required on the management network to download the configuration and license files used by the VSR when it is booting up.

Step 1. Create a volume drive using the OpenStack Cinder module:

```
cinder create --display_name $NAME $SIZE ↵
```

Where: SIZE is the volume drive size in GB

Step 2. Create a volume drive diskStorage1 to connect to the VSR.

Base the volume disk dimensions on the amount of storage required.

```
cinder create --display_name diskStorage1 32 ↵
```

Step 3. Verify the volume drive creation.

```
cinder list ↵
```

Output of the above command should list diskStorage1 as an available volume drive.

5.3.3.6   Create Nova Flavors Appropriate for VSR VMs

VM flavor is a template that defines the compute resources required by a VM, including the number of CPU cores to allocate to the VM and the amount of RAM to allocate to the VM. A Nova flavor can also specify extra-specs.

The flavor used to instantiate a VSR VM must specify:

• an appropriate number of vCPUs; this must be an even number if the VSR VM is deployed on a hyper-threaded host

- an appropriate amount of vRAM memory; this must be a multiple of 1GB and must adhere to recommendations given in the *SR OS 20.x.Rx.* Software Release Notes

It must also specify (as extra specifications):

- `hw:mem_page_size=1GB`; this ensures that the guest memory is backed by 1GB huge pages
- `hw:numa_mempolicy=strict`; this ensures that CPU and memory resources are allocated from the same NUMA node
- `aggregate_instance_extra_specs:pinned=true`; this ensures that VSR VMs are assigned to hosts that support CPU pinning
- `hw:cpu_policy=dedicated`; this ensures that one host (physical CPU) core/ thread is dedicated to each guest vCPU
- `hw:cpu_threads_policy=isolate` (if the host machine is not hyper-threaded) or `hw:cpu_threads_policy=prefer` (if the host machine is hyper-threaded)
- `hw:cpu_sockets`=1
- `hw:cpu_cores`=N

  where N = the number of vCPUs in a non-hyper-threaded host or half the number of vCPUs in a hyper-threaded host

To verify the created VM flavors, use the following command:

```
# nova flavor-list
```

5.3.3.7   Add VSR Images to OpenStack

This section describes how to upload a VSR QCOW2 file to OpenStack. Disk images are managed by the OpenStack Glance service. New VSR VMs are created in OpenStack using one of the QCOW2 image files in Glance. (You may have multiple VSR image files in Glance; for example, one for each deployed version of VSR software).

Step 1.   Use the following command to upload an image file:

```
# glance image-create --name $imageName --disk-format
qcow2 --container-format bare $image.qcow2 ↵
```

Step 2.   Verify the uploaded VSR image file as follows:

```
t# glance image-list ↵
```

5.3.3.8   Create Neutron Networks, Subnets, and Ports

Neutron networks and ports are needed to bind the vNIC interfaces of a VM to physical NIC ports so that external communication is possible.

Step 1.   Use the following command to create a network:

```
# neutron net-create --tenant-id $tenantId $networkName
--provider:network_type $networkType --
provider:physical_network $physicalNetwork --
provider:segmentation_id $vlanId ↵
```

where:

- supported values for networkType are vlan and flat
- provider:segmentation_id is not applicable when the networkType is flat.
- physicalNetwork accepts values defined while installing the OpenStack node. In this section, we will assume three physical networks: physnet1, physnet2, and physnet3

Step 2.   As necessary, create the networks listed in Table 8. You may choose to adapt the sample commands.

Table 8   **Network Configuration Sample Commands**

Network Type	Sample Command
OOB management network (typically flat) shared by one of more VSR VMs	`# neutron net-create --tenant-id $tenantId mgt_network -- provider:network_type flat -- provider:physical_network physnet1 ↵`
External network; for example, tied to port 1/1/1	`# neutron net-create --tenant-id $tenantId Ext_Data_Network -- provider:network_type vlan -- provider:physical_network physnet3 -- provider:segmentation_id 4 ↵`

Step 3.   After creating all required Neutron networks, associate a subnet with each one. Use the following command to create a subnet:

```
# neutron subnet-create --no-gateway --name $subnetName
--allocation-pool start=$startIp,end=$endIp --disable-
dhcp $networkName $CIDR ↵
```

Table 9 lists subnet configurations examples for the networks.

Table 9   **Subnet Configuration Sample Commands**

Network Type	Sample Command
OOB management network (typically flat) shared by one of more VSR VMs	`# neutron subnet-create --no-gateway --name mgt_subnet --allocation-pool start=135.227.248.2,end=135.227.248.22 --disable-dhcp mgt_network 135.227.248.0/21 ↵`
External network; for example, tied to port 1/1/1	`# neutron subnet-create --no-gateway -name Ext_Data_Subnet --allocation-pool start=10.2.2.1,end=10.2.2.1 --disable-dhcp Ext_Data_Network 10.2.2.0/30 ↵`

Step 4. After creating the subnets, verify them as follows:

`# neutron subnet-list ↵`

The command output should list subnets `mgt_subnet`, `Int_Ctrl_Subnet`, and `Ext_Data_Subnet` if you followed the above examples.

Step 5. Create Neutron ports and assign an IP address to the ports using the following command. The port will be attached as a vNIC interface when the VM is created.

`# neutron port-create $networkName --name $name --fixed_ip ip_address=$ipAddress -binding:vnic-type $type↵`

Table 10 describes sample commands for configuring neutron ports that connect to the networks.

Table 10   **Neutron Port Configuration Sample Commands**

Network Type	Sample Command
OOB management network (typically flat) shared by one of more VSR VMs	`# neutron port-create mgt_network --name OOBmgmtPort --fixed_ip ip_address=135.227.248.2` Ensure that IP addresses assigned to the ports for external connectivity are consistent with the VSR CLI configuration. For example, verify that the management port IP address is the same as the IP address configured in the bof.cfg.

Table 10   **Neutron Port Configuration Sample Commands (Continued)**

Network Type	Sample Command

External network; for example, tied to port 1/1/1	`host# neutron port-create Ext_Data_Network --name Port111 -- fixed_ip ip_address=$ipAddress -- binding:vnic-type direct`
	
	The `binding:vnic-type direct` configures the Neutron as an SR-IOV port.
	
	For non SRIOV ports it may be necessary to change the anti-spoofing rules applied by Neutron. For example, to allow a second CP-VM to take over the management IP address of the first CP-VM you would modify the Neutron port configuration as follows:
	`# neutron port-update $standby-cp-vm-port-id --allowed_address_pairs list=true type=dict ip_address=$A ↵`
	where A is the management IP of the active CP-VM.
	
	To completely bypass anti-spoofing processing, disable the IP tables service on the compute host, as follows:
	`# service iptables stop`

Step 6. After creating all the Neutron ports, verify them as follows:

```
# neutron port-list ↵
```

5.3.3.9   Create Security Groups

When creating a VM (Nova server) in OpenStack, a security-group must be specified to apply to that VM. Security-groups control the traffic flows that can ingress to the VM and egress from the VM over non-SRIOV interfaces. The default security group provided by OpenStack is generally too restrictive for most VSR VMs and therefore it is recommended to create new security group profiles.

The general workflow for security group creation is as follows:

Step 1. Create a security group for the VSR instance.

To create a security group called SrosSecurityGroup, use the following OpenStack command:

```
# neutron security-group-create SrosSecurityGroup ↵
```

Step 2. Verify the configured security group.

```
# neutron security-group-list ↵
```

Step 3. Assign access rules to the security group.

```
# neutron security-group-rule-create --direction
ingress(or egress) --ethertype IPv4 (or IPv6) –protocol
$protocolNum --remote-ip-prefix $remoteIpCidr --port-
rangemin $startPortRange --port-range-max $endPortRange
SrosSecurityGroup ↵
```

Step 4. Create TCP, ICMP, and UDP ingress and egress rules.

Step 5. Verify the configured security group rules as follows; the command output
should list all the security rules for the configured SrosSecurityGroup:

```
# neutron security-group-rule-list ↵
```

5.3.4   Deploying a VSR Instance Using OpenStack CLI

Perform Create VMs to deploy a VSR instance using OpenStack CLI commands.

5.3.4.1   Create VMs

To create a VSR VM and start it up immediately, perform the following steps.

Step 1. Use the following CLI command:

```
# nova boot --image $imageName --flavor $flavorType --
nic port-id=$port-id --config-drive $cfgfile --
securitygroups $securityGroup block-device-mapping
vdb=$diskstorage1id --availability-zone
nova:$availabilityZone $VMName ↵
```

➡

  Note: For more information about `config-drive`, see ConfigDrive.

For example, to create a VSR-I VM with one vNIC bound to the
management port A/1 and one vNIC bound to port 1/1/1, you could use a
command such as the following:

```
# nova boot --image srosVmImage --flavor vsrIntegrated
--nic port-id=$OAMportID -nic port-id-$port111id --
config-drive $cfgfile --securitygroups
SrosSecurityGroup block-device-mapping
vdb=$diskstorage1id --availability-zone nova:$zone1
vsrA ↵
```

This assumes that `cfgfile` is a text file with a properly formatted SMBIOS product string starting with the characters "TIMOS:".

Step 2. Verify VM creation using the following OpenStack command:

```
# nova list ↵
```

5.3.5   Deploying a VSR Instance Using OpenStack HEAT

Perform Create the HEAT Stack to deploy a VSR instance using OpenStack HEAT.

5.3.5.1   Introduction to OpenStack HEAT

VSR instances can also be deployed using OpenStack HEAT. HEAT provides template-based orchestration within OpenStack. A HEAT Orchestration Template (HOT) defines a HEAT stack. In an NFV context, the HEAT stack describes the resources needed to create a VNF instance including the type and number of VMs, the VM image, storage, networks, routers, security groups and so on. The template is defined in YAML format and it is reusable; that is, a template can be invoked multiple times to create several instances of a VNF.

Table 11 describes the main HEAT terms.

Table 11   **Main HEAT Terms**

Term	Description
HEAT stack	The stack objects are a collection of resources that HEAT creates. This could include instances (VMs), networks, subnets, routers, ports, security groups, and so on.
Template	It is used to define a stack.
Parameters	HOT files have three major sections and one section defines a template's parameters. Information includes, specific image IDs, network IDs, and subnet IDs that are passed on to the HOT file by the user.

Table 11　　**Main HEAT Terms (Continued)**

Term	Description
Resources	Specific objects that HEAT will create and/or modify. This is one of the three major sections in a HOT file.
Output	The major section of a HOT file which is information that is passed on to the user either via OpenStack Dashboard or via the **heat stack-list** or **heat stack-show** commands.

5.3.5.2　Overview of a VSR HEAT Template

The following configuration shows an OpenStack Heat template that could be used to create VSR-I VMs. The sample is commented to provide an explanation of the template:

```
heat_template_version: 2016-04-08
# must be either 2015-10-15 (compatible with Liberty) or 2016-04-
08 (compatible with Mitaka)
description: Simple template to create VSR-I
parameters:
  image:
    type: string
    description: VSR image name
    default: "VSR-20.2.R1"
    constraints:
      - custom_constraint: glance.image
        description: Must identify an image known to Glance -
 Use "openstack image list" to see available images
  flavor:
    type: string
    description: VSR flavor name
    default: "vsr-i-extra-specs"
    constraints:
      - custom_constraint: nova.flavor
        description: Must identify a known flavor -
 Use "openstack flavor list" to see available flavors
  mgt_network:
    type: string
    description: Management network attached to A/1
    constraints:
      - custom_constraint: neutron.network
        description: Must identify an existing network
  network1:
    type: string
    description: External network attached to port 1/1/1
    constraints:
      - custom_constraint: neutron.network
        description: Must identify an existing network
  network2:
    type: string
    description: External network attached to port 1/1/2
```

```
        constraints:
          - custom_constraint: neutron.network
            description: Must identify an existing network
    license_file:
      type: string
      description: License file
      default: "cf3:/license.txt"
    primary_config:
      type: string
      description: Primary configuration file
      default: "cf3:/config.cfg"
    static_routes:
      type: string
      description: Static routes for out of band - example static-route="100.0.0.0/
8 next-hop 10.10.10.10"
      default: ""
    mda_1:
      type: string
      description: mda type for slot 1/1
      default: m20-v
      constraints:
        - allowed_values:
          - m20-v
          - isa-aa-v
          - isa-bb-v
          - isa-tunnel-v
    cpmSmbios:
      type: string
      description: SMBIOS string
      default: "TiMOS: slot=$slot chassis=VSR-I card=cpm-v mda/1=$mda_1 license-
file=$license_file primary-config=$primary_config address=$adrp/
24@active $static_routes "
resources:
# Create Neutron ports
  mgmt_port:
    type: OS::Neutron::Port
    properties:
        replacement_policy: AUTO
        binding:vnic_type: normal
        network_id: { get_param: mgt_network }
  port_1:
    type: OS::Neutron::Port
    properties:
        replacement_policy: AUTO
        binding:vnic_type: direct
        network_id: { get_param: network1 }
  port_2:
    type: OS::Neutron::Port
    properties:
        replacement_policy: AUTO
        binding:vnic_type: direct
        network_id: { get_param: network2 }
# Create VSR-I
  VSR_I:
    type: OS::Nova::Server
    depends_on: [ mgmt_port, port_1, port_2 ]
    properties:
      name:
        list_join: ['-', [ {get_param: 'OS::stack_name'}, 'cp-a']]
```

```
                    image: { get_param: image }
                    flavor: { get_param: flavor }
                    networks:
                    - port: { get_resource: mgmt_port }
                    - port: { get_resource: port_1 }
                    - port: { get_resource: port_2 }
                    config_drive: "true"
                    user_data_format: RAW
                    user_data:
                      str_replace:
                        template: { get-param: cpmSmbios }
                        params:
                          $slot: "A"
                          $adrp: { get_attr: [ mgmt_port, fixed_ips, !!int 0, ip_address] }
                          $static_routes: { get-param: static_routes }
                          $license_file: { get-param: license_file }
                          $primary_config: { get-param: primary_config }
                          $mda_1: { get-param: mda_1 }
outputs
  vsr_ip:
    description: fixed ip assigned to the server
    value: { get_attr: [ mgmt_port, fixed_ips, 0, ip_address ] }
```

5.3.5.3 Create the HEAT Stack

To instantiate a VSR-I instance using the HEAT template example in Overview of a VSR HEAT Template perform the following steps.

Step 1. Use a command such as:

```
#heat stack-create -f VSR-I.yaml -P mgt_network="VSR-
mgt" -P network1="network1" -P network2="network2" vsr1
```

This would create a VSR-I VM with the following properties:

- it would be created from:
  - a Glance disk image called "VSR-20.*x*.R1"
  - a Nova flavor called "vsr-i-extra-specs". This Nova flavor would be expected to conform with the recommendations given in Create Nova Flavors Appropriate for VSR VMs.
- it would use:
  - a license file called "license.txt" on the CF3 disk
  - the default config file on CF3
- it would have:
  - one m20-v MDA
  - three ports: a management port, an MDA port 1/1/1 (bound, using SR-IOV, to network1), and an MDA port 1/1/2 (bound, using SR-IOV, to network2)

- initially, no static routes for management connectivity
- a management IP address allocated from the subnet associated with the "VSR-mgt" network

Step 2. Use the **# heat stack-list** command to verify the successful deployment of one or more VSR instances from HEAT templates.

6 Deploying VSR-I on VMware ESXi Hosts

6.1 VMware Overview

This chapter describes how to create and start up VSR-I VMs on host machines using the VMware ESXi hypervisor.

There are two methods for deploying VSR-I VMs on VMware ESXi hosts: automated onboarding of a vAPP using VMware vCloud Director (vCD) or OVA import using the vSphere Web Client interacting with a vCenter Server. The automated onboarding using vCD is recommended for deployments using VMXNET3 connectivity, while the vSphere Web client method is required for deployments using SR-IOV or PCI passthrough connectivity.

Note: Other techniques for deploying VSR-I VMs on ESXi hosts (for example, direct ESXi shell access) are not covered in this guide.

Regardless of the deployment method, the following VMware features are supported with VSR-I VMs:

- Distributed Resource Scheduler (DRS)—except fully-automated mode
- High Availability
- vSphere standard switch—connected to the guest using an E1000 or VMXNET3 driver
- vSphere distributed switch (vDS)—connected to the guest using an E1000 or VMXNET3 driver
- SR-IOV and PCI passthrough (NIC model dependent)

The following vSphere features are unsupported:

- DRS fully-automated mode
- vMotion
- Storage vMotion
- Fault Tolerance
- NSX-enabled distributed switch

6.2   VMware ESXi Host Setup

Regardless of application, a VSR-I VM can be deployed on any compute host running the ESXi 6.0, ESXi 6.5, or ESXi 6.7 hypervisor. In addition, when configured only as a control plane BGP route reflector, a VSR-I VM may also be deployed on any compute host running the ESXi 5.5 hypervisor. No other ESXi versions are supported.

6.2.1   Optimize BIOS and Host Settings

Follow the recommendations presented in BIOS Settings to ensure that the BIOS settings of each ESXi compute node are appropriate for the VSR-I VMs that will be deployed to that host.

If the VMXNET3 driver in the VSR software is used to connect VSR-I vNIC interfaces to a vSphere standard switch (VSS) or a vSphere distributed switch (VDS), the following host-level changes may improve performance.

- Increase the number Tx buffers associated with each physical NIC port to 4096. Depending on the NIC, use one of the following commands (for the physical NIC port corresponding to vmnic0):

  ```
  # ethtool -G vmnic0 tx 4096
  # esxcli network nic  ring current set -n vmnic0 -t 4096
  ```

- Increase the number of Rx buffers associated with each physical NIC port to 4096. Depending on the NIC, use one of the following commands (for the physical NIC port corresponding to vmnic0):

  ```
  # ethtool -G vmnic0 rx 4096
  # esxcli network nic  ring current set -n vmnic0 -r 4096
  ```

- Enable the NetQueue feature using the following command:

  ```
  # esxcli system settings kernel set --setting
  "netNetqueueEnabled" --value "true"
  ```

6.3 Deploying the VSR-I vApp using vCloud Director

A VSR-I VM can be instantiated in a VMware data center by onboarding the VSR-I OVF package (.ova) into vCloud Director, using the web interface. The OVF package is an archive containing, at a minimum, an XML-based OVF descriptor file (.ovf), a manifest file, and a VMDK disk image containing the SR OS software images. One section of the OVF descriptor file is a VirtualSystem object. The VirtualSystem object describes the abstract resource requirements of a VSR-I VM and contains the following subsections:

- OperatingSystemSection—indicates that the VSR-I guest OS type is "otherGuest64" (a non-Linux, non-Windows 64-bit guest O/S)
- VirtualHardwareSection—specifies resource requirements of the VSR-I VM, including vCPUs, memory, disks, and virtual NICs. It also contains ExtraConfig information with settings for additional parameters that improve the performance and stability of VSR-I instances. Further details about the VirtualHardwareSection are presented in Table 12, which describes some of the most important parameters in this section, the default value of each one in the OVF, and whether the default value can be changed at time of onboarding.

Table 12   **VirtualHardwareSection Parameters**

Parameter	Default Value in OVF	Notes
Number of vCPUs	6	Can be changed at time of onboarding
Amount of memory	8GB	Can be changed at time of onboarding
vNIC interface #1 (ethernet0)	E1000 adapter	Network binding can be changed at time of onboarding
vNIC interface #2 (ethenet1)	VMXNET3 adapter	Network binding can be changed at time of onboarding
ExtraConfig: sched.cpu.latencySensitivity	"high"	Setting latencySensitivity to "high" is mandatory when VSR-I VMs are deployed without CPU pinning directives in the VMX file. This ensures that each vCPU of the VM is pinned to a physical CPU and that no other workload is assigned to that physical CPU even if hyper-threading is not disabled on the host machine. This configuration also disables interrupt coalescing automatically.

Table 12   **VirtualHardwareSection Parameters (Continued)**

Parameter	Default Value in OVF	Notes
ExtraConfig: sched.cpu.latencySensitivity.sysContexts	"1"	This setting allocates one dedicated physical CPU for the Rx and Tx threads of all vNIC ports of the VM, provided that there is at least one non-reserved physical CPU core available after vCPU reservations. If VMXNET3 interfaces are used into a VSS or VDS switch, it is recommended to increase this value to a value of 6, but a minimum of 4 at time of onboarding.
ExtraConfig: ethernet1.ctxPerDev	"1"	Setting ethernet1.ctxPerDev to "1" is required when ethernet1 is a high traffic rate vNIC port using the VMXNET3 adapter, as will always be the case when deploying a VSR-I from the supplied vApp template. This setting allocates a dedicated Tx thread to the ethernet1 vNIC interface. By default, there is only one software Tx thread for the entire VM, shared by all its vNIC ports.
ExtraConfig: numa.nodeAffinity	"1"	Setting numa.nodeAffinity to "1" ensures that all vCPUs and all memory are allocated to the VM from NUMA node "1" on any server with two CPU sockets and two associated NUMA nodes. This can be changed to a value of "0" at time of onboarding. It is only important that all resources are NUMA aligned.

Note: When a VSR-I VM is deployed on a server, ensure that there is a sufficient number of physical CPU cores unreserved by VMs to allow for general hypervisor functions and the number of sched.cpu.latencySensitivity.sysContexts.

6.3.1 vCD Requirements for OVA Onboarding

A user must have additional permissions to onboard the VSR-I vAPP in vCloud Director. By default, the pre-created organizational roles do not consider ExtraConfig parameters when onboarding OVA files. The system administrator must add a new role in the organization that allows vCD to consider ExtraConfig parameters during the OVA onboarding. To create this new role using the vCD web console, follow the instructions in the VMware vCD documentation. When assigning rights to the new role, the following rights must be included:

- vApp: Preserve All ExtraConfig Elements During OVF Import and Export
- vApp: Preserve Latency ExtraConfig Elements During OVF Import and Export
- vApp: Preserve NUMA Node Affinity ExtraConfig Elements During OVF Import and Export

6.3.1.1 Create Networks

To onboard the VSR-I vApp the following networks must be pre-created in vCD prior to OVA deployment:

- a management network for OOB management of the VSR-I and potentially other VNFs

  A single port group must be created for the management network.
- an external network that becomes attached to port 1/1/1 of the VSR-I

  This external network allows the VSR-I to communicate with other network elements.

6.3.2 vApp Installation Steps Through vCD

This section describes the required steps to install VSR-I vApp using vCD.

6.3.2.1 VSR-I OVA Onboarding to the vCD Catalog

Perform the following steps to upload the VSR-I OVA to the vCD catalog.

Step 1. Log in to the vCD web console as an organization user with the rights described in vCD Requirements for OVA Onboarding.

Step 2.   Navigate to the Catalogs tab.

Step 3.   Select and open the organization to be used for the VSR-I deployment.

Step 4.   Under the vApp Template tab, select the upload option, and enter the required information as shown in Figure 5.

➡ **Note:** The correct OVA file is named sros-vsr.ova. Do not use the sros-vm.ova file for onboarding a VSR-I. This OVA is intended to be used for the vSIM only.
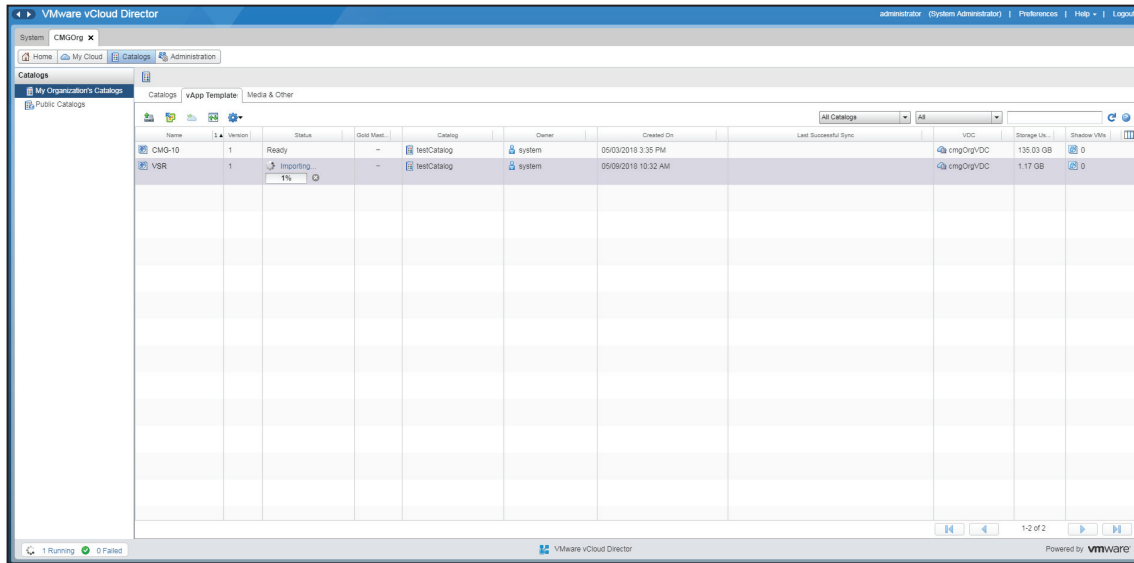
Figure 5   **vApp Template Tab**
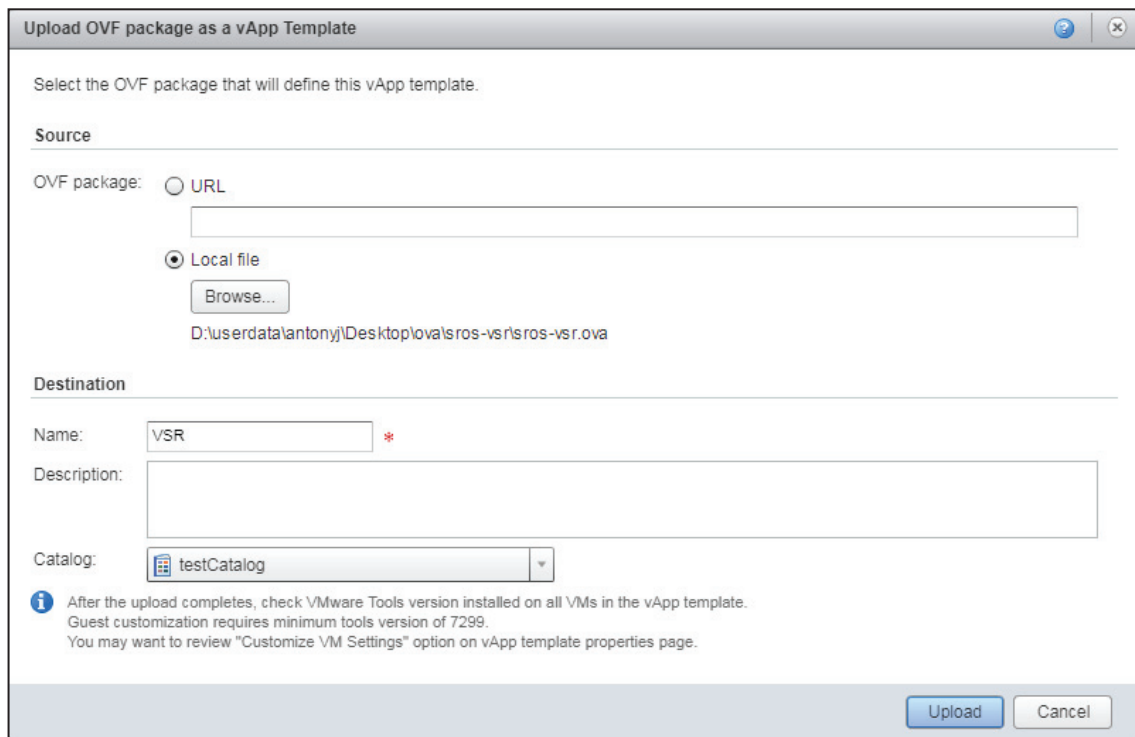
sc0022

Step 5.   Click Upload to complete the upload procedure.

Step 6. After the upload is complete, the vApp will appear in the vApp catalog under the specified name, as shown in Figure 6.
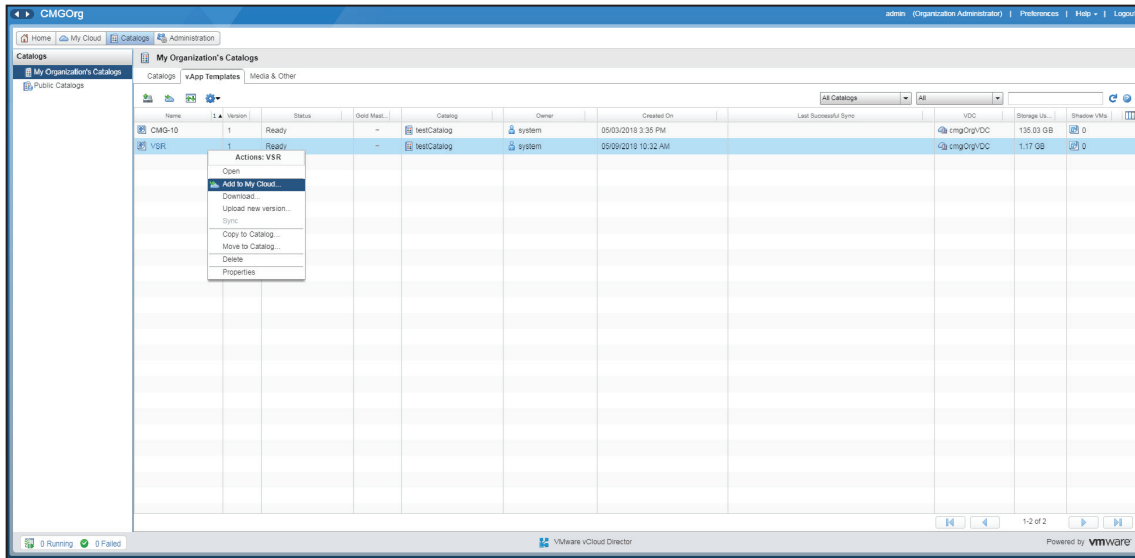
Figure 6   **vApp Template Catalog**

sc0023

6.3.2.2   Deploy the VSR-I vApp

Perform the following steps to deploy the VSR-I vApp.

Step 1. Navigate to the organization's Catalogs tab.

Step 2. Right-click the VSR vApp Template and select the option Add to My Cloud, as shown in Figure 7.

Figure 7 **Add to My Cloud**

sc0015

Step 3. Enter the name for the vApp to be deployed and a virtual data center location, as shown in Figure 8.

Figure 8 **Enter the Name for the vApp**

sc0016

Step 4. In Configure Resources, select a storage policy for the VM included in the vApp, as shown in Figure 9.

Figure 9   **Select a Storage Policy**
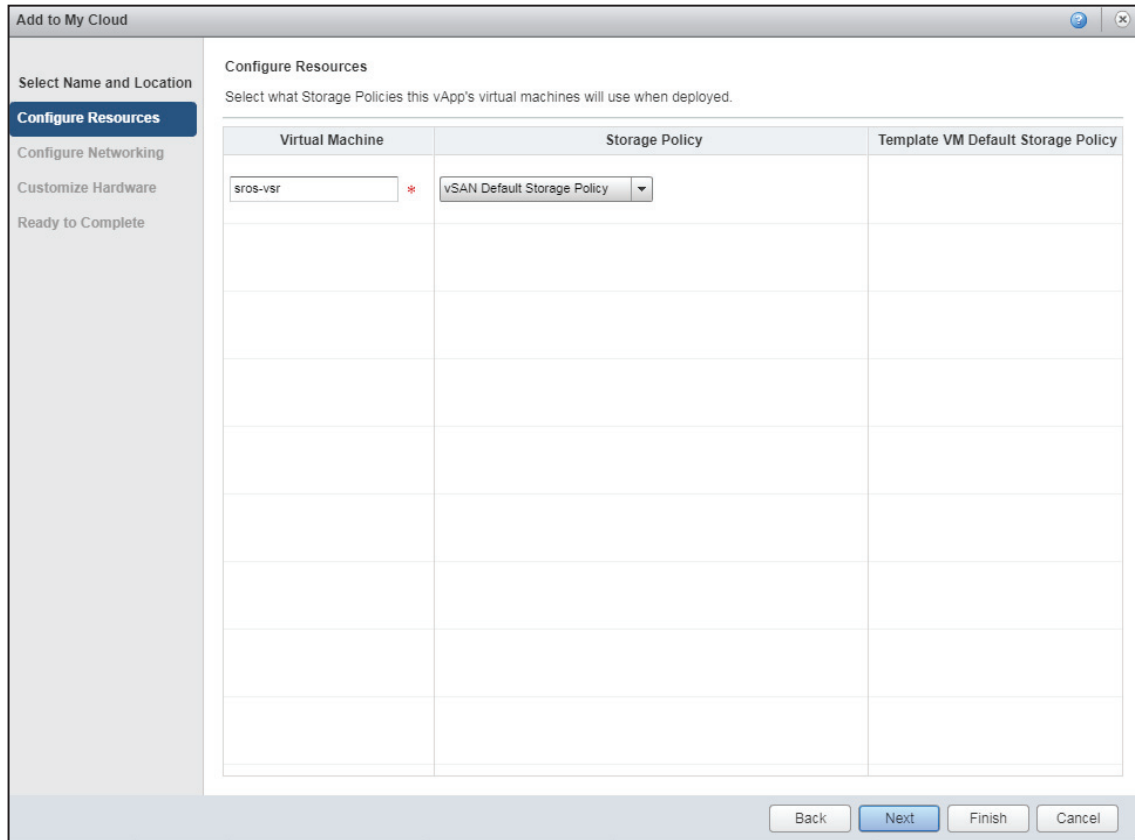
sc0017

Step 5. In Configure Networking, map the networks in the vApp to the pre-created networks (port groups) in the vDC and choose the static or DHCP IP allocation scheme, as shown in Figure 10.

Figure 10 **Map the Networks in the vApp**
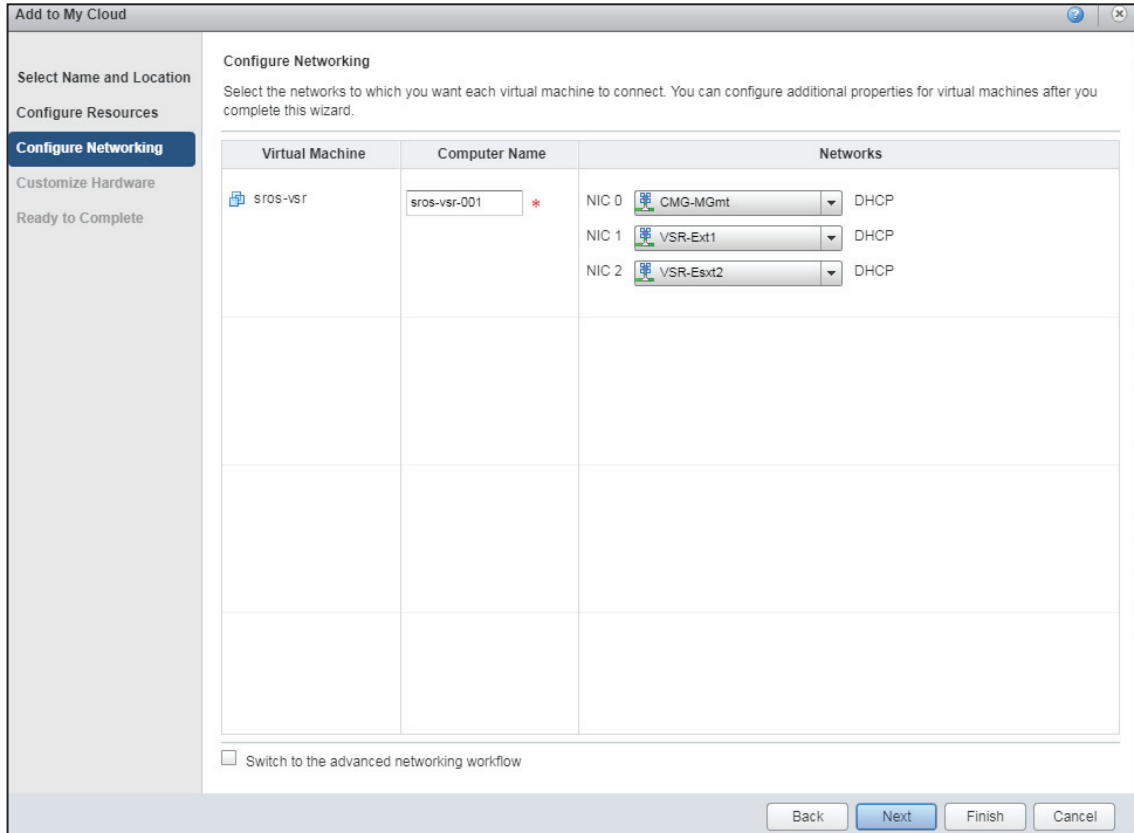
Step 6. In Customize Hardware, if necessary, modify values for the number of virtual CPUs, cores per socket, total memory, and disk size, as shown in Figure 11. Refer to the *SR OS 20.x.Rx.* Software Release Notes for more information.

Figure 11   **Modify Values in Customize Hardware**
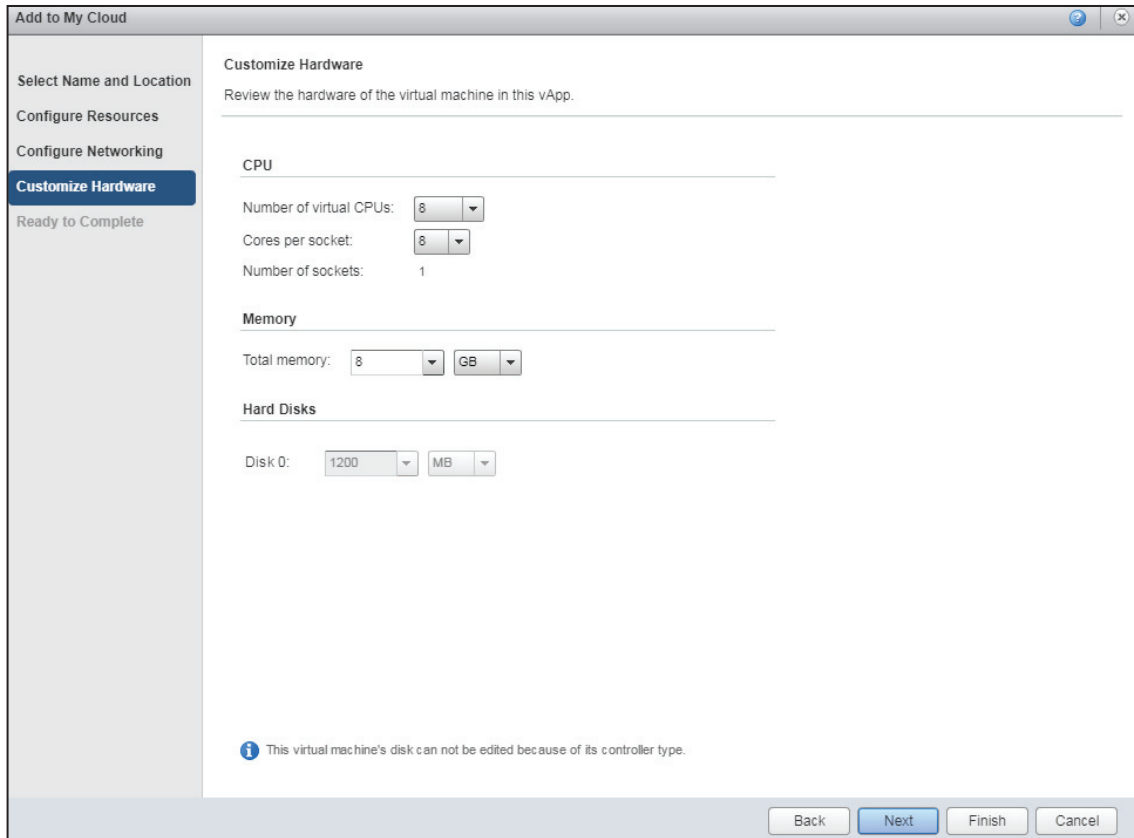
Step 7. Proceed to the final Ready to Complete review screen and deploy the vApp by clicking on Finish, as shown in Figure 12. This will create the VSR-I.

Figure 12 **Ready to Complete Screen**
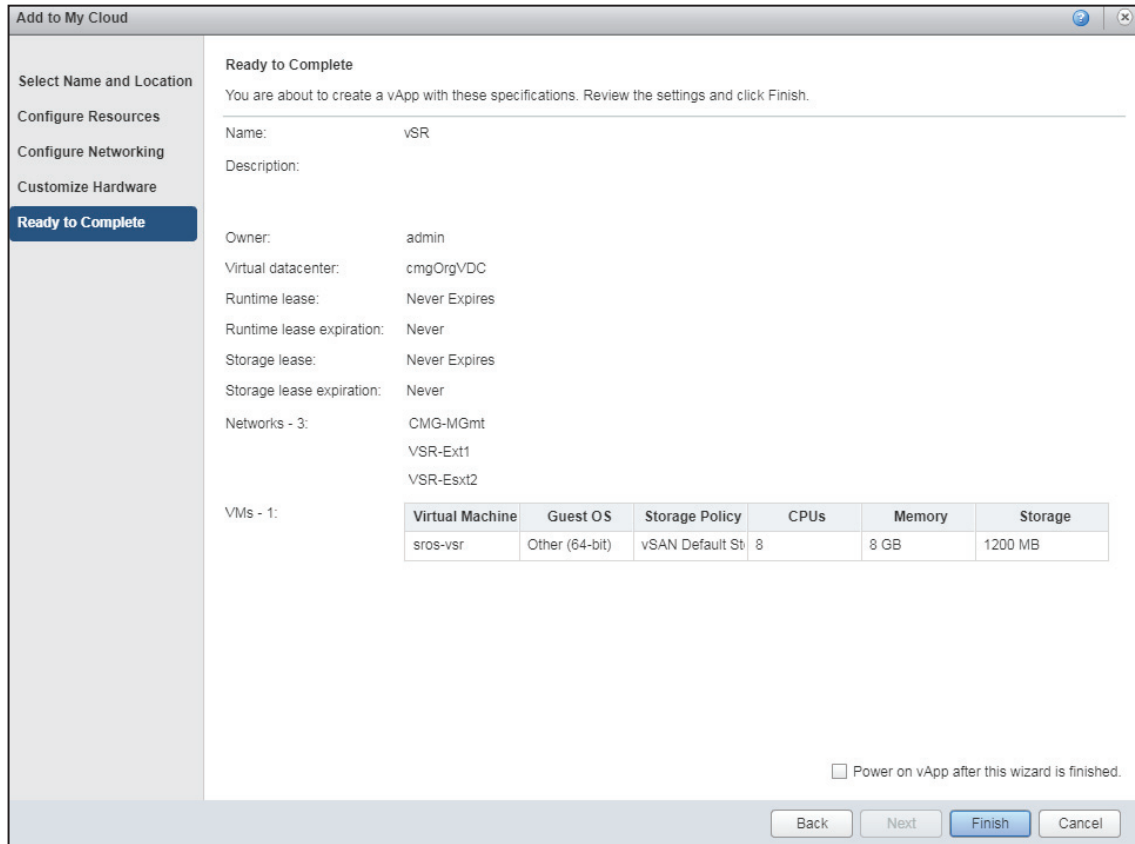
sc0020

Step 8. Start the VSR-I by right-clicking the vApp and selecting the Start action from the menu, as shown in Figure 13.

Figure 13 **VSR Actions**
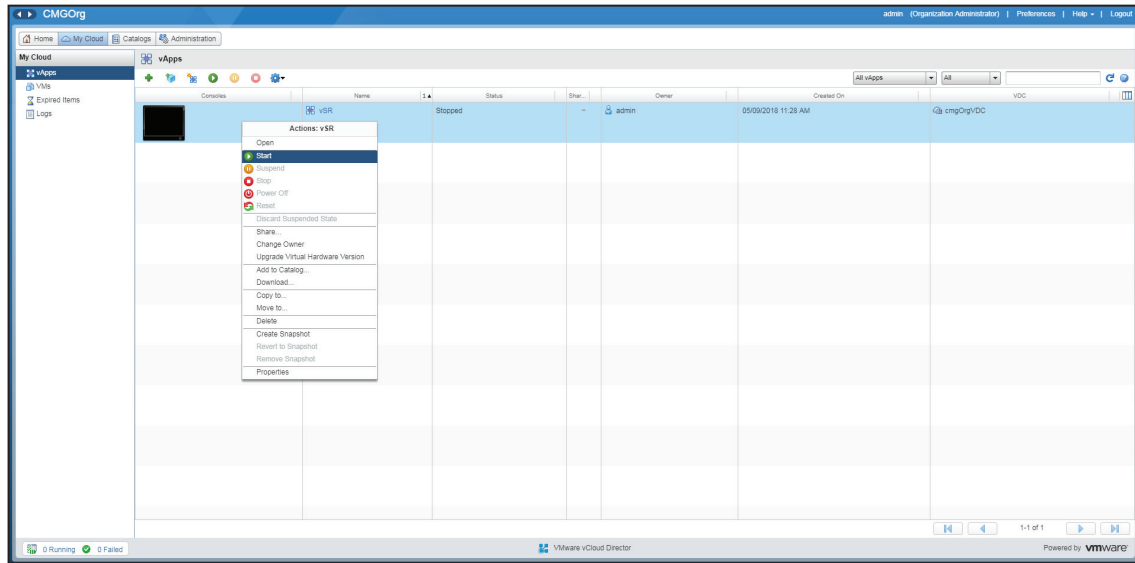
sc0021

6.4   Instantiating a VSR-I using vSphere Web Client

A VSR-I VM can also be instantiated in a VMware data center by direct import of the OVA package into vCenter Server. In this workflow, you interact with the vCenter Server using the vSphere Web Client interface. You need to use this method if you intend to use SR-IOV or PCI passthrough with the VSR-I VM. It is also necessary to use this method if you want to optimize the performance of a VSR-SeGW or VSR-AA application to use the resources of a hyper-threaded host machine as efficiently as possible.

➡ **Note:** The vSphere Web Client is preferred over the legacy vSphere Client for instantiation and management of VSR-I VMs in vCenter Server.

Before following this procedure, ensure that the vCenter Server is already installed, and at least one ESXi host is added to the data center group.

6.4.1   Connect to the vCenter Server

Perform the following steps to connect to the vCenter Server using the vSphere Web Client.

Step 1.   Connect to the vCenter Server over HTTP and log in from the VMware vCenter Single Sign-On window, as shown in Figure 14.

Figure 14   **VMware vCenter Single Sign-On**
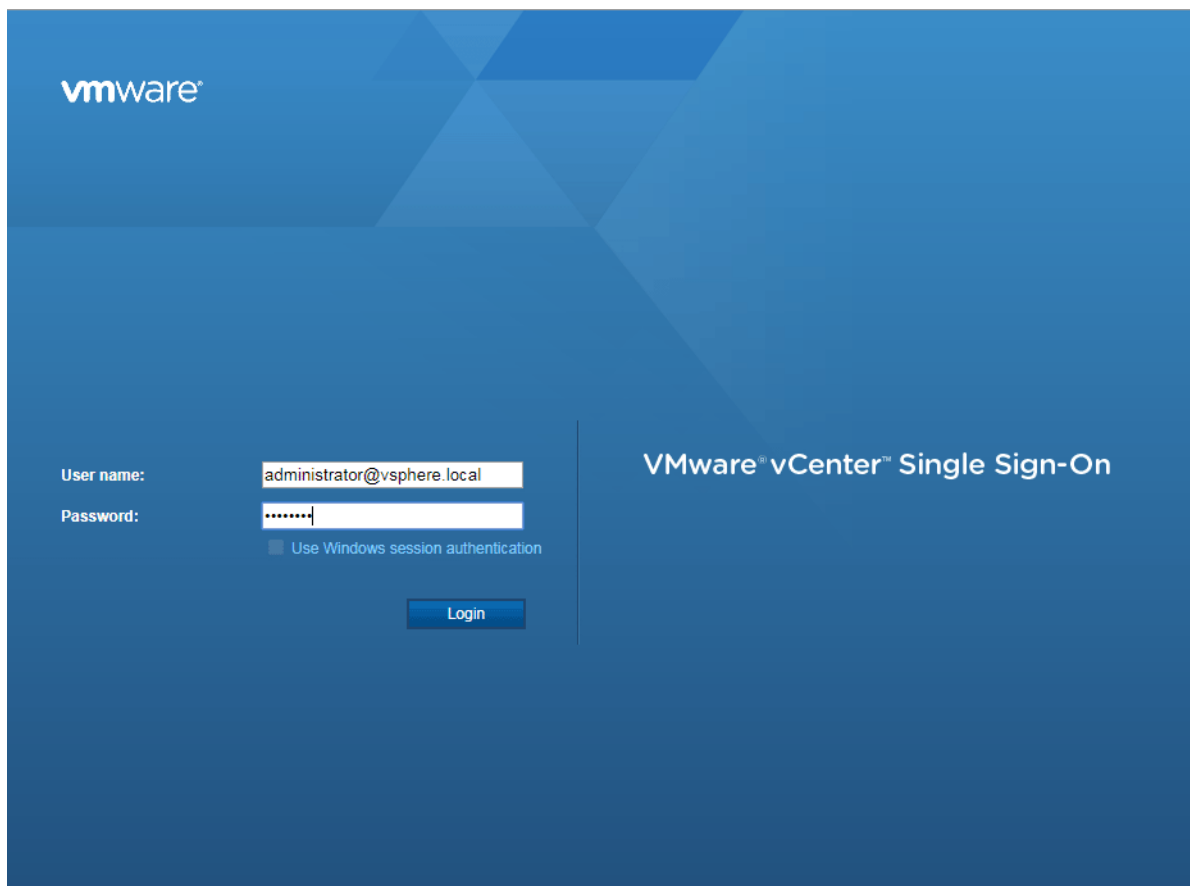
Step 2. If the login is successful, the vSphere Web Client dashboard is displayed, as shown in Figure 15.

Figure 15 **vSphere Web Client**
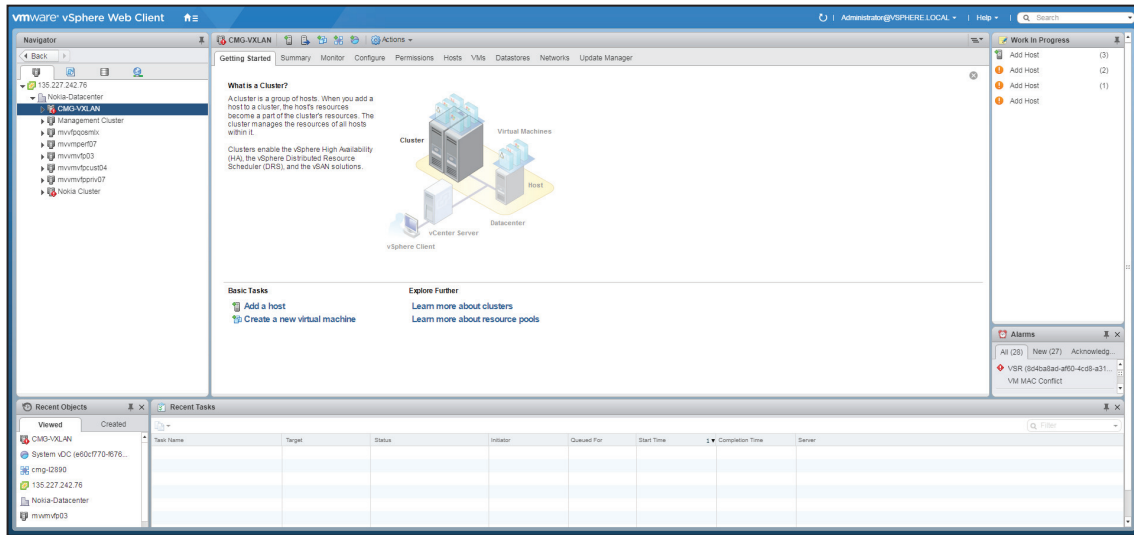
sc0024

6.4.2 Create Networks

Perform the following steps to create virtual switches for providing connectivity to VSR-I instances.

For VSR-I deployment, a virtual switch should be created for each of the following networks that are needed:

- one management network. This network should use E1000.
- up to nine external data networks. Each of these networks can use SR-IOV or PCI passthrough or VMXNET3.

Step 1. Navigate to the data center where the VSR-I will be deployed. Right click on the data center and choose New Distributed Switch from the menu, as shown in Figure 16.

→ **Note:** The VSR-I can be connected over a standard or a distributed switch.

Figure 16   **Data Center Page**
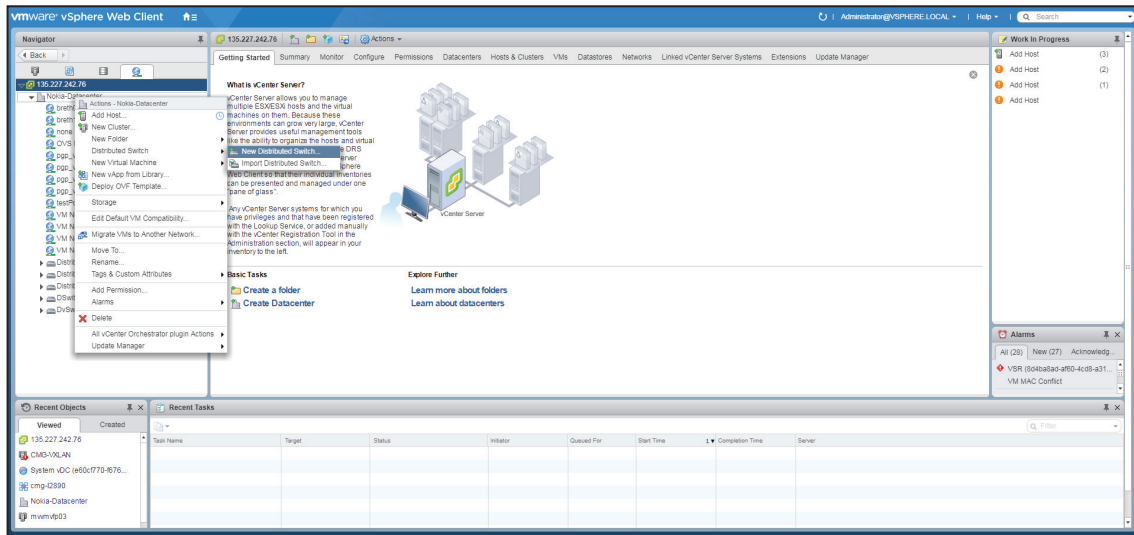
sc0026

Step 2.   Input a name for the distributed switch, as shown in Figure 17.

Figure 17   **Network Name**
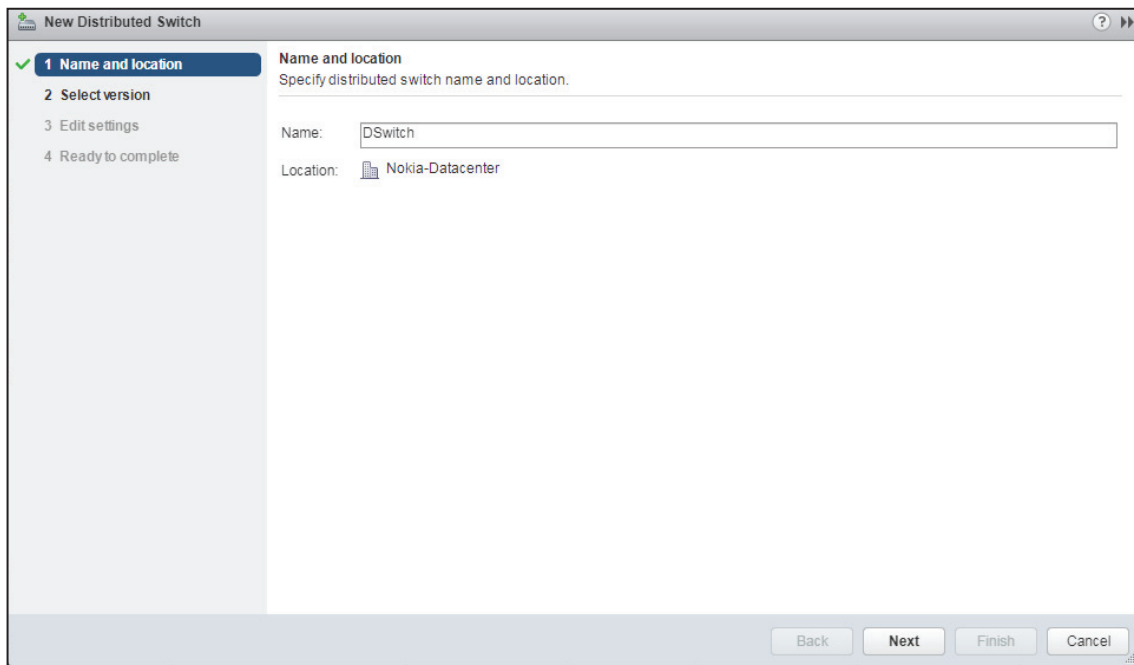
sc0027

Step 3. In the Select Version tab, select a distributed switch version, as shown in Figure 18.
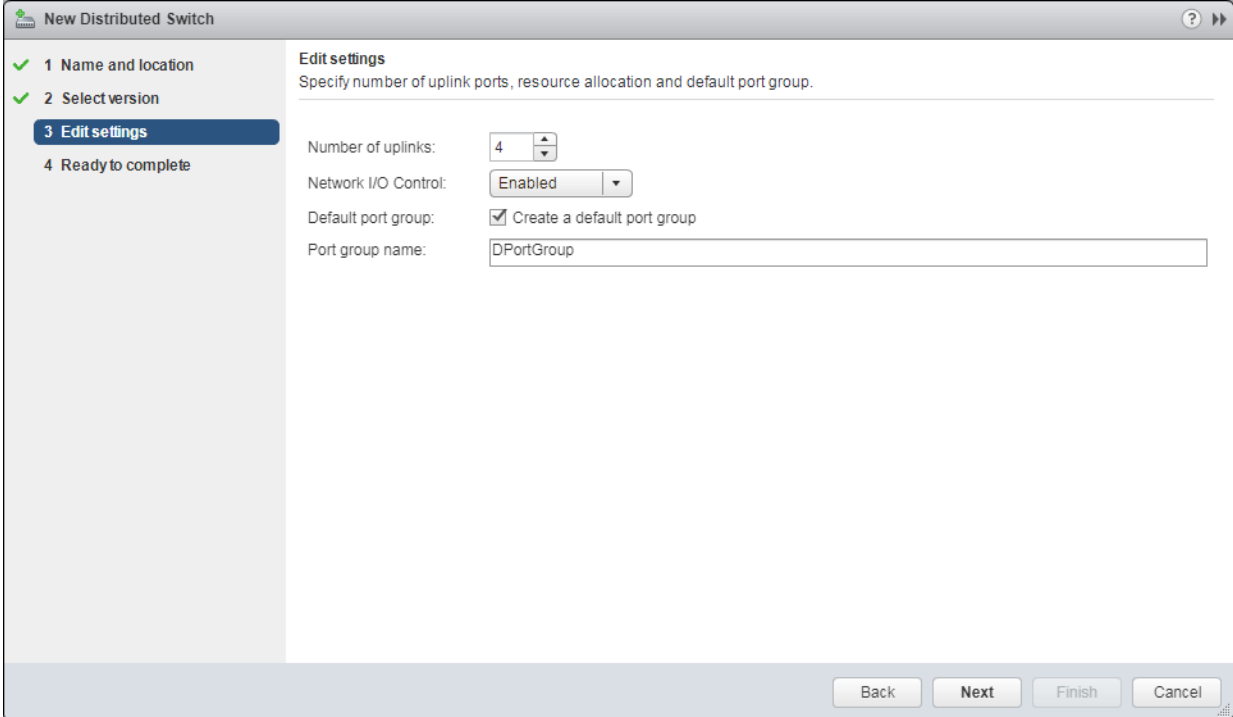
Figure 18 **Distributed Switch Version**
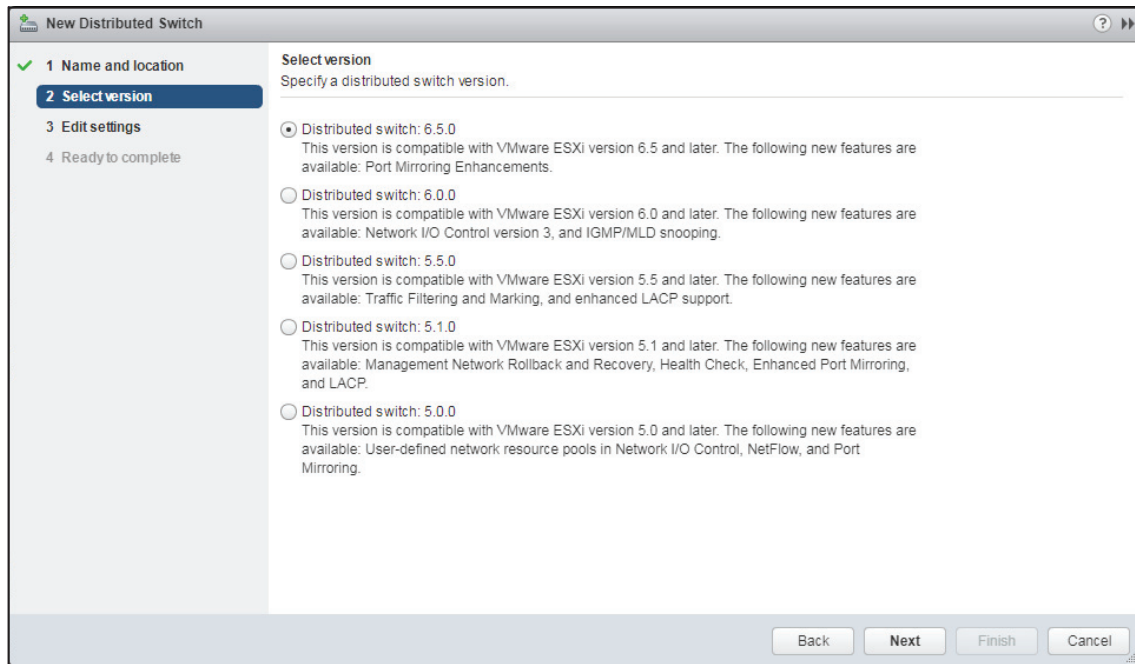
sc0028

Step 4. In the Edit Settings tab, specify the number of uplink ports, the resource allocation, and the default port group, as shown in Figure 19.

Figure 19 **Uplink Ports**

Step 5. In the Ready to Complete tab, review the settings before selecting Finish, as shown in Figure 20.

Figure 20   **Review Settings**
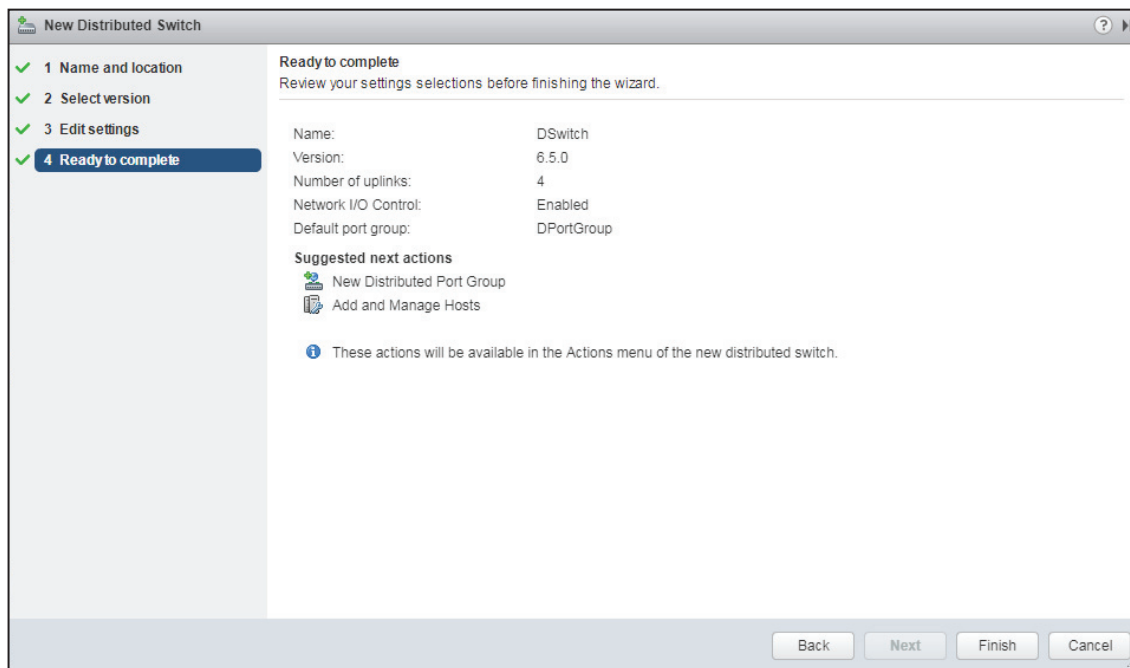
sc0029

Step 6.   In the Navigator, right click on the distributed port group and choose Edit Settings... from the menu, as shown in Figure 21.

Figure 21   **Edit Settings**
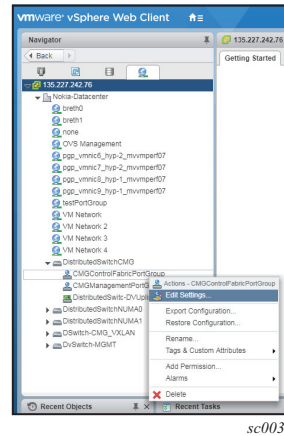
sc0030

Step 7.   In the Security tab, set each option to Accept as shown in Figure 22.

Figure 22   **DPortGroup Panel**
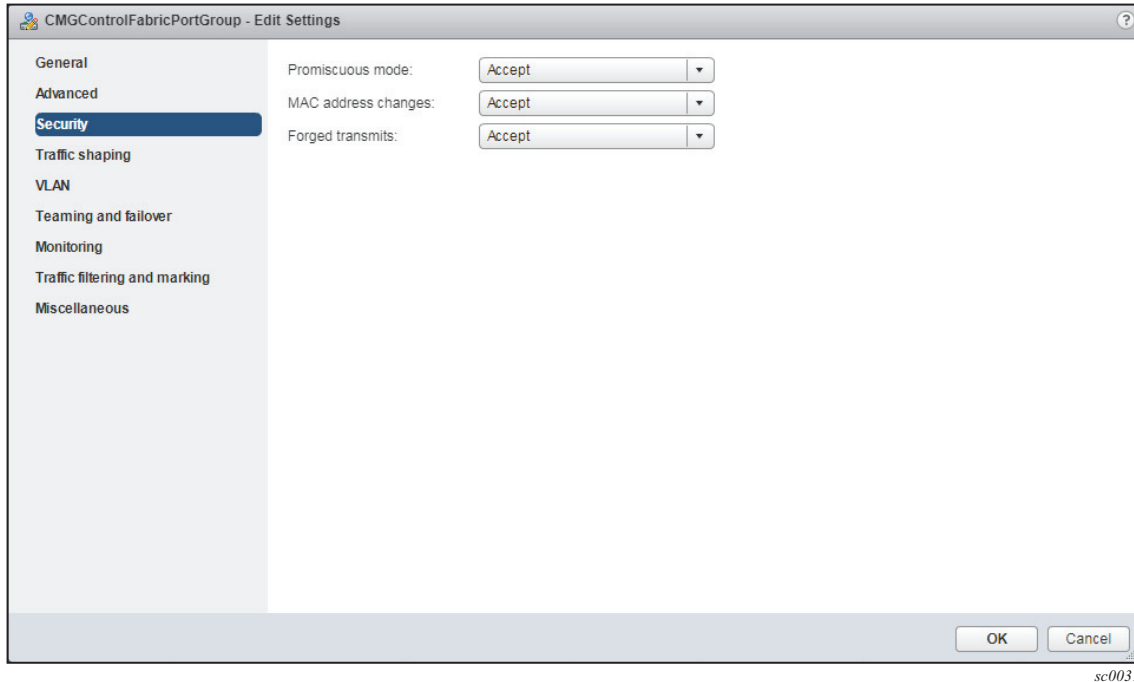
sc0031

Step 8. In the VLAN tab, select the VLAN type and VLAN trunk range, as shown in Figure 23.

Figure 23 **VLAN Type**
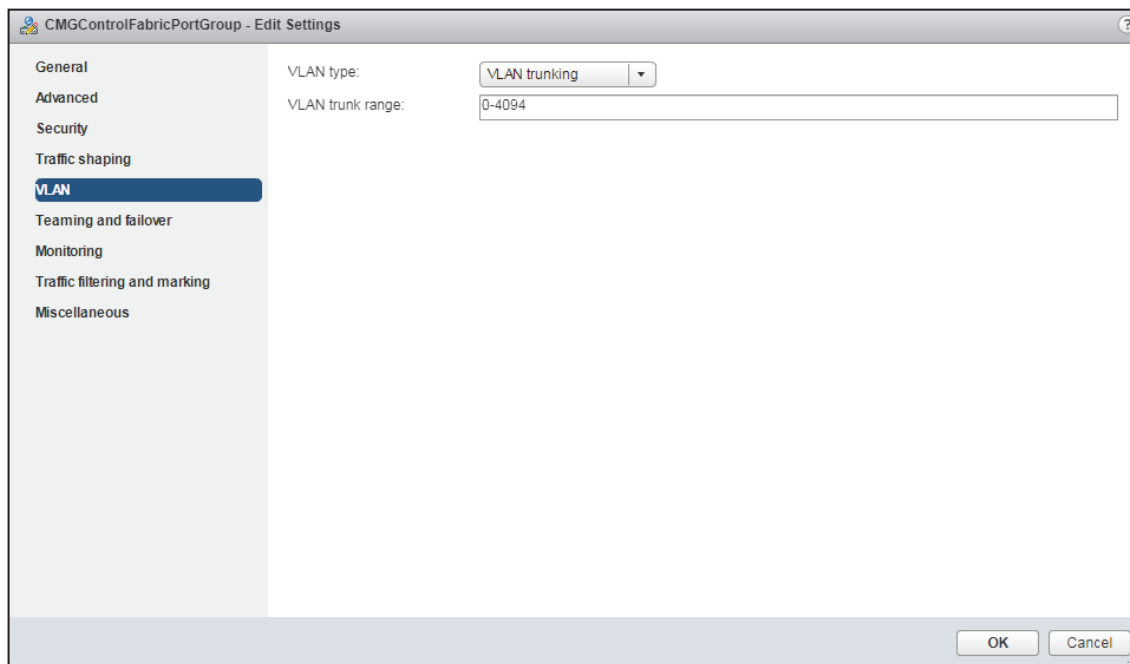
sc0032

Step 9. In the Teaming and Failover tab, set the options, as shown in Figure 24.

Figure 24   **Teaming and Failover**
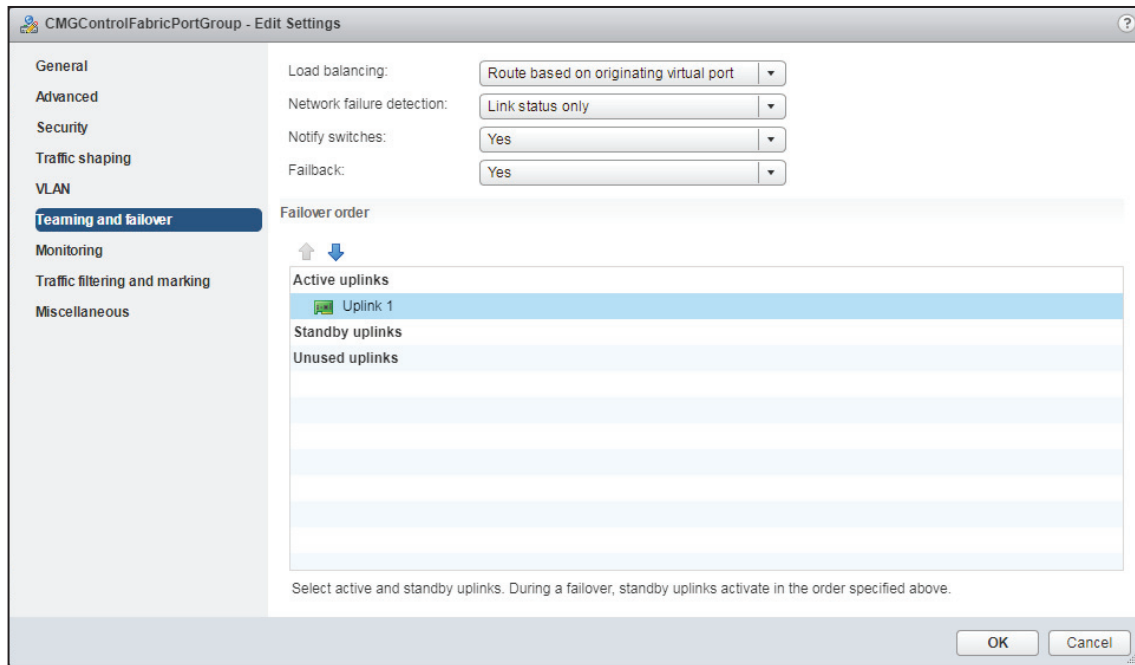
Step 10. In the Navigator, right click on the distributed switch and choose Edit Settings... from the menu as shown in Figure 25.

Figure 25   **Distributed vSwitch Settings**
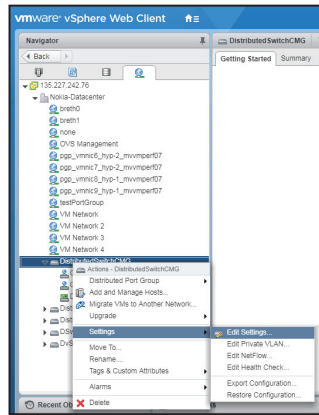
sc0034

Step 11. In the Advanced tab, set the options, as shown in Figure 26.

Figure 26   **DSwitch Advanced Options**
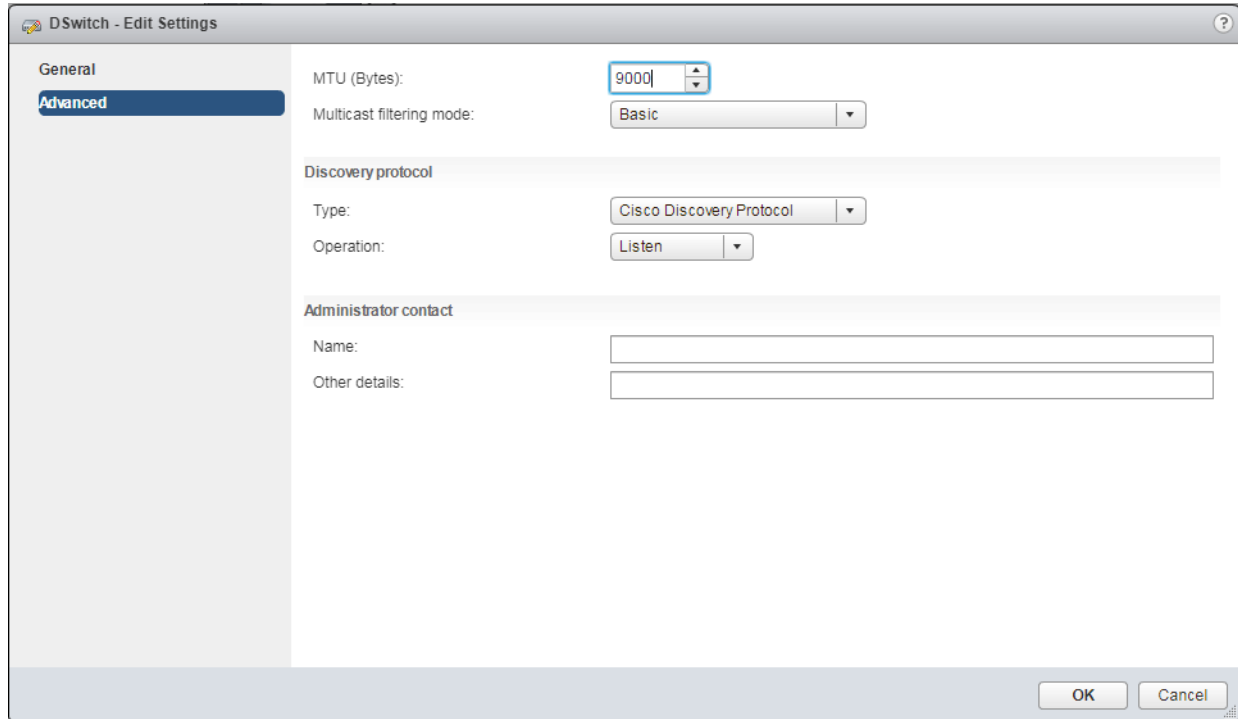
Step 12. In the Navigator, right click on the cluster and choose Add Host... from the menu, as shown in Figure 27.

Figure 27   **Add Host**
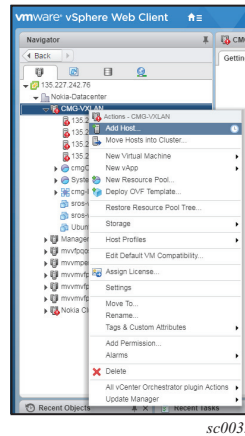
sc0035

Step 13. In the Name and location tab, enter the ESXi host IP address, as shown in Figure 28.

Figure 28   **ESXi Host IP Address**
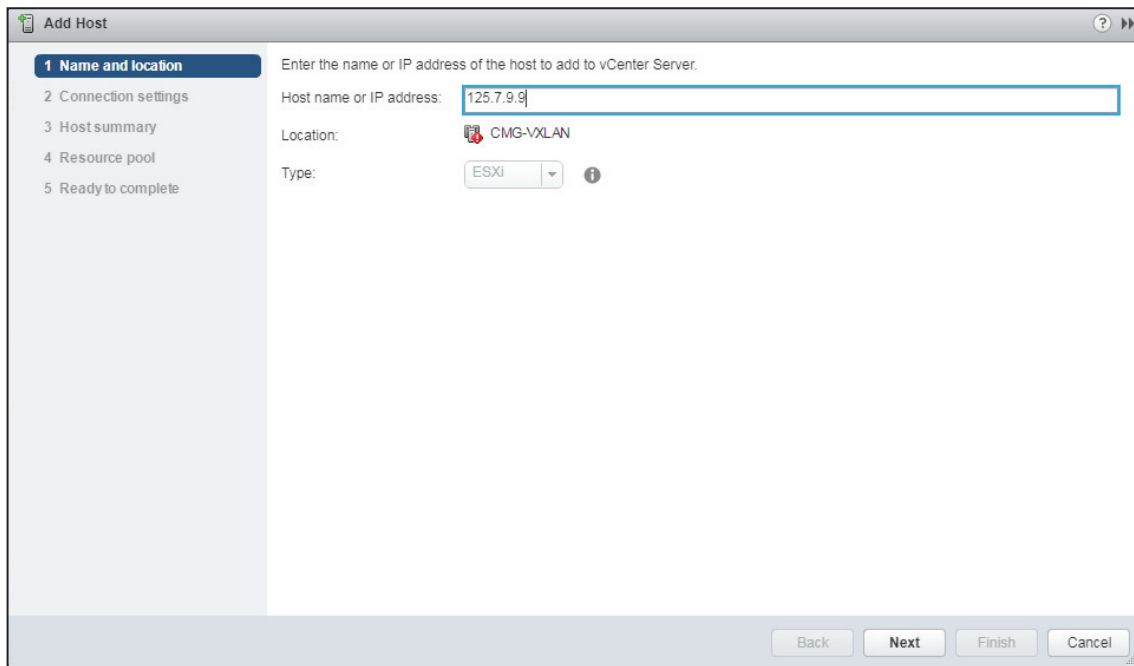
sc0036

Step 14. Start the process of adding the ESXi host to the distributed switch, as shown in Figure 29.

Figure 29   **Add New Hosts to the Distributed Switch**
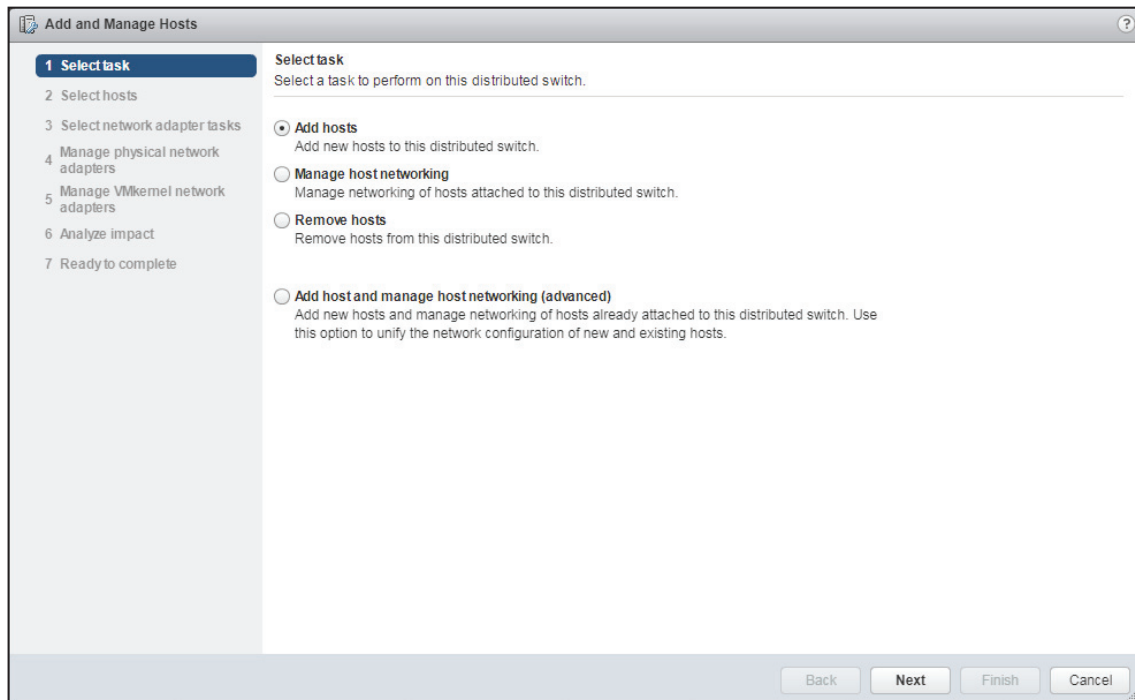
sc0037

Step 15. In the Select hosts tab, click on the "+" button to add New Hosts..., as shown in Figure 30.

Figure 30 **Select Hosts to Attach to the Switch**
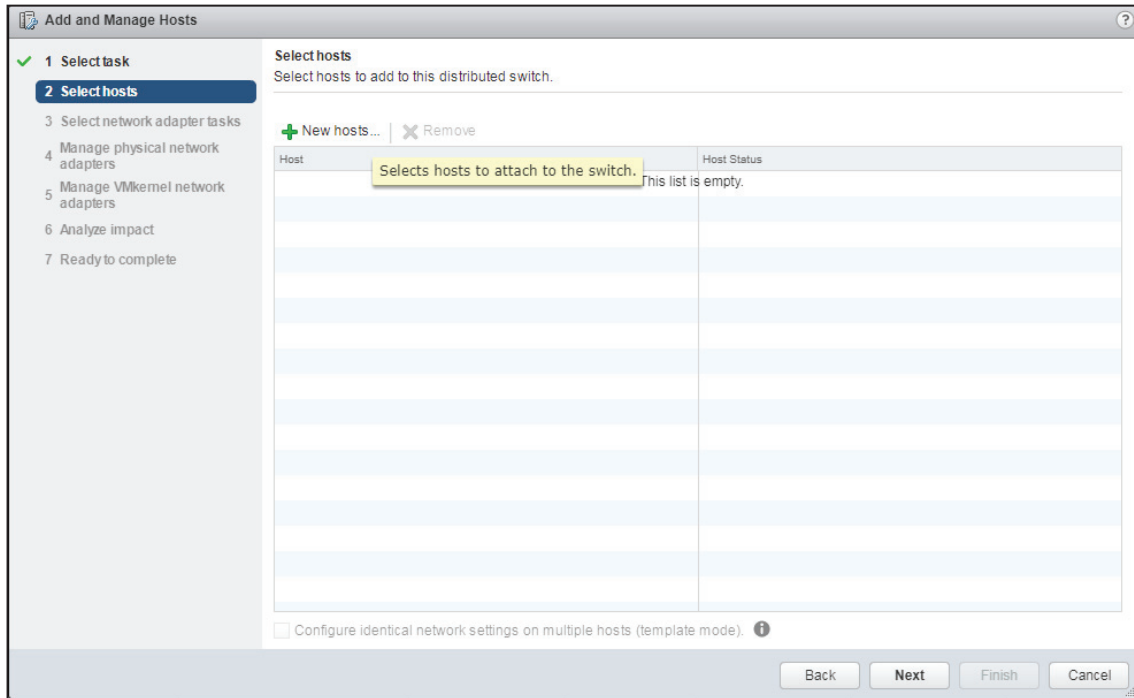
sc0038

Step 16. Select the check box for the ESXi host, as shown in Figure 31.

Figure 31 **Check box for ESXi Hosts**
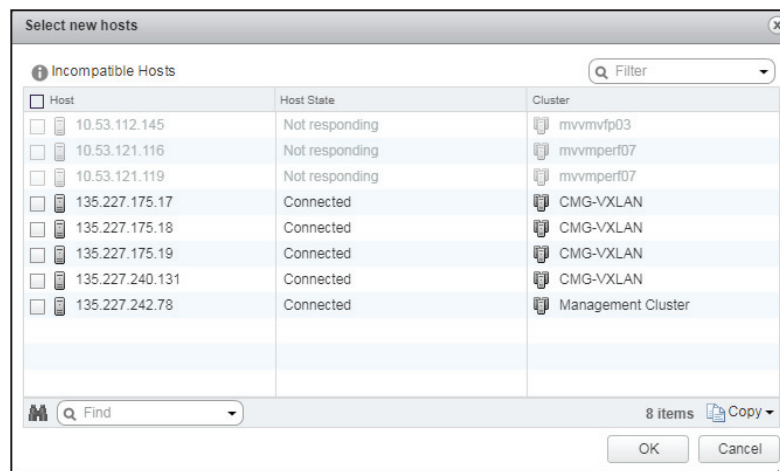
sc0039

Step 17. In the Select network adapter tasks tab, select Manage physical adapters, as shown in Figure 32.

Figure 32   **Manage Physical Adapters**
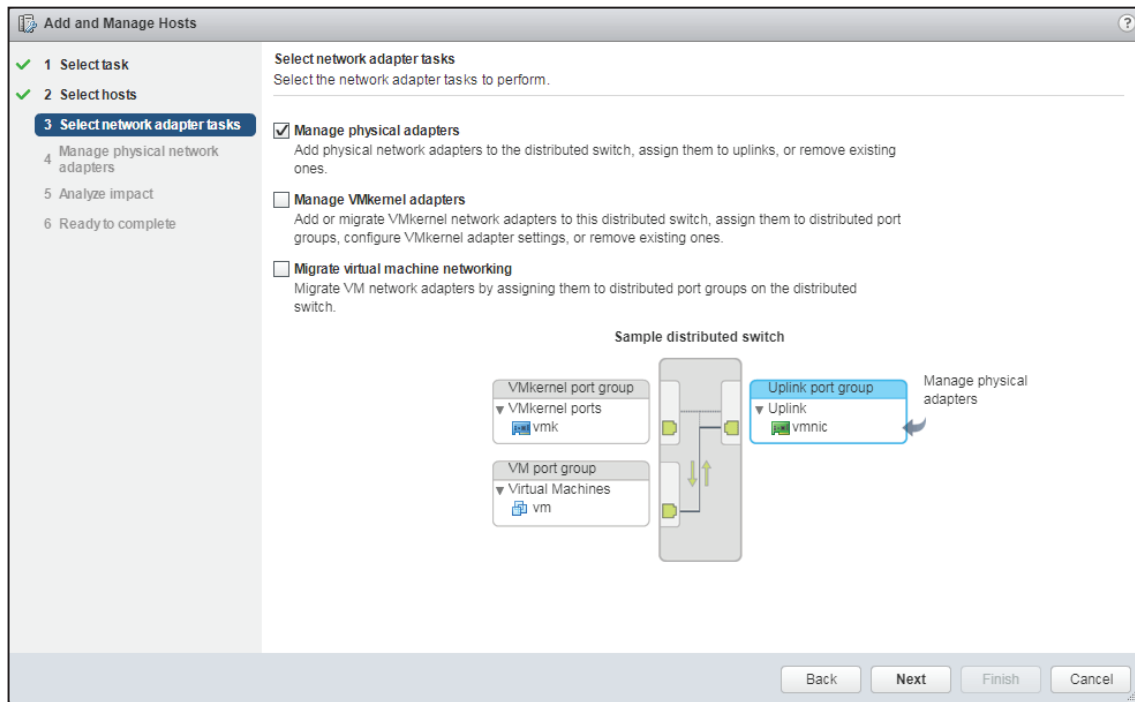
Step 18. In the Manage physical network adapters tab, add the physical network adapters to the distributed switch, as shown in Figure 33.

Figure 33   **Add Physical Network Adapters**
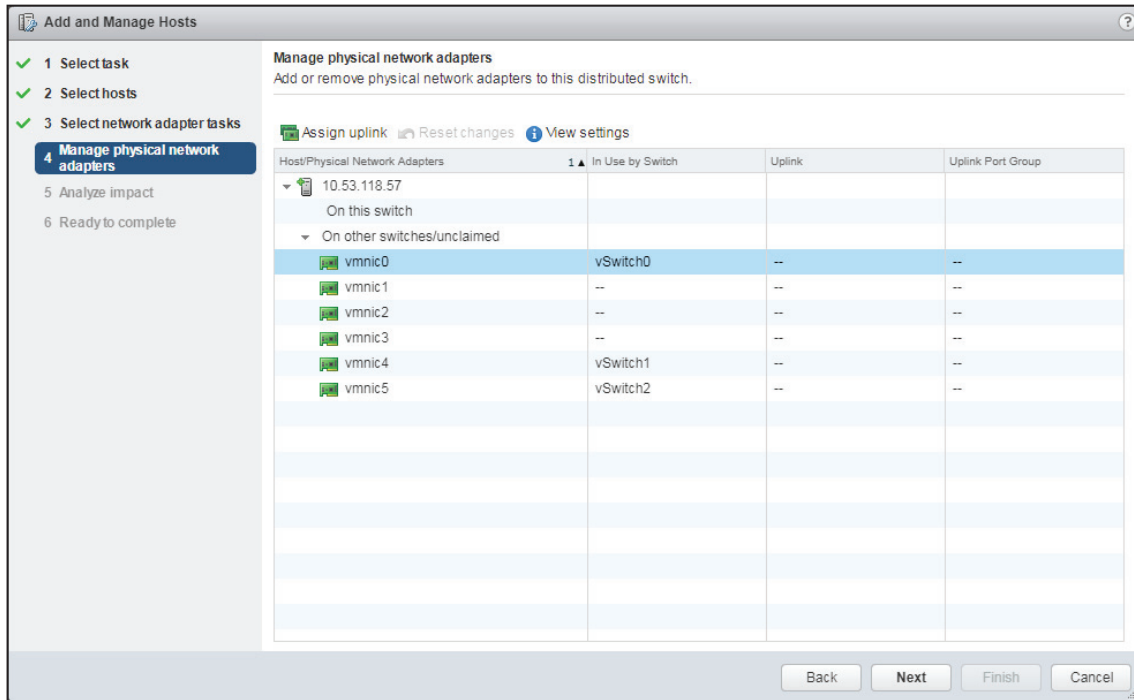
sc0041

6.4.3   Create the VSR-I VM

Perform the following steps to create the VSR-I VM.

Step 1. In the Navigator, right click on the cluster and choose New Virtual Machine from the menu, as shown in Figure 34.

Figure 34   **New Virtual Machine**

sc0042
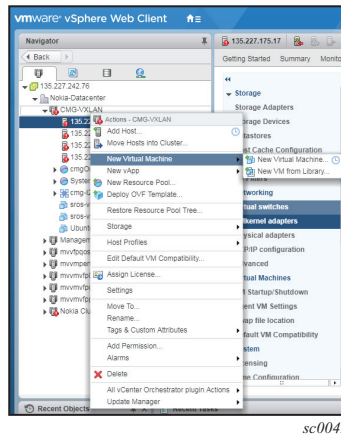
Step 2.   In the Select a creation type tab, select Create a new VM, as shown in
Figure 35.

Figure 35   **New Virtual Machine**
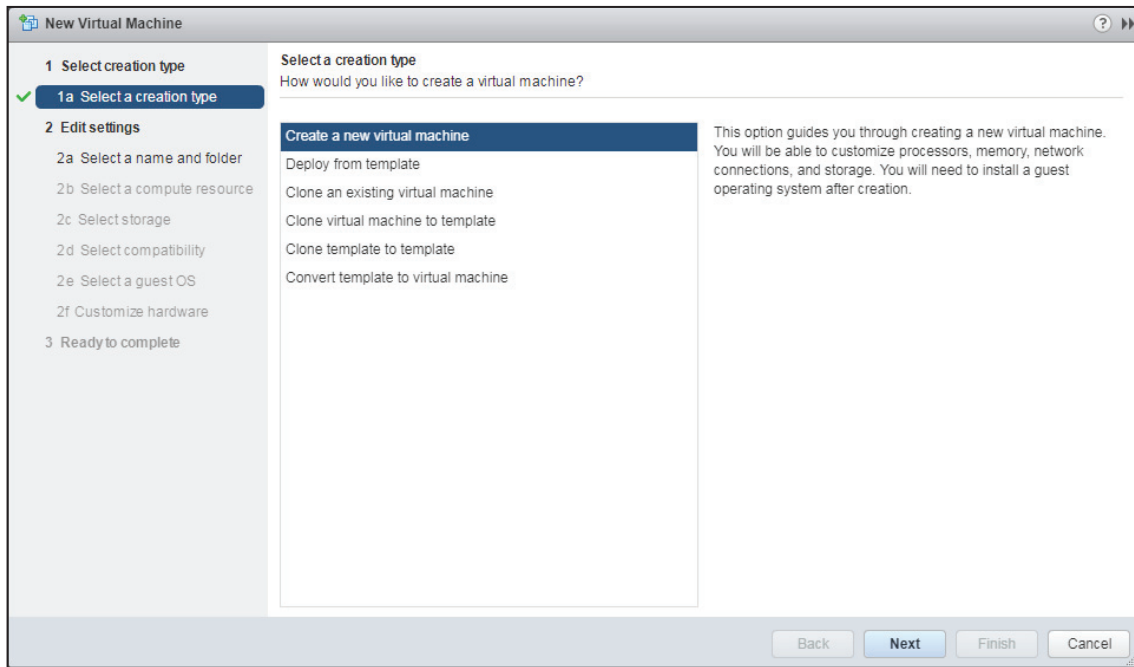
sc0043

i. In the Select a name and folder tab, select a name for the VM and a data center location for the VM, as shown in Figure 36.

Figure 36   **Select a Name and Folder**
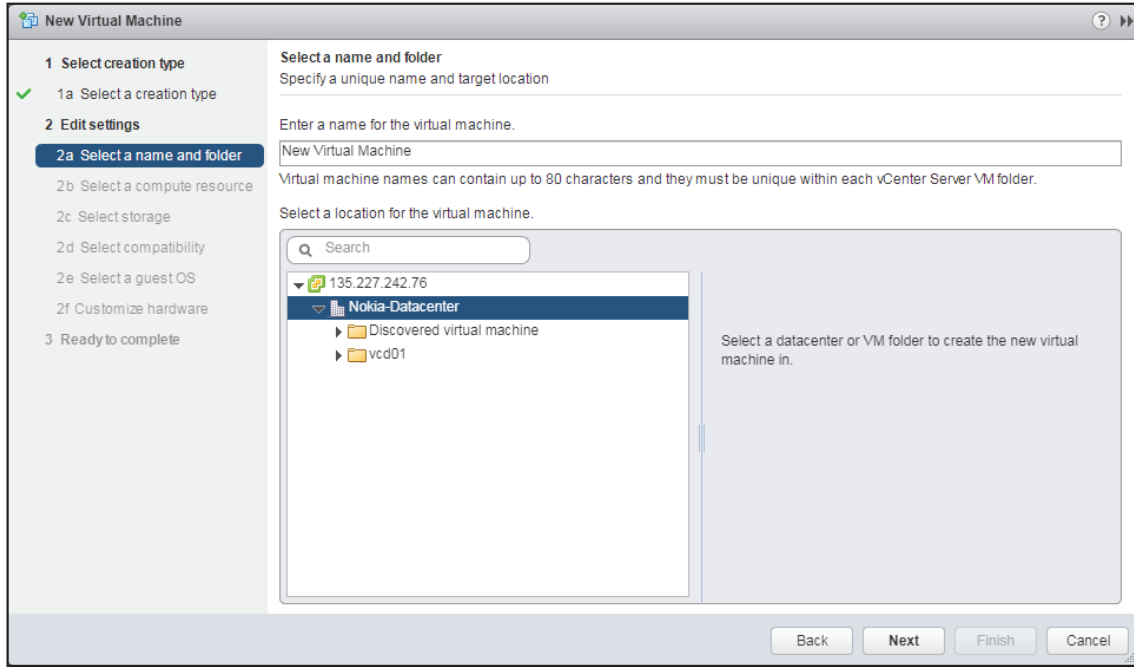
sc0044

ii. In the Select a compute resource tab, select the cluster or host where you want to run the VM as shown Figure 37.

Figure 37   **Select a Compute Resource**
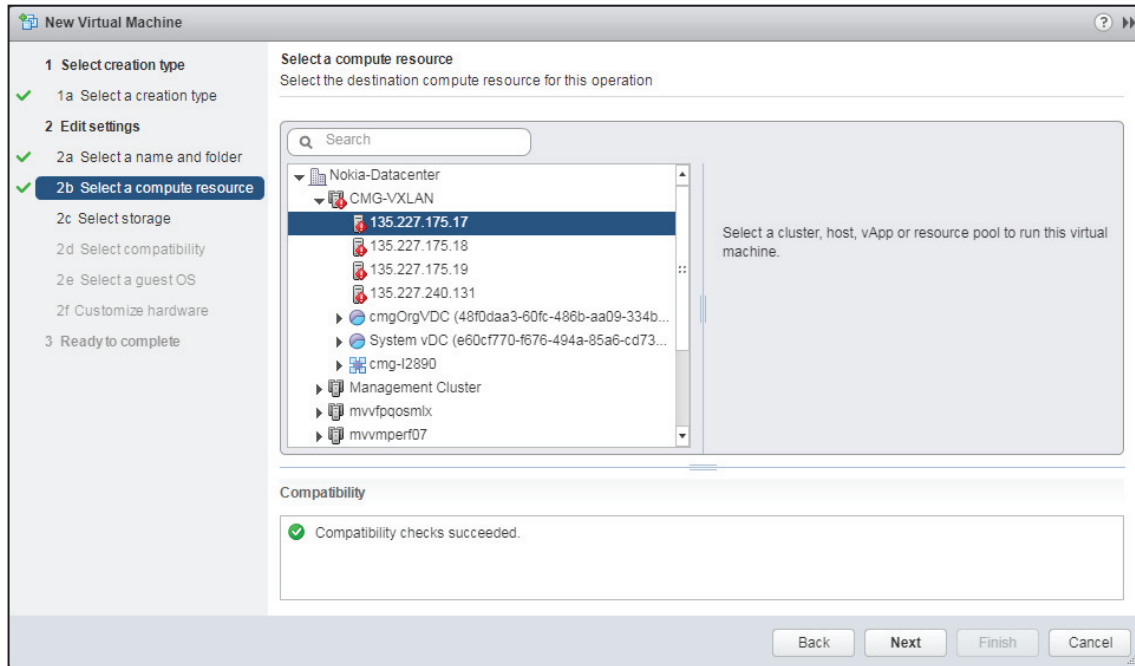
iii. In the Select storage tab, choose the data store to use for storage of the VM files as shown in Figure 38.

Figure 38 **Select Storage**
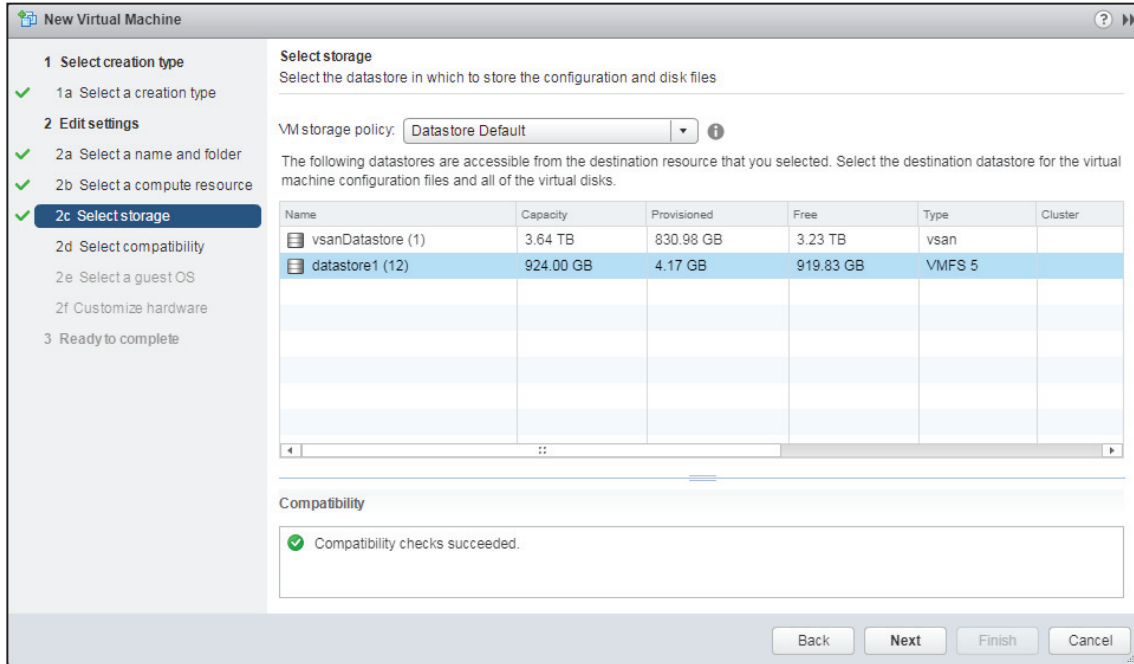
sc0046

iv. In the Select compatibility tab, choose Compatible with ESXi 6.5 and later, as shown in Figure 39.

Figure 39 **Select Compatibility**
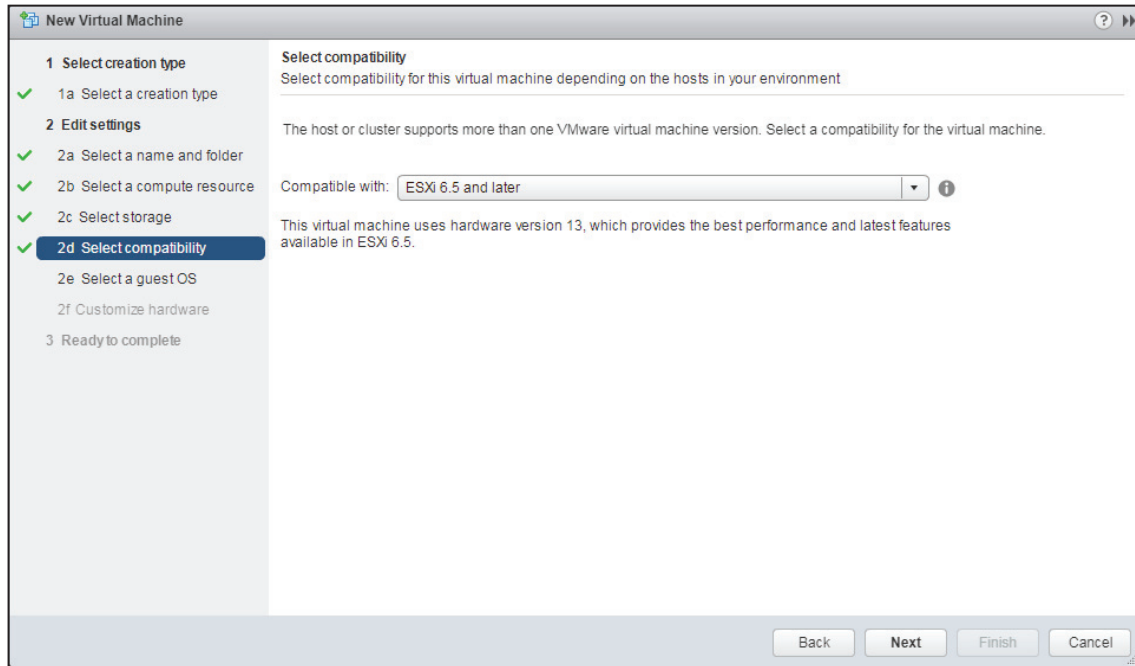
sc0047

v. In the Select a guest OS tab, fill in the family as "Other", the Version as "Other (64 bit)", and guest OS name as desired, as shown in Figure 40.

Figure 40   **Select a Guest OS**
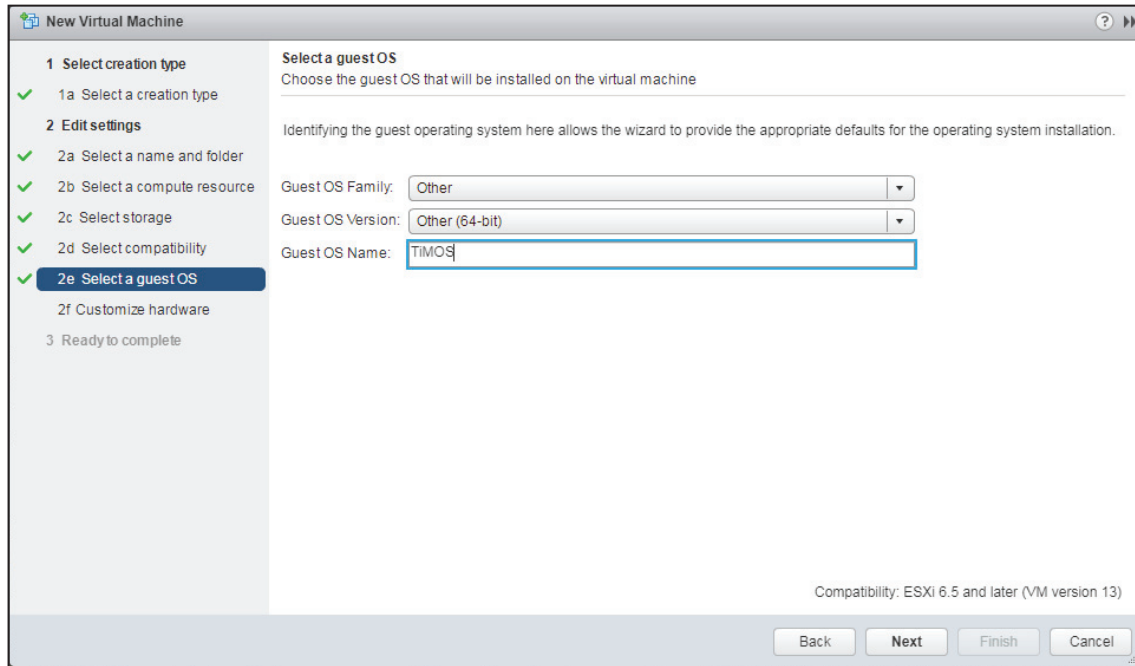
sc0048

Step 3. In the Customize hardware tab, customize the VM hardware, as shown in Figure 41.

 i. In CPU configuration, select the option to "Expose hardware assisted virtualization to the guest OS".

 ii. In Memory configuration, all memory should be reserved using the option "Reserve All Guest Memory (All locked)".

 iii. Modify hard disk configuration if needed. By default, the first disk (IDE 0:0) that is created emulates the CF3 (cf3:\) drive; it should be created from the VMDK disk image contained in the OVA package.

Figure 41   **Customize Hardware**
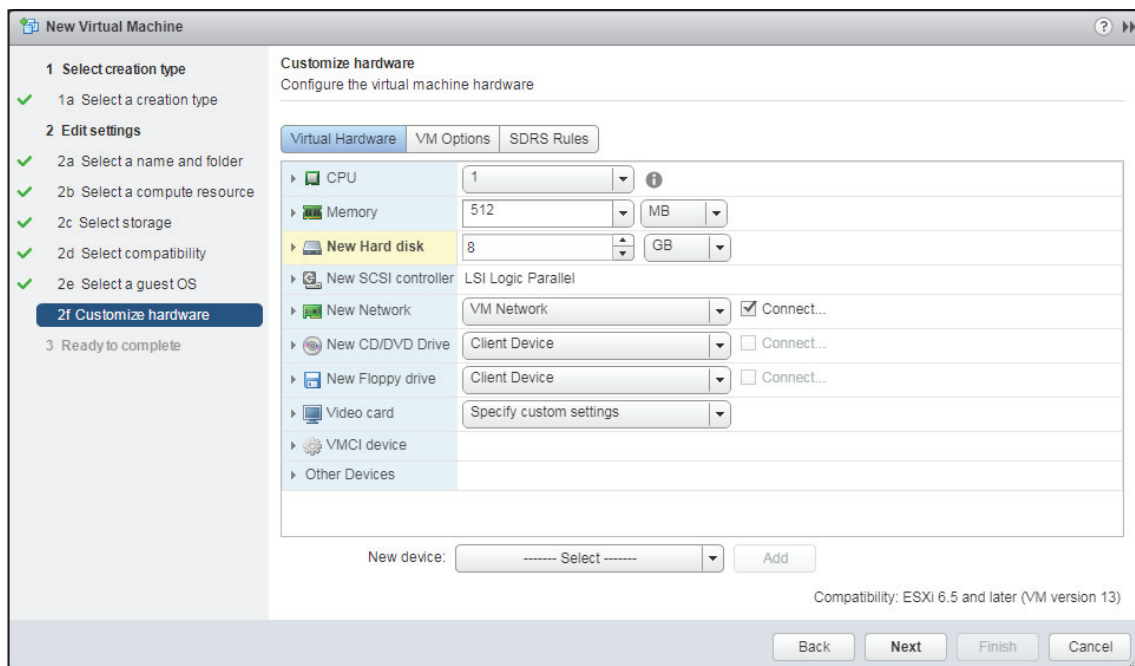
sc0050

iv. Additional disks mapped to CF1 (IDE 0:1) and CF2 (IDE 1:0) are optional and may be created, as shown in Figure 42.

Figure 42   **Select the Device Type**
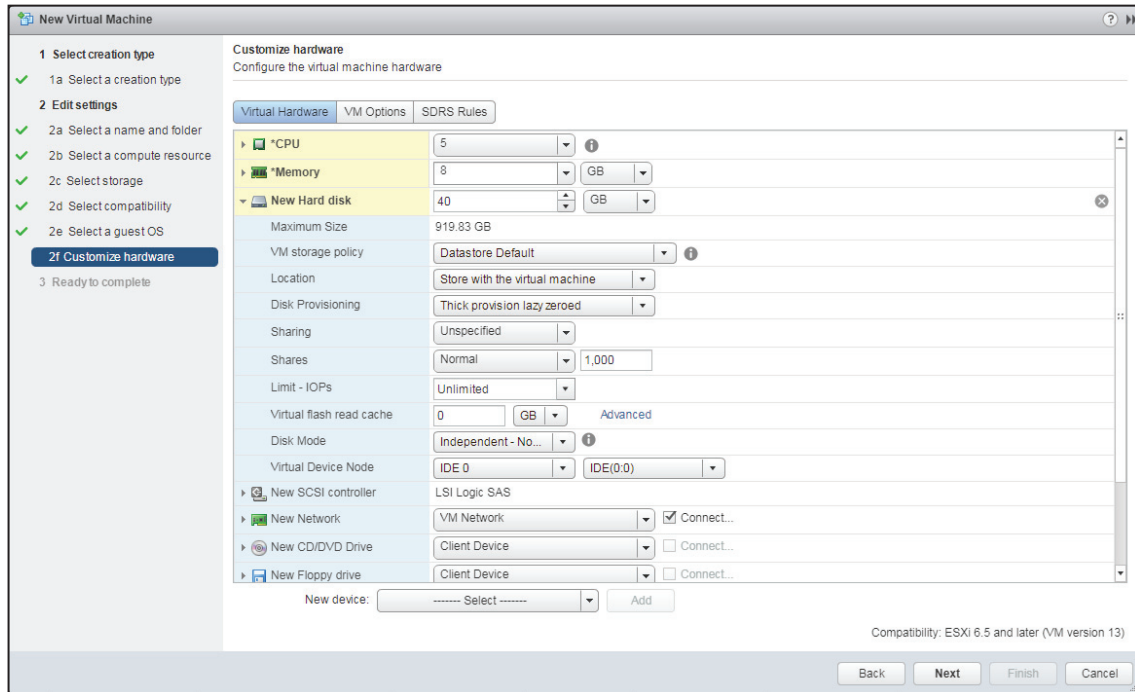
sc0051

Step 4. Add network adapters as needed and connect them to the VSR-I VM. A maximum of 10 vNICs are supported by each VM. The mapping of vNIC ports to SR OS ports is described in Guest vNIC Mapping in VSR VMs.

i. Click on "Add" to add a new network.

ii. Select the adapter type, as shown in Figure 43. Choose E1000, SR-IOV passthrough, or VMXNET3, depending on the adapter number. The adapter with the lowest PCI address must use E1000. The other adapters can use any of the technologies except VMXNET2 and e1000e.

Figure 43   **Edit Settings**
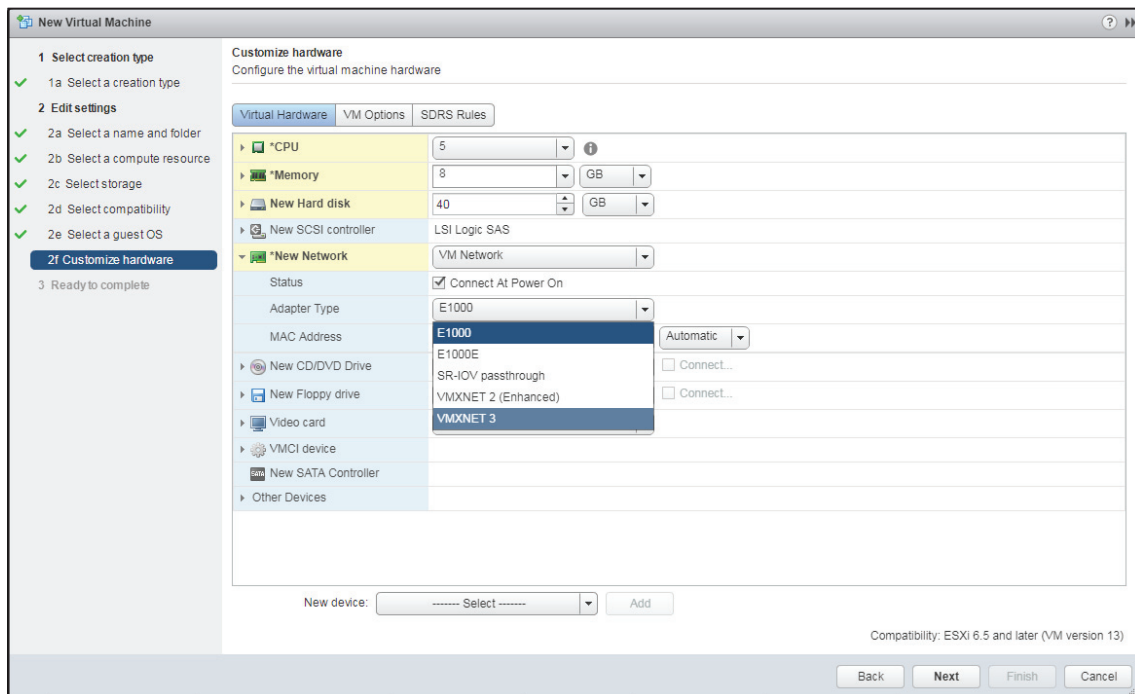
sc0052

iii. When selecting SR-IOV passthrough, the physical function (VF) must also be selected, as shown in Figure 44.

Figure 44 **Physical Function**

sc0053
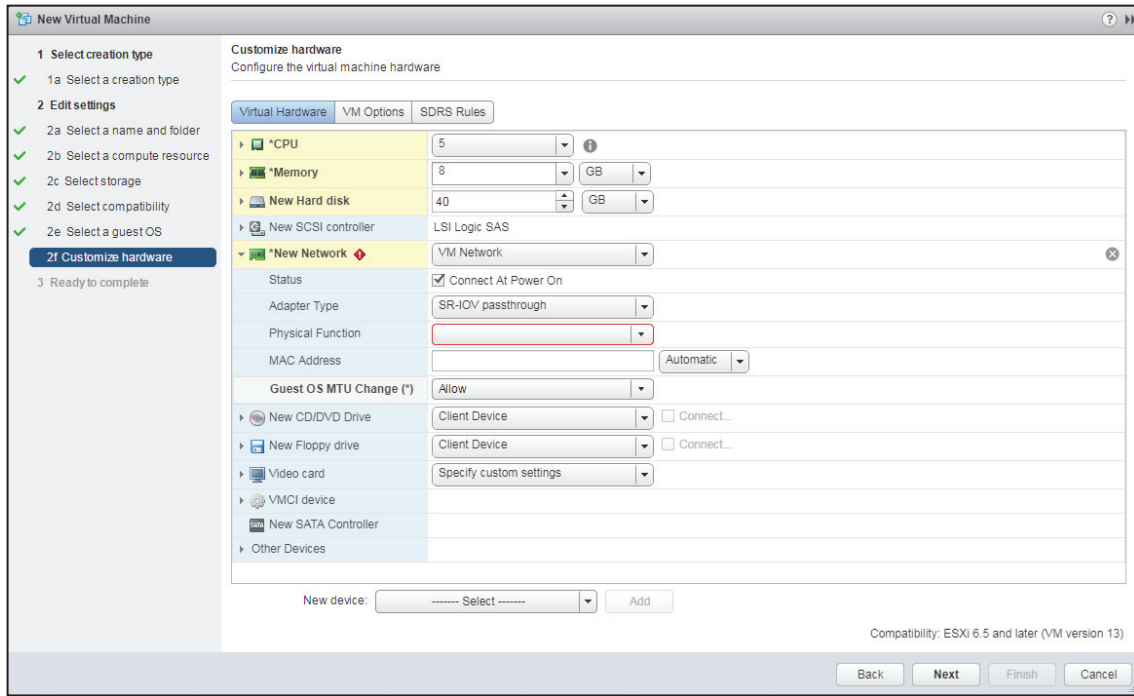
iv. Attach the network adapter to the corresponding port group.

Step 5. Click Next in the dialog box and proceed to finalize the VM creation.

6.4.4   Customizing the VSR-I VM

6.4.4.1   Set Latency Sensitivity

Setting latencySensitivity to "high" is mandatory when VSR-I VMs are deployed without CPU pinning directives. This ensures that each vCPU of the VM is pinned to a physical CPU and that no other workload is assigned to that physical CPU even if hyper-threading is not disabled on the host machine. This configuration also disables interrupt coalescing automatically.

To set latencySensitivity to "high":

Step 1. Right click on the VSR VM and choose Edit Settings... from the menu.

Step 2. Click on the VM Options tab.

Step 3. For Latency Sensitivity, choose High from the menu, as shown in Figure 45.

Figure 45 **Latency Sensitivity**
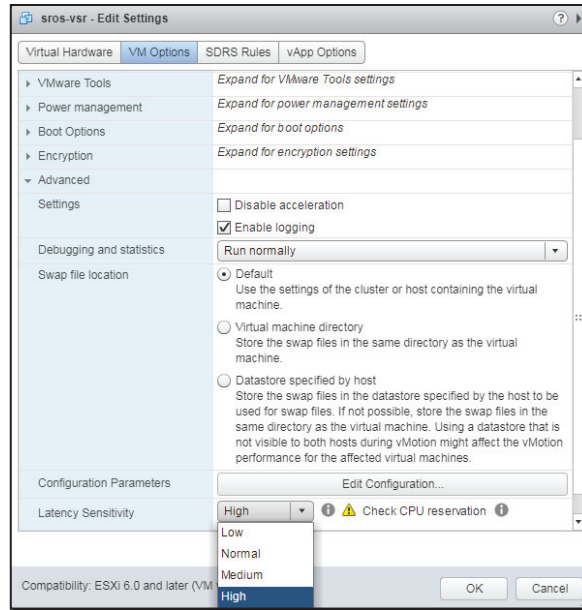
sc0054

6.4.4.2   Set NUMA Node Affinity

Every VSR VM should have numa.nodeAffinity set to either "0" or "1" to ensure that all vCPUs and all memory are allocated to the VM from NUMA node "0" or NUMA node "1". It is critical that all resources are NUMA aligned.

To change the NUMA affinity for the VM:

Step 1. In the VM Options tab, select Edit Configuration...

Step 2. In the Value box for numa.nodeAffinity, enter "0" or "1" as appropriate (the NUMA node ID). After this step the configuration parameters may be similar to those shown in Figure 46.

Figure 46 **NUMA node ID**
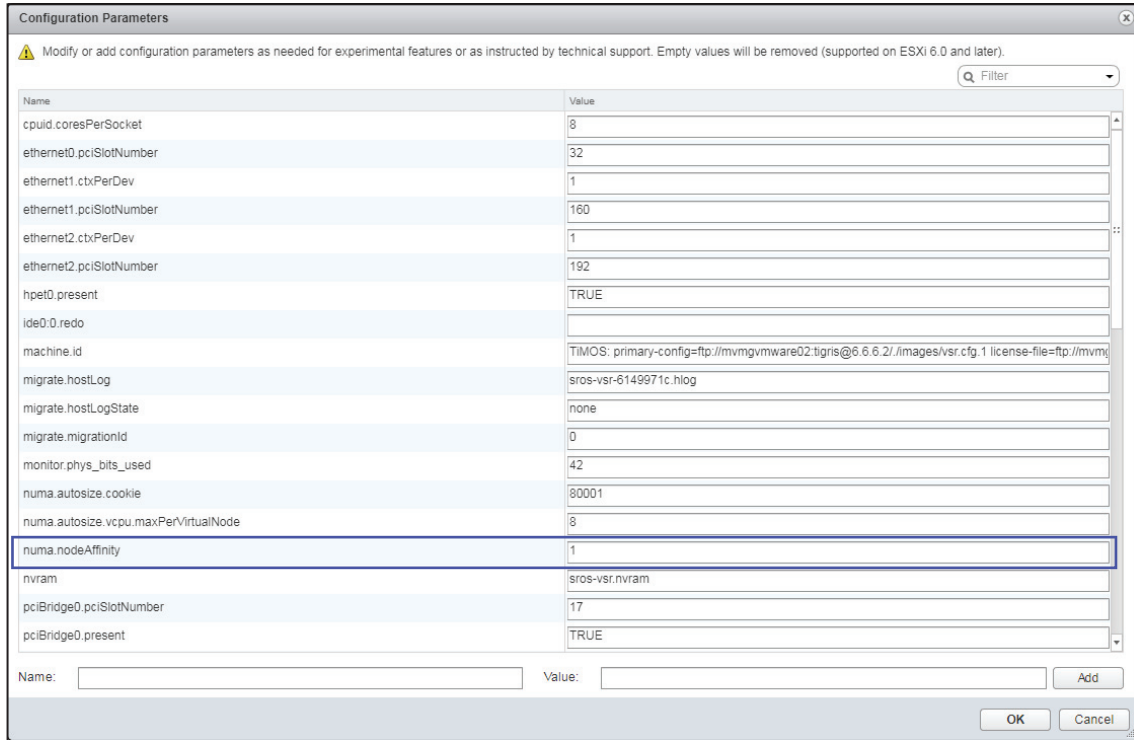
sc0055

6.4.4.3 Configure the SMBIOS Configuration String

VMware allows the SMBIOS configuration string to be passed to a VSR-I VM by specifying it as a machine.id value. To add a machine.id to a VSR-I VM or edit the default machine.id provided in the OVF, perform the following steps:

Step 1. In the VM Options tab, select Edit Configuration...

Step 2. In the Value box for machine.id, edit the SMBIOS configuration string, making sure that it starts with the string TIMOS. See Sysinfo for information about a properly formatted SMBIOS string and the allowed parameters for VSR VMs.

6.4.4.4 Configure CPU Pinning for Deployment on a Hyper-Threaded Host

If you are deploying a VSR-SeGW or VSR-AA application on a hyper-threaded ESXi host and you want the highest possible performance for that application, perform the following steps:

Step 1. The machine.id setting must include a hyperthreading=1 directive. See Configure the SMBIOS Configuration String.

Step 2. The sched.cpu.latencySensitivity setting must be set to "High". See Set Latency Sensitivity.

Step 3. The sched.cpu.affinity.exclusive setting must be set to "True".

    i. In the VM Options tab, select Edit Configuration...

    ii. In the Name box, enter sched.cpu.affinity.exclusive.

    iii. In the Value box, enter True; see Figure 47.

Figure 47   **Configuring Parameters for CPU Pinning**
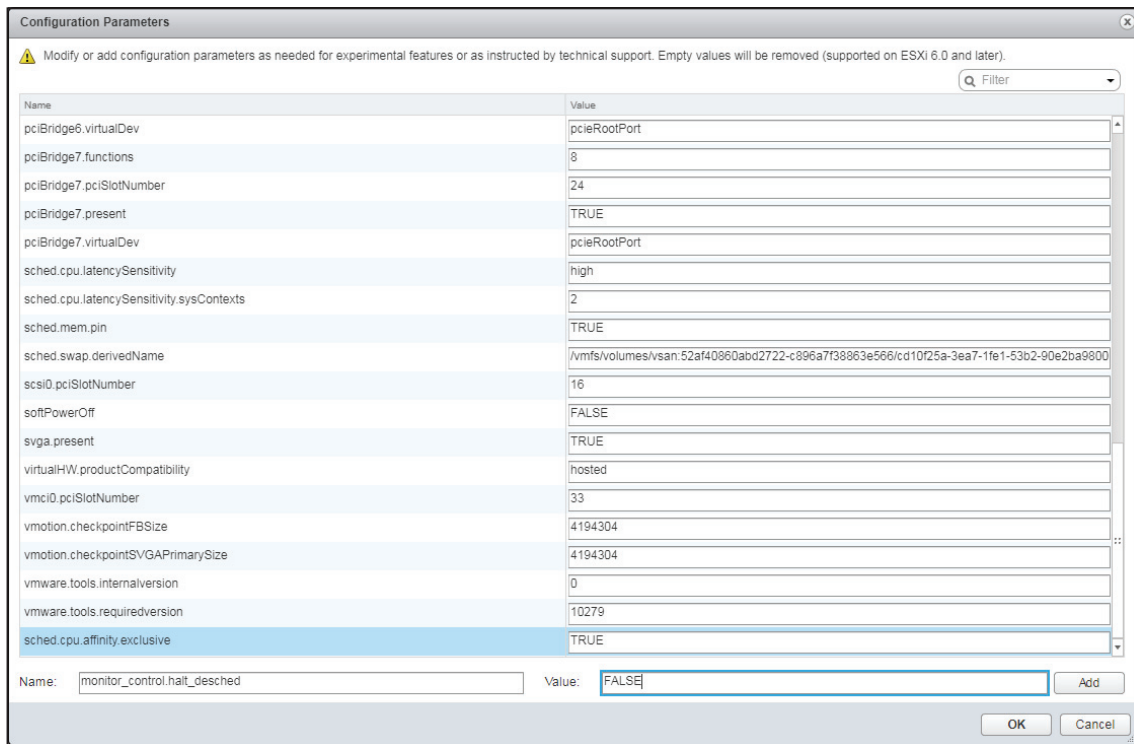
sc0057

Step 4. The monitor_control.halt_desched setting must be set to "False".

    i. In the VM Options tab, select Edit Configuration...

    ii. In the Name box, enter monitor_control.halt_desched.

       iii. In the Value box, enter False; see Figure 47.

Step 5. Pin each vCPU to a pCPU hyper-thread, following the guidelines provided in NUMA.

       i. In the VM Options tab, select Edit Configuration...

       ii. For each vCPU you need a row in the table such as following: sched.vcpu0.affinity (Name) = 4 (Value); see Figure 48.

➡️ **Note:** The first entry value must be "4".

Figure 48   **Configuring vCPU Pinning**
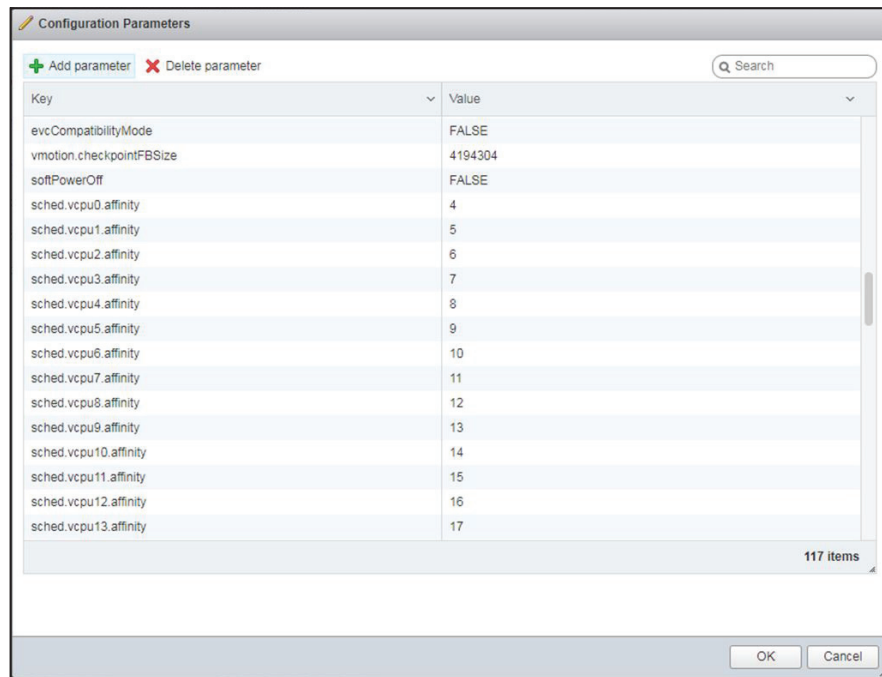
sc0070

6.4.5 Start the VSR-I VM

After creating and configuring the VSR-I VMs the VSR instance can be brought into service by powering it on (Actions → Power on), as shown in Figure 49.

Figure 49   **Power On**

sc0056
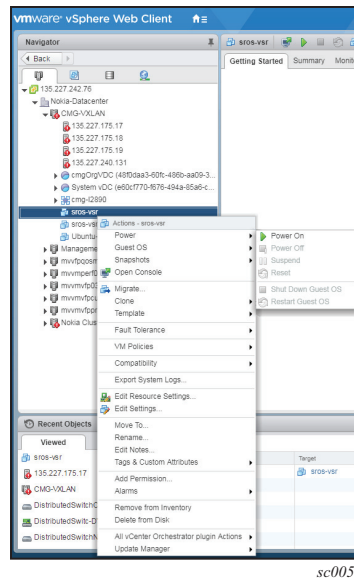
7 Virtual Machine Configuration Parameters

VM configuration parameters should be set differently, depending on how a VM is deployed.

7.1 VMs Deployed on KVM Compute Hosts

7.1.1 Virsh Command Line and Libvirt Domain XML File

When a VM is deployed on a KVM compute host using virsh command line tools and a manually edited libvirt domain XML file, VM configuration parameters should be set as follows:

- The SMBIOS product string should include **slot=A chassis=VSR-I card=cpm-v** in addition to other necessary attributes.
- The SMBIOS product string should include a **vsr-deployment-model=route-reflector** directive if the VSR is intended to be used only as a control plane BGP route reflector (no transit forwarding requirements).
- The SMBIOS product string should include a **vsr-deployment-model=queue-scale** directive if the VSR requires thousands of egress traffic queues (such as a scaled VSR-BNG). This increases the amount of VM memory used for buffering but also has some performance trade-offs.
- The SMBIOS product string should include a **vsr-deployment-model=high-packet-touch** directive if the VSR is used for SeGW applications. This increases performance of these applications on a hyperthreaded host.
- The SMBIOS product string should include a **control-cpu-cores=N** directive to reserve N pCPU cores (2*N hyper-threads) for control plane processing. Without this directive N takes a value of 1, which does not provide sufficient control plane capability for most applications. N has a maximum value of 16.

➡️ **Note:** Configuration of **control-cpu-cores** is not necessary if the **vsr-deployment-model** is set to **route-reflector**.

- When a VM is deployed on any host machine with a CPU having 18 or more CPU cores, the ACPI feature must be enabled in the domain XML file.

- When a VM supporting any application is deployed on a hyperthreaded Linux host:
    - the domain XML file must enable the ACPI feature
    - the domain XML file must include a CPU topology section that exposes two threads per core to the guest.
    - The domain XML file must pin vCPU0 and vCPU1 to sibling threads of the same pCPU core, vCPU2 and vCPU3 to sibling threads of some other pCPU core, etc.
- The number of vCPU cores must be in the range 2-56. For some applications, such as NAT, the minimum number of vCPUs is 3. The recommended number of vCPUs depends on performance requirements; consult your Nokia representative for further details.
- The amount of vRAM must be in the range of 4GB-64GB. The vRAM must be entirely backed by 1GB huge pages. Minimum vRAM memory depends on the application (see Table 13).
- In a multi-socket system supporting NUMA the vCPUs and vRAM of the VM should be allocated from one single NUMA node, and this should also be the NUMA node associated with the SR-IOV and PCI pass-through NIC ports used by the VM.
- The vhost-net threads should be pinned to vCPU cores not used by any virtual VSR machine (using the "emulatorpin" setting).
- The UUID parameter must match the one encoded in the license.
- The VM can be assigned up to 16 vNIC interfaces. The first virtual interface (lowest PCI bus/device/function address) must be VirtIO. The remaining virtual interfaces can be any combination of VirtIO, SR-IOV and PCI pass-through.

7.1.2 OpenStack

When a VM is deployed on a KVM compute host using OpenStack (Nova boot or Heat stack create) certain parameters should be set as follows:

- A config_drive must be attached to the VM and the user-data written to this config_drive (in "RAW" format) must be a valid SMBIOS product string that starts with "TIMOS:".
- The SMBIOS product string read from the config_drive should include **slot=A chassis=VSR-I card=cpm-v** in addition to other necessary attributes.
- The SMBIOS product string read from the config_drive should include a **vsr-deployment-model=route-reflector** directive if the VSR is intended to be used only as a control plane BGP route reflector (no transit forwarding requirements).

- The SMBIOS product string read from the config_drive should include a **vsr-deployment-model=queue-scale** directive if the VSR requires thousands of egress traffic queues (such as a scaled VSR-BNG). This increases the amount of VM memory used for buffering but also has some performance trade-offs.

- The SMBIOS product string read from the config_drive should include a **vsr-deployment-model=high-packet-touch** directive if the VSR is used for SeGW applications. This increases performance of these applications on a hyperthreaded host.

- The SMBIOS product string read from the config_drive should include a **control-cpu-cores=N** directive to reserve N pCPU cores (2*N hyper-threads) for control plane processing. Without this directive N takes a value of 1, which does not provide sufficient control plane capability for most applications. N has a maximum value of 16. NOTE: configuration of control-cpu-cores is not necessary if the vsr-deployment-model is set to route-reflector.

- The Nova flavor extra_spec **hw:cpu_policy** should always be set to **dedicated** in order to dedicate one pCPU core (or thread) to each vCPU of the VM.

- When a VM supporting any application is to be deployed on a non-hyperthreaded Linux host the Nova flavor extra_spec **hw:cpu_threads_policy** should be set to **isolate**.

- When a VM supporting any application is to be deployed on a hyperthreaded Linux host:

    – the Nova flavor extra_spec **hw:cpu_threads_policy** should be set to **prefer**

    – the Nova flavor extra_spec **hw:cpu_threads** should be set to "2". **hw:cpu_sockets** and **hw:cpu_cores** should also be set appropriately.

- The number of vCPUs for the Nova flavor used to create the VM must be in the range from 2-56. For some applications (such as NAT) the minimum number of vCPUs is 3. The recommended number of vCPUs depends on performance requirements; consult your Nokia representative for further details.

- The amount of vRAM for the Nova flavor used to create the VM must be in the range of 4GB-64GB. Minimum vRAM memory depends on the application (see Table 13).

- The Nova flavor extra_spec **hw:mem_page_size** must be set to **1048576** in order to allocate 1GB huge pages to the VM.

- The **allowed_address_pairs** attribute of Neutron ports that use VirtIO should specify "0.0.0.0/0" for ip_address in order to bypass restrictive anti-spoofing rules.

- For Neutron ports that use SR-IOV the **binding:vnic-type** attribute should be set to **direct**.

- Up to three Cinder volumes may be attached (as block devices) to a VM. The "hda" device corresponds to CF3, the "hdb" device corresponds to CF1 and the "hdc" device corresponds to CF2. If a Cinder volume is created from the QCOW2 disk image provided by Nokia the size is limited to 1189 MB even if the requested volume size is greater.

Table 13   **VM Memory Requirements by Function Mix**

Functions	Minimum Memory for VM (GB)	Notes
PE	4	
RR	4	
NAT	profile-1: 28 profile-2: 56	
PE + NAT	20	
Transit AA (Res 8K mode)	8	
PE + AA (VPN 1K mode)	8	
IPsec	20	Requirement to achieve maximum number of IPSec tunnel scale
IPsec + AA firewall	30	Requirement to support AA at maximum number of IPsec tunnel scale
LNS	24	
BNG without ISA	16 32 (>32k queues)	
BNG with BB-ISA	24 48 (>32k queues)	
vRGW	24	
WLAN-GW	32 48 (>32k queues)	
WLAN-GW + AA	40 48 (>32k queues)	

7.2   VMs Deployed on a VMware ESXi 6.0 or 6.5 Compute Host using vSphere

When a VM is deployed on a VMware ESXi 6.0 or 6.5 compute host using the vSphere Web Client interface to the vCenter Server, certain VM configuration parameters should be set as follows:

- The machine.id setting should include **slot=A chassis=VSR-I card=cpm-v** in addition to other necessary attributes.
- The machine.id setting should include a **vsr-deployment-model=route-reflector** directive if the VSR is intended to be used only as a control plane BGP route reflector (no transit forwarding requirements).
- The machine.id setting should include a **vsr-deployment-model=queue-scale** directive if the VSR requires thousands of egress traffic queues (such as a scaled VSR-BNG). This increases the amount of VM memory used for buffering but also has some performance trade-offs.
- The machine.id setting should include a **vsr-deployment-model=high-packet-touch** directive if the VSR is used for SeGW applications. This increases performance of these applications on a hyperthreaded host.
- The machine.id setting should include a **control-cpu-cores=N** directive to reserve N pCPU cores (2*N hyper-threads) for control plane processing. Without this directive N takes a value of 1, which does not provide sufficient control plane capability for most applications. N has a maximum value of 16.

→ **Note:** Configuration of control-cpu-cores is not necessary if the **vsr-deployment-model** is set to **route-reflector**.

- If the host machine has hyperthreading disabled, the **latencySensitivity** setting of the VM should be set to **high**. Navigate to VM Settings→ Options → Advanced General → Configuration Parameters and set **sched.cpu.latencySensitivity** to **high**. No CPU pinning is required in this case.
- When a VM supporting any application is deployed on a hyperthreaded ESXi host:
  - The machine.id setting must include a **hyperthreading=1** directive
  - The **latencySensitivity** is not required unless VMXNET3 networking is used
  - Edit the VMX file to pin each consecutive vCPU to a pCPU hyperthread. For example to pin vCPU0 to pCPU4 you would include the following line in the VMX: sched.vcpu0.affinity = "4".

–   Add the following settings to the VMX after the pinning statements: sched.cpu.affinity.exclusive = "TRUE"

–   monitor_control.halt_desched = "FALSE"

• The number of vCPU cores must be in the range 2-56. For some applications (such as NAT) the minimum number of vCPUs is 3. The recommended number of vCPUs depends on performance requirements. Consult your Nokia representative for further details.

• The vCPU cores assigned to the VM must be reserved. Navigate to VM Settings→ Resources → CPU → Reservation and set the value to the number of vCPUs assigned to the VM multiplied by the physical CPU clock speed.

• The vCPU cores assigned to the VM must not have limited CPU shares. Navigate to VM Settings → Resources → CPU → Limit and specify the value **unlimited**.

• The amount of vRAM must be in the range of 4GB-64GB. Minimum vRAM memory depends on the application (see Table 13).

• In a multi-socket system supporting NUMA the vCPUs and vRAM of the VM should be allocated from one single NUMA node, and this should also be the NUMA node associated with the SR-IOV and PCI pass-through NIC ports used by the VM. Navigate to VM Settings → Options → Advanced General → Configuration Parameters and set numa.nodeAffinity to "0" (NUMA0) or "1"(NUMA1) as appropriate.

• Add or edit the uuid.bios line in the VMX file and assign the parameter the same value encoded in the license file.

• The VM can be assigned up to 10 vNIC interfaces. The first virtual interface (lowest PCI bus/device/function address) must be E1000. The remaining virtual interfaces can be any combination of E1000, VMXNET3, SR-IOV and PCI pass-through. NOTE – the first interface that is attached to the VM may not be the lowest numbered PCI device.

• A separate Tx thread should be allocated for each VMXNET3 vNIC interface. (By default ESXi uses one Tx thread per VM.) Navigate to VM Settings → Options → Advanced General → Configuration Parameters and give **ethernetX.ctxPerDev** a value of "1" for each VMXNET3 interface.

• If the VM has VMXNET3 vNIC interfaces then a minimum of two host physical CPU threads should be reserved for handing the Tx threads of the VMXNET3 vNIC interfaces (and also the Rx threads starting with ESXi 6.5). Navigate to VM Settings → Options → Advanced General → Configuration Parameters and give **sched.cpu.latencySensitivity.sysContexts** a value of "2" or greater; a NUMA node where the VM resides must have at least this number of unreserved cores.

7.3   Intel QuickAssist

To achieve maximum IPsec throughput with QAT, apply the following VM configuration guidelines.

- QAT is supported only on VSR-I using SR-IOV on a KVM hypervisor.
- QAT is a PCIe device.

  In general, a higher number of QAT PCIe lanes is helpful to achieve higher throughput. A PCH chipset with 24x PCIe lanes might result in higher throughput than a PCH chipset with 16x PCIe lanes, depending on the traffic pattern and core allocation.

- There are three Physical Functions (PF) per supported QAT hardware.

  The number of Virtual Functions (VF) per PF attached to the VSR must be the same for all three PFs.

- If hyperthreading (HT) is enabled on host, for each physical CPU core, assign only one sibling HT core to the VSR. Do not assign the other sibling HT core to any other VM.

- The number of QAT VFs must be equal to or larger than the number of virtual FP workers. QAT VFs from each QAT PF must be configured in XML in a round-robin order; for example: PF0_VF0, PF1_VF0, PF2_VF0, PF0_VF1, PF1_VF1, PF2_VF1, and so on.

- With the current generation of supported CPU, assigning more cores to VSR should result in higher IPsec throughput, depending on VSR core assignment.

- The number of vFP workers must be multiples of the number of QAT PFs assigned to the VSR-I VM.

8 VSR-I Lifecycle Management Using CBAM

8.1 Overview

Lifecycle management of VSR-I instances deployed on an OpenStack-based NFVI is supported by the CloudBand Application Manager (CBAM) software. A CBAM template package can be downloaded from Nokia OLCS and used to on-board a VSR-I into CBAM.

This chapter provides information about customizing the VSR-I template package for a specific NFVI environment and managing the lifecycle of a VSR-I instance using CBAM. CBAM terms are described in Appendix A: VSR Glossary of Key Terms.

8.2   Introduction to CBAM

CBAM is the component of the Nokia CloudBand suite that serves the role of a Virtualized Network Function Manager (VNFM) for VSR-I and many other Nokia VNFs. It supports an open template system for modeling and managing VNFs, based on ETSI NFV IFA 011 GS and OASIS TOSCA. CBAM uses HEAT Orchestration Templates (HOT) that describe the virtual resources needed by a VNF.

The VNF lifecycle management operations are modeled as a sequence of tasks executed by the built-in Mistral workflow engine. CBAM supports out-of-the-box basic workflows (for example, instantiation and termination) and the Mistral workflow engine provides flexibility to create advanced custom workflows (such as upgrading and healing). As part of Mistral workflows, commissioning tasks can leverage CBAM's built-in Ansible engine.

The set of resource templates, lifecycle management workflows, and Ansible playbooks are modeled in CBAM through a VNF descriptor (VNFD). The full set of CBAM artifacts for a VNF (VNFD, HOT, Mistral workflows, and Ansible playbooks) are referred to as the VNF package.

Figure 50 shows the high-level architecture of CBAM.

Figure 50   **CBAM Architecture**

sw0102

3HE 15837 AAAA TQZZA 01   Issue: 01
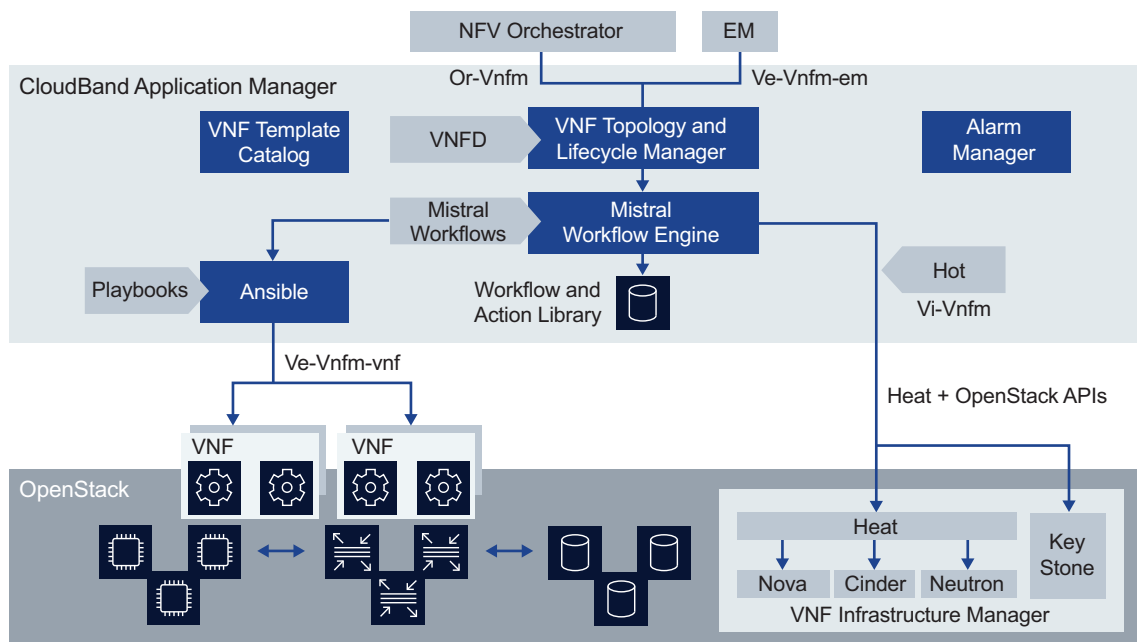
Table 14 describes the CBAM interfaces referenced in Figure 50 and used for VSR-I lifecycle management.

Table 14   **CBAM Interfaces**

Interface Reference Point	Description
Or-Vnfm	A REST-based interface for integration with orchestration. The VSR-I VNFD describes the VNF requirements for the orchestration layer and the supported operations.
Ve-Vnfm-em	A REST-based interface for integration with the VNF element management system. For VSR-I, CBAM is integrated with NSP NFM-P. For more information, see Lifecycle Management Actions Supported for VSR-I VNFs.
Ve-Vnfm-vnf	An SSH-based interface for VSR-I commissioning, used in VSR-I Ansible playbooks.
Vi-Vnfm	A REST-based interface for HEAT and Keystone OpenStack components, used to instantiate and manage the virtual resources needed by the VSR-I VNF.

8.3 Lifecycle Management Actions Supported for VSR-I VNFs

CBAM lifecycle management actions supported for VSR-I instances are summarized in Table 15. Some of the actions depend on whether NSP NFM-P is integrated with CBAM. NSP NFM-P is the element management system (EMS) of the VSR, and supports integration with CBAM over the Ve-Vnfm-Em reference point, as described and diagrammed in Introduction to CBAM. When it is integrated, NFM-P can actively monitor the status of VSR-I virtual resources and trigger certain LCM actions.

Table 15   **Supported LCM Actions for VSR-I Instances**

LCM Action	Description
On-board VNF	From CBAM GUI or REST API, on-boards the VNF package in CBAM so that it is available for later use in VNF creation
Create VNF	From CBAM GUI or REST API, creates a uniquely identified VNF in CBAM.
Modify VNF information	From CBAM GUI or REST API, modifies the VNF metadata and extensions from their defaults as defined in the VNFD.
Upgrade VNF package	From CBAM REST API, upgrades the VNF package, not the VSR-I instance. Through this operation, a new VNFD containing corrections may be on-boarded.
Instantiate VNF	From CBAM GUI or REST API, instantiates VSR-I specified OpenStack-based/VMware NFVI either in grantless or grantful mode. The necessary parameters are provided to CBAM in the form of a JSON file. After the action is complete an Ansible playbook verifies that the VSR-I is up by attempting to connect to TCP port 22 (SSH server port). If NSP is integrated, CBAM notifies NFM-P upon successful creation and NFM-P auto-discovers the new instance. **Note**: In order to be managed by NSP, the new VSR-I instance must be instantiated with config.cfg and bof.cfg files that configure a proper system IP address, enable SNMP and enable BOF persistence. Ansible playbooks perform these initial commissioning tasks.
Terminate VNF	From CBAM GUI or REST API, terminates a VSR-I instance.

Table 15 **Supported LCM Actions for VSR-I Instances (Continued)**

LCM Action	Description
Manual Heal VNFC	If NSP is integrated, manual heal request of VSR-I can be initiated by NFM-P; CBAM then implements the workflow by rebooting (first step) or rebuilding (second step) the VM and NSP is notified of the status. If NSP is not integrated, then the manual heal workflow must be initiated through the CBAM GUI or REST interface.
Auto-heal VSR-I	Requires NSP integration. If VSR-I becomes unreachable to NFM-P (according to user-defined policy rule), NSP raises an alarm and sends a heal workflow request to CBAM.

8.4   VSR-I VNF Package Design

The zip archive that can be downloaded from Nokia OLCS contains the VNF package for VSR-I (vSRCbamTemplate.csar) and an example JSON parameters file (such as vSRCbamInstantiate_Grantless.json). The VNF package has the following structure:

- `ansible/`
- `CBAM-Extensions/`
- `hot/`
- `images/`
- `javascript/`
- `Licenses/`
- `mistral-workbooks/`
- `TOSCA-Definitions/`
- `TOSCA-Metadata/`
- `TOSCA-Metadata/TOSCA.meta`
- `ChangeLog.txt`
- `vsr.vnfd.tosca.mf`
- `vsr.vnfd.tosca.yaml`

The `TOSCA.metafile` is the highest-level entry of the VNF package. It contains the following:

- `Entry-Definitions`—VNFD filename as included in the VNF package (for example, `vSR.vnfd.tosca.yaml`). The VNFD refers to in the TOSCA.meta file and refers to all other files included in the VNF package.
- `CSAR-Version`—version of the CSAR package format
- `Created-by`—vendor of the VNF package (for example, Nokia)
- `TOSCA-Meta-File-Version`—version of the VNF package (for example, 1.0, 2.0, and so on); increments with each VNF package release
- `ETSI-Entry-Manifest`—manifest filename as included in the VNF package (vsr.vnfd.tosca.mf)
- `ETSI-Entry-Change-Log`—changelog Filename as included in the VNF package (ChangeLog.txt)
- `ETSI-Entry-Licenses`—licenses folder as included in the VNF package (Licenses)
- `ETSI-Entry-Tests`—testing folder as included in the VNF package (Tests)

The VNFD referred to in the `TOSCA.meta` file refers to all other files included in the VNF package.

The VNFD components that are visible to the operator (through the CBAM GUI) are the following:

- VNF metadata and properties
- VNF requirements (external connection points)
- VNF modifiable attributes
- VNF supported lifecycle management operations (interfaces)

8.4.1   VNFD Metadata and Properties

The VNFD contains metadata and properties that uniquely identify the VNFD for CBAM. The information is duplicated in metadata and the VNF properties. The following output contains an example of these sections, from the default Nokia supplied VNFD.

```
metadata:
  nokia_vnfd_extensions: CBAM-Extensions/cbam_extensions.yaml
  nokia_vnfd_implementation_version: 4.0.0

dsl_definitions:
  descriptor_id: &descriptor_id e9e0cb86-ea33-4036-8c0d-d8ec0901529b
  provider: &provider "Nokia"
  product_name: &product_name 'VSR'
  product_info_name: &product_info_name 'Virtual Service Router'
  software_version: &software_version '20.2.1.1'
  descriptor_version: &descriptor_version '20.2.1.1.sriov'
  flavour_id: &flavour_id 'default'
  flavour_description: &flavour_description 'Default flavor of VSR'
  vnfm: &vnfm "1:CBAM-20.2"
  customize_grant_request: &customize_grant_request
    javascript: javascript/compute_affinity_rules.js
  compute_grantless_response: &compute_grantless_response
    javascript: javascript/compute_grantless_response.js
  grant_actions: &grant_actions
    compute_grantless_response: *compute_grantless_response
    customize_grant_request: *customize_grant_request
```

Note: The software_version indicates the VSR-I release for which the template was initially tested.

8.4.2 VNFD External Connection Points

The external connection points provide information about the external networks to connect to the VNF. The networks expected for a VSR-I instance are the OOB management network and the external networks attached to the m20-v MDA ports. External networks must be provided at VNF instantiation.

In the default Nokia supplied VNFD, the external connection points are modeled as follows:

```
topology_template:
  substitution_mappings:
    node_type: nokia.vSR
    requirements:
      vsrManagementECP_virtual_link: [ vsrManagementECP, virtual_link ]
      vsrExt1ECP_virtual_link: [ vsrExt1ECP, virtual_link ]
      vsrExt2ECP_virtual_link: [ vsrExt2ECP, virtual_link ]
      vsrExt3ECP_virtual_link: [ vsrExt3ECP, virtual_link ]
      vsrExt4ECP_virtual_link: [ vsrExt4ECP, virtual_link ]
      vsrExt5ECP_virtual_link: [ vsrExt5ECP, virtual_link ]
      vsrExt6ECP_virtual_link: [ vsrExt6ECP, virtual_link ]
      vsrExt7ECP_virtual_link: [ vsrExt7ECP, virtual_link ]
      vsrExt8ECP_virtual_link: [ vsrExt8ECP, virtual_link ]
```

8.4.3 VNFD Deployment Flavors

Multiple deployment flavors can be defined in the VNFD (for example, static and scalable deployments). The VSR-I VNFD currently supports a single rigid flavor as follows:

```
dsl_definitions:
  flavour_id: &flavour_id 'default'
  flavour_description: &flavour_description 'Default flavor of VSR'
node_types:
  nokia.vSR:
    derived_from: nokia.nodes.nfv.VNF
    properties:
      ….
      flavour_id:
        type: string
        constraints: [ equal: *flavour_id ]
        default: *flavour_id
      flavour_description:
        type: string
        constraints: [ equal: *flavour_description ]
        default: *flavour_description
```

8.4.4   VNFD Instantiation Level

The VNFD instantiates a VSR-I VNF with a fixed (not scalable) configuration of one
VM (VDU).

```
instantiation_levels:
      type: tosca.policies.nfv.InstantiationLevels
      properties:
        levels:
          default:
            description: VSR default instantiation level
        default_level: default
    vsrServer_instantiation_levels:
      type: tosca.policies.nfv.VduInstantiationLevels
      properties:
        levels:
          default:
            number_of_instances: 1
      targets: [ vsrServer ]
```

8.4.5   VNFD Extensions

The VNFD uses extensions to customize HOT files so that a single VNFD can be
used to instantiate multiple VNF instances. It uses the extensions to populate the VM
configuration strings (SMBIOS information) and set additional configuration and
deployment options. These parameters are set using the Modify VNF workflow prior
to VNF instantiation.

```
nokia.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      systemIpAddr:
        type: string
        default: "10.20.1.1"
      staticRoute:
        type: string
        default: "135.0.0.0/8@172.21.32.1"
      primaryConfigFile:
        type: string
        default: "ftp://*:*@172.21.36.97/./images/dut-a.cfg"
      licenseFile:
        type: string
        default: "ftp://*:*@172.21.36.97/./images/vmLICENSE/timos.vsr-all.txt"
        required: false
      vsrManagementNetmask:
        type: string
        default: "21"
      securityGroup:
        type: string
        default: "srosSecurityGroup"
      controlPlaneCores:
```

```
        type: string
        default: "4"
    snmpCommunity:
        type: string
        default: "private"
    vsrCompactFlashSize:
        type: string
        default: "64"
    vsrAllowedAddressPairList:
        type: string
        default: "0.0.0.0/0"
```

Table 16 lists the parameters of the VNFD extensions.

Table 16   **Parameters of VNFD Extensions**

Parameter	Description	Usage Notes
systemIpAddr	Specifies the system IP address of the VSR-I. Used for identification and management of the VNF from NSP	Mandatory parameter Must be a valid IPv4 or IPv6 IP address; for example: `10.10.10.10` Must be unique per VNF instance
staticRoute	Specifies a space-separated list of up to 10 static routes; for example: `static-route:0.0.0.0/ 1@1.1.1.1` `static-route:0.0.0.0/ 1@1.1.1.1`	Mandatory parameter Configuration options: • include during instantiation • configure manually after instantiation
primaryConfigFile	Specifies the FTP location of the VSR-I configuration file; for example: `ftp://user:pass@IP_address/ ~/dir/filename.cfg`	Mandatory parameter Configuration options: • include during instantiation • configure manually after instantiation
licenseFile	Specifies the FTP location of the license file; for example: `ftp://user:pass@IP_address/ ~/dir/filename.txt`	Optional parameter Configuration options: • include during instantiation • configure manually after instantiation, but this will require a reboot
vsrManagementNetmask	Specifies the OAM network subnet mask; for example, `27` for a /27 subnet	Mandatory parameter Must be a number from 1 to 32
securityGroup	Specifies the neutron security group associated with the ports created under the HOT	Mandatory parameter Format is a string that must be a valid (pre-created) OpenStack security group

Table 16   **Parameters of VNFD Extensions (Continued)**

Parameter	Description	Usage Notes
controlPlaneCores	Specifies the number of control plane CPU cores that should be allocated to the VSR-I	The template value of 4 is the minimum for most deployments
snmpCommunity	Specifies the SNMP community string used to access the VSR-I	—
vsrCompactFlashSize	The default size of the CF3 compact flash attached to the VSR	—
vsrAllowedAddressPairList	The allowed IP addresses on each interface	—
templateName	Specifies the vApp template name which is uploaded in vCD catalog	Applicable and mandatory for VMware deployments. The value can be a valid string such as "vsr.flexible"

8.4.6   VNFD Node Templates

The node templates section of the VNFD provides descriptions for all components of the VNF (networks, VMs, and storage). This section must be aligned with the flavor created in OpenStack and the resources created via HOT.

➡ **Note:** CBAM does not verify that the information described in the VNFD is aligned with OpenStack and HOT. The integrator must ensure that it is manually aligned.

In the Nokia supplied VNFD, the following characteristics apply.

- The VSR-I VM is allocated 16GB of RAM.
- The VSR-I VM is allocated 9 vCPU cores.
- The only supported virtualization environment is indicated to be KVM.
- The VSR-I is allocated one vNIC for management connectivity and eight SR-IOV vNICs for external connectivity to other VMs and network elements.

8.5 VSR-I Lifecycle Management Using CBAM

The following sections describe the workflows to on-board the VNF package for VSR-I into CBAM, create a VSR-I VNF, modify the VSR-I VNF information, instantiate a VSR-I, terminate a VSR-I, and heal a VSR-I.

8.5.1 On-board the VSR-I VNF Package

Prerequisite: none

Execution: CBAM GUI/REST

The first step before starting to manage VSR-I VNFs in CBAM is to upload the VSR-I VNF package to CBAM.

To upload the VNF package follow the steps described in the *CBAM User Guide*. After the VSR-I VNF package has been uploaded successfully to CBAM, the VNFD ID should be available in CBAM.

Before uploading the package to CBAM, potential modifications to the VNFD may be needed as described in VNFD Extensions. All VNF package files are text files that may be modified using a text editor that supports the YAML file format. The modified files must be repackaged in a zip file preserving the file structure, and ensuring that the file, directory, and tree structures are maintained, as described in VSR-I VNF Package Design.

8.5.2 Create the VNF

Prerequisite: a VSR-I VNF package must have been on-boarded in CBAM

Execution: CBAM GUI/REST

After the VNF package has been uploaded, the next step is to create the VNF. VNF creation does not instantiate the VNF in the NFVI, instead it creates the VNF in CBAM for subsequent instantiation. As part of the VNF creation, a unique VNF name must be assigned to the VNF.

For detailed VNF creation instructions, follow the steps described in the *CBAM User Guide*.

8.5.3   Modify the VNF Information

Prerequisite: a VSR-I VNF must have been created in CBAM

Execution: CBAM GUI/REST

Through the VNF modify operation, you can modify the metadata of the VNF, as specified in the VNFD, and modify the extensions from their defaults set in the VNFD.

Note: The metadata should not typically need modification prior to instantiation and may be left as specified in the VNFD.

The VSR-I extensions must be modified prior to the instantiation of each VSR-I VNF instance (see VNFD Instantiation Level). VNF extensions can be modified either through the CBAM GUI or by uploading a configuration file. An example configuration file for SR-IOV deployment is included in the zip file that also contains the VNF package.

For detailed VNF modify instructions, follow the steps described in the *CBAM User Guide*.

8.5.4   Instantiate the VNF

Prerequisites:

- a VSR-I VNF must have been created and its extension parameters set
- OpenStack prerequisites:
    - the VSR-I software image must be on-boarded into OpenStack Glance
    - OpenStack Nova flavors must be created for the VSR-I VMs
    - OpenStack Neutron networks must be created for the external networks and the OOB management network
- VMware requirements
    - external orgVDC networks must have been created
    - OVA must be on-boarded into the vCD catalog
    - the VSR-I SMBIOS configuration string must have been specified as a machine.id value

        VMware allows the SMBIOS configuration string to be passed to a VSR-I VM when it is specified it as a machine.id value.

To specify the SMBIOS configuration string as a machine.id value, perform the following steps.

    i. In the OVF descriptor file, edit the SMBIOS configuration string for the machine.id, ensuring that it starts with the string "TIMOS".

       See Sysinfo for information about a properly formatted SMBIOS string and the allowed parameters for VSR VMs.

    ii. Re-generate the OVA for CBAM use.

Execution: CBAM GUI/REST

The operation instantiates a VSR-I VNF in the OpenStack/VMware NFVI.

Table 17 describes the parameters that must be specified in the operation.

Table 17   **Operation Parameters**

Parameter	Description
VIM Information	Specifies information about the Openstack/VMware VIM that is selected for the instantiation of VSR-I including: • tenant access information (tenant ID and access credentials) • VIM contact address
VNF Flavor	Specifies the VNF flavor, as defined in the VNFD, to be used for instantiation. Currently, only a single rigid flavor is available for deployment.
Instantiation Level	Specifies an instantiation level, as defined in the VNFD The VSR-I VNFD supports a single default instantiation level that instantiates a VSR-I with a single VM.
External Networks	Specifies the external networks connected to the VNF. These are specified as Openstack/VMware network names.
NFVO Integration	The current VNFD supports both grantless and grantful mode.
VM Flavors	Reference to a pre-created OpenStack flavor to use with the VSR-I VM
Software Image Name	The software image name as on-boarded into OpenStack Glance or VMware vCD Catalog
Availability Zone	The Openstack Nova availability zone or VMware organization VDC to use with the VSR-I VM

The parameters listed in Table 17 are imported to CBAM through a parameter file in JSON format. An example parameter file for SR-IOV deployment, named "vSRCbamInstantiate.json" is included in the zip that also contains the VNF package.

For detailed VNF instantiation instructions, follow the steps described in the *CBAM User Guide*.

8.5.5   Terminate the VNF

Prerequisites: an instantiated VSR-I VNF

Execution: CBAM GUI/REST

The operation terminates a VSR VNF and deletes all resources in the NFVI. The operation may be forced or graceful.

For detailed VNF termination instructions, follow the steps described in the *CBAM User Guide*.

8.5.6   Heal the VNFC

Prerequisites:

  • an instantiated VSR-I VNF

Execution: CBAM GUI/REST/NSP NFM-P

The operation heals a VSR-I VM that cannot be recovered by guest actions. If healing is executed in OpenStack, then soft reboot is initially made on the VSR-I VM; if this action is unsuccessful, then a hard reboot is attempted. If the hard reboot is unsuccessful, then a nova rebuild action is used to attempt to recover the VM.

In VMware, the heal workflow requests NFVI to power off then power on the VM. If this action fails, then the VSR-I is re-instantiated (deleted and re-created). Card status checks are performed after heal completion in VIM level.

Healing may be triggered either through CBAM GUI/REST or from NSP NFM-P:

  • CBAM-triggered healing

   Healing is triggered manually by the operator, who must identify the VNF to be healed. There is no integration with VSR-I NFM-P to detect failure conditions that require healing.

The vnfcToHeal parameter, which corresponds to the target healed VM ID, must be configured in the CBAM GUI. Use one of the following values to configure this parameter:

– the combined slot ID (A,1)

– the VNFC name

– the internal CBAM machine ID (for example, when using OpenStack, this may be "VSR_I")

If you use OpenStack, you can execute a soft reboot, hard reboot, or nova rebuild action. If you use VMware, you can power off and power on the vCD; if this fails, then the VM is re-instantiated.

• NSP NFM-P-triggered healing

NSP NFM-P is integrated with CBAM and can automatically detect failure conditions that require healing.

9 VSR Troubleshooting

9.1 Overview

This section provides information about troubleshooting common VSR issues that relate to host or VM misconfiguration. Issues related to SR OS misconfiguration are beyond the scope of this section. Refer to the *7450 ESS and 7750 SR Troubleshooting Guide* for more information.

→ **Note:** This chapter provides a set of guidelines to use in the problem-solving process; it is not intended for use as a comprehensive set of procedures to treat VSR issues. Additional troubleshooting steps may be required to resolve the problem.

Before troubleshooting a VSR issue, ensure that there are no preexisting network problems.

Table 18 lists common VSR troubleshooting issues and potential causes.

Table 18   **Common VSR Troubleshooting Issues**

Issue	Description	Relevant sections
Excessive packet loss	A very small amount of packet loss is not unusual in NFV deployments, but if packet loss exceeds 1 to 2%, it may indicate a more serious problem	• vCPUs Not Pinned or Isolated • Insufficient CPU Resources • Incorrect Hyper-threading Settings • Incorrect NIC Driver or Firmware Versions in the Host • Incorrect MTU Settings • Insufficient Packet Buffer Memory
Lower than expected data plane performance	If the data plane performance of the VSR is much lower than expected or if it deteriorates from an established baseline, this may be the result of incorrect host or VM configuration	• vCPUs Not Pinned or Isolated • Insufficient CPU Resources • Incorrect Hyper-threading Settings • Incorrect NIC Driver or Firmware Versions in the Host • NUMA Misalignment • Insufficient Packet Buffer Memory

Table 18   **Common VSR Troubleshooting Issues (Continued)**

Issue	Description	Relevant sections
Control plane sessions drop unexpectedly Control plane protocols converge slowly	If the control plane performance of the VSR is not operating robustly, with expected convergence performance, this may indicate a CPU or memory resource allocation issue	• Insufficient CPU Resources • Insufficient VM Memory
System instability	If the VSR regularly crashes or becomes unresponsive, this could be due to an unsupported configuration of the host or VM	• vCPUs Not Pinned or Isolated • Insufficient CPU Resources • Unsupported host OS (see Compute Server Software Requirements for host OS requirements) • Insufficient VM Memory • Incorrect BIOS Settings (see BIOS Settings for configuration recommendations)

To properly troubleshoot one or more of the above problems, the following recommended workflow is:

Step 1. Collect information about the problem, as described in Collecting Linux KVM Host Information. If using VMware, refer to VMware documentation for the most up-to-date commands.

Collect statistics and configuration information from the host, the hypervisor, and the SR OS software.

Step 2. Perform one of the following.

a. If handing off troubleshooting to another party, submit the collected information. If handing off troubleshooting to a Nokia representative, also generate and submit an admin tech-support file. End the workflow here.

b. If performing troubleshooting, continue to step 3.

Step 3. Look for configuration errors or statistics that may indicate the nature of the problem, as described in Troubleshooting Common Problems.

Step 4. Attempt to resolve the issue by changing the host, hypervisor, or VSR configuration and re-testing. Changing one parameter at a time is advised so that the effect of each change can be assessed individually.

Note: Root access is assumed in the following command examples.

In some of the following output examples, comments are denoted by two forward slashes ("//").

9.2   Collecting Linux KVM Host Information

This section describes how and when to collect Linux KVM host information for troubleshooting purposes. See Troubleshooting Common Problems for troubleshooting tools and procedures.

9.2.1   Collecting Information at Host Bootup

9.2.1.1   BIOS Settings of the Host Machine

To determine the BIOS settings of the host machine, perform the following steps.

Step 1.   During bootup when performing a server reboot, press the key specified in POST (such as F2 key or DEL key, as applicable to the BIOS) to show the BIOS setup screen.

Step 2.   Navigate the BIOS screens and record the setting for each of the parameters listed in Table 19.

On a Nokia Airframe server, the American Megatrends BIOS is used. Server manufacturers may use different BIOS vendors.

→   **Note:** Setting names and navigation paths vary depending on the BIOS vendor. Some vendors may require enabling or disabling related parameters. See BIOS Settings for more information about recommended settings.

Table 19   **BIOS Settings**

Parameter	Setting	American Megatrends Navigation Path
SR-IOV Support	Enabled or Disabled	Advanced tab → PCI Subsystem Settings
Hyper-Threading	Enabled or Disabled	IntelRCSetup tab → Processor Configuration
VMX	Enabled or Disabled	IntelRCSetup tab → Processor Configuration
Hardware Prefetcher	Enabled or Disabled	IntelRCSetup tab → Processor Configuration
Adjacent Cache Prefetch	Enabled or Disabled	IntelRCSetup tab → Processor Configuration
X2APIC	Enabled or Disabled	IntelRCSetup tab → Processor Configuration

Table 19   **BIOS Settings (Continued)**

Parameter	Setting	American Megatrends Navigation Path
Intel® VT for Directed I/O (VT-d)	Enabled or Disabled	IntelRCSetup tab → IIO Configuration
CPU C3 report	Enabled or Disabled	IntelRCSetup tab → Advanced Power Management Configuration → CPU C State Control
CPU C6 report	Enabled or Disabled	IntelRCSetup tab → Advanced Power Management Configuration → CPU C State Control
Turbo Mode	Enabled or Disabled	IntelRCSetup tab → Advanced Power Management Configuration → CPU P State Control
NUMA	Enabled or Disabled	IntelRCSetup tab → Common RefCode Configuration
CPU processor	CPU processor model in each socket	IntelRCSetup tab → Processor Configuration
Power management	Energy Performance BIAS Setting	IntelRCSetup tab → Advanced Power Management Configuration→ CPU – Advanced PM Tuning→ Energy Performance BIAS

9.2.2   Collecting Information Before Any VSR VMs Are Running

9.2.2.1   Used and Available Huge Pages (VMs Not Running)

Use the **cat /proc/meminfo** command to show the following statistics:

- HugePages_Total — the total number of huge pages
- HugePages_Free — the number of huge pages that are free

For example:

```
[root@vsr ~]# cat /proc/meminfo
MemTotal:       65759060 kB
MemFree:        11688396 kB
MemAvailable:   11749136 kB
Buffers:           85636 kB
Cached:           205368 kB
SwapCached:            0 kB
Active:           203668 kB
Inactive:         182536 kB
Active(anon):     108172 kB
```

```
Inactive(anon):     9276 kB
Active(file):      95496 kB
Inactive(file):   173260 kB
Unevictable:       41208 kB
Mlocked:           41208 kB
SwapTotal:       32965628 kB
SwapFree:        32965628 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        136424 kB
Mapped:            49224 kB
Shmem:             10204 kB
Slab:             108880 kB
SReclaimable:      77308 kB
SUnreclaim:        31572 kB
KernelStack:        3568 kB
PageTables:         6304 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     39630756 kB
Committed_AS:     752012 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      484788 kB
VmallocChunk:   34358945788 kB
HardwareCorrupted:     0 kB
AnonHugePages:     51200 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:      50
HugePages_Free:       34
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
DirectMap4k:      179620 kB
DirectMap2M:     4947968 kB
DirectMap1G:    63963136 kB
[root@vsr ~]#
```

9.2.3   Collecting Information When the Host OS Is Running, Whether or Not VSR VMs Are Running

9.2.3.1   Linux OS Distribution and Version

Use the **cat /etc/*release*** and **uname --all** commands to check the CentOS, RedHat, Ubuntu, or Linux version.

The following **cat /etc/*release*** output shows the host OS as "CentOS 7.3.1611".

```
[root@vsr ~]# cat /etc/*release*
```

```
CentOS Linux release 7.3.1611 (Core) //This line shows the host OS.
Derived from Red Hat Enterprise Linux 7.3 (Source)
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
CentOS Linux release 7.3.1611 (Core) //This is an incorrect version.
CentOS Linux release 7.3.1611 (Core) //This is an incorrect version.
cpe:/o:centos:centos:7
```

The following **cat /etc/*release*** output shows the host OS as "Ubuntu 16.04.1".

```
root@vsr:~# cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.4.1 LTS"
NAME="Ubuntu"
ID=ubVERSION="16.04.1 LTS (Xenial Xerus)" //This is an incorrect version.
untu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
```

9.2.3.2   Linux Kernel Version

Use the **uname --all** command to determine the kernel name, version, release level, and release date.

The following **uname --all** output shows the kernel version as "3.10.0-514".

```
[root@vsr ~]# uname --all
Linux vsr 3.10.0-
514.6.1.el7.x86_64 #1 SMP Wed Jan 18 13:06:36 UTC 2017 x86_64 x86_64 x86_64 GNU/
Linux
```

The following **uname --all** output, from an Ubuntu machine, shows the kernel version as "4.4.0-53-generic".

```
root@sc-03:~# uname --all
Linux sc-03 4.4.0-53-generic #74-
Ubuntu SMP Fri Dec 2 15:59:10 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
```

9.2.3.3   PCI Slots in the Host Machine

PCI slots have different bandwidth capabilities, expressed, for example, as "PCI Express x1" or "PCI Express x4/x8/x16", where the number after "x" represents a number of PCI lanes. The higher the value after "x", the more PCI lanes are available per slot, resulting in more bandwidth. The physical slot size is also different for each value.

→ **Note:** All slots are backward compatible. While a large card can be inserted into a small slot and can function, the speed is reduced. Consider the NIC requirements for a specified PCI slot to achieve the maximum speed.

Use the Linux **dmidecode -t slot** command to show information about every PCI slot in the host machine. For each PCI slot, the following information is provided:

- server slot and CPU number (CPU1, CPU2)
- PCI bus width (x8, x16)
- PCI bus address (domain:bus:slot:function)

The following **dmidecode-t** output shows that the host machine has a PCI SLOT1 on the motherboard, connected to CPU1. It is a PCIe version 3 bus with x8 width and therefore has a maximum theoretical bandwidth of 63 Gb/s. The PCI slot has the bus address 0000:02:00.0.

```
[root@vsr vsr-ws]# dmidecode -t slot
Handle 0x001C, DMI type 9, 17 bytes
System Slot Information
        Designation: CPU1 SLOT1
        Type: x8 PCI Express 3 x8
        Current Usage: Available
        Length: Short
        ID: 1
        Characteristics:
                3.3 V is provided
                Opening is shared
                PME signal is supported
        Bus Address: 0000:02:00.0
Handle 0x001D, DMI type 9, 17 bytes
System Slot Information
        Designation: CPU1 SLOT2
        Type: x16 PCI Express 3 x16
        Current Usage: In Use
        Length: Long
```

```
                    ID: 2
                    Characteristics:
                            3.3 V is provided
                            Opening is shared
                            PME signal is supported
                    Bus Address: 0000:03:00.0
...
Handle 0x0020, DMI type 9, 17 bytes
System Slot Information
        Designation: CPU2 SLOT5
        Type: x8 PCI Express 3 x8
        Current Usage: In Use
        Length: Short
        ID: 5
        Characteristics:
                3.3 V is provided
                Opening is shared
                PME signal is supported
        Bus Address: 0000:81:00.0
```

9.2.3.4   CPU Mapping to NUMA Nodes

Use the **numactl -H** command to record the memory size and CPU processor
mapping to each NUMA node.

In the following output, lines that start with "node 0" refer to NUMA 0, and those that
start with "node 1" refer to NUMA 1.

```
[root@vsr vsr-ws]# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
node 0 size: 65424 MB
node 0 free: 38045 MB
node 1 cpus: 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
node 1 size: 65536 MB
node 1 free: 38958 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

9.2.3.5   NIC Driver and Firmware Details

9.2.3.5.1   Show the List of PCIe Devices

Use the **lshw** command to show the list of PCIe devices. Determine if there are any missing device names or descriptions for any installed NIC, which may suggest a problem with driver activation or that an alternate driver is in use.

The **lshw** command is not present by default, so must be installed using the **yum install** or **apt install** commands.

```
[root@vsr vsr-ws]# yum install lshw
root@sc-03:~# apt install lshw
```

The following **lshw** output does not have missing device names or descriptions.

```
[root@vsr ~]# lshw -c network -businfo
Bus info         Device      Class     Description
=========================================================
pci@0000:03:00.0  enp3s0f0    network  82599 10 Gigabit Dual Port Network Connection
pci@0000:03:00.1  enp3s0f1    network  82599 10 Gigabit Dual Port Network Connection
pci@0000:06:00.0  enp6s0f0    network  Ethernet Controller X540-AT2
pci@0000:06:00.1  enp6s0f1    network  Ethernet Controller X540-AT2
pci@0000:81:00.0  ens6f0      network  82599 10 Gigabit Dual Port Network Connection
pci@0000:81:00.1  ens6f1      network  82599 10 Gigabit Dual Port Network Connection
pci@0000:83:00.0  enp131s0f0  network  82599 10 Gigabit Dual Port Network Connection
pci@0000:83:00.1  enp131s0f1  network  82599 10 Gigabit Dual Port Network Connection
pci@0000:85:00.0  ens4f0      network  MT27700 Family [ConnectX-4]
pci@0000:85:00.1  ens4f1      network  MT27700 Family [ConnectX-4]
<>
```

The following **lshw** output shows bridges.

```
[root@vsr ~]# lshw -c network -businfo
Bus info         Device        Class     Description
=========================================================
pci@0000:03:00.0  enp3s0f0      network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:03:00.1  enp3s0f1      network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:05:00.0  enp5s0f0      network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:05:00.1  enp5s0f1      network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:07:00.0  enp7s0f0      network   Ethernet Controller 10-Gigabit X5
pci@0000:07:00.1  enp7s0f1      network   Ethernet Controller 10-Gigabit X5
pci@0000:81:00.0  enp129s0f0    network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:81:00.1                network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:81:10.0                network   82599 Ethernet Controller Virtual
pci@0000:83:00.0  enp131s0f0    network   82599ES 10-Gigabit SFI/SFP+ Netwo
pci@0000:83:00.1  enp131s0f1    network   82599ES 10-Gigabit SFI/SFP+ Netwo
                  enp5s0f0.10   network   Ethernet interface
                  ovs-mgmt      network   Ethernet interface
                  ovs-vsr1-fab  network   Ethernet interface
                  ovs-system    network   Ethernet interface
                  virbr0-nic    network   Ethernet interface
```

```
                        virbr0        network        Ethernet interface
                        vsr1-mgmt     network        Ethernet interface
```

The following **lshw** output shows two missing device names. In this scenario, the VFIO-PCI driver is in use for DPDK purposes.

```
[root@vsr vsr-ws]#  lshw -c network -businfo
Bus info          Device       Class    Description
=========================================================
pci@0000:03:00.0  enp3s0f0     network 82599 10 Gigabit Dual Port Network Connection
pci@0000:03:00.1  enp3s0f1     network 82599 10 Gigabit Dual Port Network Connection
pci@0000:05:00.0               network 82599 10 Gigabit Dual Port Network Connection
pci@0000:05:00.1               network 82599 10 Gigabit Dual Port Network Connection
pci@0000:07:00.0  enp7s0f0     network Ethernet Controller X540-AT2
pci@0000:07:00.1  enp7s0f1     network Ethernet Controller X540-AT2
pci@0000:81:00.0  ens5f0       network 82599 10 Gigabit Dual Port Network Connection
pci@0000:81:00.1  ens5f1       network 82599 10 Gigabit Dual Port Network Connection
<>
```

The following **lshw** output is missing the device name and description in several fields, where the description is "Illegal Vendor ID". In this scenario, the VFIO-PCI driver is in use for SR IOV purposes.

```
[root@vsr vsr-ws]# lshw -c network -businfo
Bus info          Device       Class        Description
=========================================================
pci@0000:03:00.0  enp3s0f0     network      82599 10 Gigabit Dual Port Network Conn
ection
pci@0000:03:00.1  enp3s0f1     network      82599 10 Gigabit Dual Port Network Conn
ection
pci@0000:05:00.0  ens3f0       network      82599 10 Gigabit Dual Port Network Conn
ection
pci@0000:05:00.1  ens3f1       network      82599 10 Gigabit Dual Port Network Conn
ection
pci@0000:05:10.0               network      Illegal Vendor ID
pci@0000:05:10.1               network      Illegal Vendor ID
pci@0000:05:10.2               network      Illegal Vendor ID
pci@0000:05:10.3               network      Illegal Vendor ID
pci@0000:05:10.4               network      Illegal Vendor ID
pci@0000:05:10.5               network      Illegal Vendor ID
pci@0000:07:00.0  enp7s0f0     network      Ethernet Controller X540-AT2
pci@0000:07:00.1  enp7s0f1     network      Ethernet Controller X540-AT2
```

9.2.3.5.2   Show the Kernel Drivers and Modules

Use the **lspci** command to show the kernel driver and kernel module used for every installed NIC.

The following **lspci** output shows information for two Intel X520 NICs.

Note: For the second NIC, the kernel driver in use (vfio-pci) is different from the kernel module (ixgbe), which indicates a limited usage of this interface (DPDK or PCI passthrough).

```
[root@vsr vsr-ws]# lspci  -k | grep -i ethernet -A2
03:00.0 Ethernet controller: Intel(R) 82599 10 Gigabit Dual Port Network Connection
(rev 01)
        Subsystem: Intel(R) Ethernet Server Adapter X520-2
        Kernel driver in use: ixgbe
        Kernel modules: ixgbe
...
05:00.0 Ethernet controller: Intel(R) 82599 10 Gigabit Dual Port Network Connection
(rev 01)
        Subsystem: Intel(R) Ethernet Server Adapter X520-2
        Kernel driver in use: vfio-pci
        Kernel modules: ixgbe
```

The following **lspci** output corresponds to two ports of a Mellanox Connect X-4 NIC.

```
[root@vsr ~]# lspci  -k | grep -i ethernet -A2
85:00.0 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
        Subsystem: Mellanox Technologies Device 0050
        Kernel driver in use: mlx5_core
        Kernel modules: mlx5_core
85:00.1 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
        Subsystem: Mellanox Technologies Device 0050
        Kernel driver in use: mlx5_core
        Kernel modules: mlx5_core
```

9.2.3.5.3   Show the Driver Version, Name, and PCI Bus Information

Note: Whenever a physical NIC is moved from one slot to another slot, collect the following information. A NIC swap may result in Linux interface name changes, MAC address changes, and/or PCI bus address changes.

For each kernel module used by an installed NIC, use the **modinfo** command to show the version of the installed driver.

The following **modinfo** output shows that the installed Intel ixgbe driver is version 4.5.4.

```
[root@vsr vsr-ws]# modinfo ixgbe
filename:       /lib/modules/3.10.0-514.6.1.el7.x86_64/updates/drivers/net/ethernet/
intel/ixgbe/ixgbe.ko
version:        4.5.4
license:        GPL
description:    Intel(R) 10GbE PCI Express Linux Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
```

```
rhelversion:      7.3
srcversion:       57F7B0031F462E8B1D7D84C
```

The following **modinfo** output shows that the installed Mellanox mlx5_core driver is version 3.4-1.0.6.

```
[root@vsr ~]# modinfo mlx5_core
filename:         /lib/modules/3.10.0-514.2.2.el7.x86_64/weak-updates/mlnx-en/drivers/
net/ethernet/mellanox/mlx5/core/mlx5_core.ko
version:          3.4-1.0.6
license:          Dual BSD/GPL
description:      Mellanox Connect-IB, ConnectX-4 core driver
author:           Eli Cohen <eli@mellanox.com>
rhelversion:      7.3
srcversion:       8ACFABDB5FCF79412802363
```

Alternatively, use the **ethtool -i** *interface name* command to show the driver name, driver version, and PCI bus information for a Linux interface.

The following **ethtool -i** output is for an Intel NIC port.

```
[root@vsr vsr-ws]# ethtool -i ens5f0
driver: ixgbe
version: 4.5.4
firmware-version: 0x61c10001
expansion-rom-version:
bus-info: 0000:81:00.0
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: yes
```

The following **ethtool -i** output is for a Mellanox NIC port.

```
[root@vsr ~]# ethtool -i ens4f0
driver: mlx5_core
version: 3.4-1.0.6 (20 Nov 2016)
firmware-version: 12.16.1020
expansion-rom-version:
bus-info: 0000:85:00.0
supports-statistics: yes
supports-test: yes
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: yes
```

9.2.3.6   Host Interface Details

Use the **ip link show** command to show information about host interfaces. Verify that the MTU value of each interface is sufficiently higher than the guest interface MTU settings.

```
[root@vsr_hyp]# ip link show
11: enp132s0f1: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc mq master
br02 state UP mode DEFAULT qlen 1000
15: br02: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEF
AULT
87: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br02 s
tate UNKNOWN mode DEFAULT qlen 500
89: vnet3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br02 s
tate UNKNOWN mode DEFAULT qlen 500
[root@vsr_hyp]#
```

9.2.3.7   Optical Transceiver Details

Use the **ethtool -m** *interface name* command to show details about the pluggable optical transceiver module associated with a NIC port.

The following **ethtool -m** output shows SFP statistics when using Intel.

```
[root@vsr vsr-ws]# ethtool -m ens5f0
        Identifier                           : 0x03 (SFP)
        Extended identifier                  : 0x04 (GBIC/SFP defined by 2-
wire interface ID)
        Connector                            : 0x07 (LC)
        Transceiver codes                    : 0x10 0x00 0x00 0x01 0x00 0x00 0x
00 0x00
        Transceiver type                     : 10G Ethernet: 10G Base-SR
        Transceiver type                     : Ethernet: 1000BASE-SX
        Encoding                             : 0x06 (64B/66B)
        BR, Nominal                          : 10300MBd
        Rate identifier                      : 0x02 (8/4/
2G Rx Rate_Select only)
        Length (SMF,km)                      : 0km
        Length (SMF)                         : 0m
        Length (50um)                        : 80m
        Length (62.5um)                      : 30m
        Length (Copper)                      : 0m
        Length (OM3)                         : 300m
        Laser wavelength                     : 850nm
        Vendor name                          : Intel Corp
        Vendor OUI                           : 00:1b:21
        Vendor PN                            : FTLX8571D3BCV-IT
        Vendor rev                           : A
        Option values                        : 0x00 0x3a
        Option                               : RX_LOS implemented
        Option                               : TX_FAULT implemented
        Option                               : TX_DISABLE implemented
```

```
        Option                                  : RATE_SELECT implemented
        BR margin, max                          : 0%
        BR margin, min                          : 0%
        Vendor SN                               : MUF1ZZL
        Date code                               : 151111
        Optical diagnostics support             : Yes
        Laser bias current                      : 7.690 mA
        Laser output power                      : 0.6140 mW / -2.12 dBm
        Receiver signal average optical power   : 0.6015 mW / -2.21 dBm
        Module temperature                      : 31.86 degrees C /
 89.34 degrees F
        Module voltage                          : 3.3232 V
```

The following **ethtool -m** output shows SFP statistics when using Mellanox.

```
[root@vsr ~]# ethtool -m ens4f0
        Identifier                              : 0x11 (QSFP28)
        Extended identifier                     : 0x00
        Extended identifier description         : 1.5W max. Power consumption
        Extended identifier description         : No CDR in TX, No CDR in RX
        Extended identifier description         : High Power Class (> 3.5 W) not e
nabled
        Connector                               : 0x23 (No separable connector)
        Transceiver codes                       : 0x88 0x00 0x00 0x00 0x00 0x00 0x
00 0x00
        Transceiver type                        : 40G Ethernet: 40G Base-CR4
        Transceiver type                        : 100G Ethernet: 100G Base-
CR4 or 25G Base-CR CA-L
        Encoding                                : 0x00 (unspecified)
        BR, Nominal                             : 25500Mbps
        Rate identifier                         : 0x00
        Length (SMF,km)                         : 0km
        Length (OM3 50um)                       : 0m
        Length (OM2 50um)                       : 0m
        Length (OM1 62.5um)                     : 0m
        Length (Copper or Active cable)         : 3m
        Transmitter technology                  : 0xa0 (Copper cable unequalized)
        Attenuation at 2.5GHz                   : 6db
        Attenuation at 5.0GHz                   : 8db
        Attenuation at 7.0GHz                   : 10db
        Attenuation at 12.9GHz                  : 16db
        Vendor name                             : Mellanox
        Vendor OUI                              : 00:02:c9
        Vendor PN                               : MCP1600-C003
        Vendor rev                              : A2
        Vendor SN                               : MT1622VS07875
        Revision Compliance                     : SFF-8636 Rev 2.0
        Module temperature                      : 0.00 degrees C / 32.00 degrees F
        Module voltage                          : 0.0000 V
```

9.2.4 Collecting Information After VSR VMs Are Running

9.2.4.1 NUMA Information

NUMA allocation has a strong impact on high-performance VNFs. NUMA misalignment can result in VSR instability and high traffic latency.

A single NUMA node represents a pool of CPU cores, memory, and PCI slots (containing NICs); these virtualized components perform best when the latency in communication between components is minimized. Allocating components (such as CPU cores and NICs) from different NUMA nodes requires communication to cross a QPU/UPI link, which is limited in bandwidth and can cause delays. These resources should reside on the same NUMA. Border violations may occur if this requirement is not observed.

Use the **numactl** command to determine if there is an issue with NUMA misalignment (see NUMA Topology for more information about this command).

Use the **numastat** command to detect NUMA border violations.

9.2.4.1.1 Used and Available Memory Per NUMA Node

Use the **numastat -m** command to show the following statistics per NUMA node:

- MemTotal — the total memory available
- MemFree — free memory
- MemUsed — used memory

The following **numastat -m** output corresponds to a server with 128 Gbytes DRAM. Each NUMA has 64 Gbytes of memory and the utilization of each NUMA memory pool is about equal.

```
[root@vsr vsr-ws]# numastat -m
Per-node system memory usage (in MBs):
                        Node 0          Node 1           Total
               --------------- --------------- ---------------
MemTotal             65424.62        65536.00       130960.62
MemFree              38044.87        38959.46        77004.33
MemUsed              27379.75        26576.54        53956.29
Active                 169.55          149.30          318.86
Inactive                63.72           57.29          121.01
Active(anon)           136.54           27.46          164.00
Inactive(anon)           1.18            9.05           10.23
Active(file)            33.01          121.84          154.85
Inactive(file)          62.55           48.23          110.78
```

```
Unevictable            111.11         306.29         417.40
Mlocked                111.11         306.29         417.40
Dirty                    0.00           0.00           0.00
Writeback                0.00           0.00           0.00
FilePages              107.59         181.03         288.61
Mapped                  38.04          11.28          49.32
AnonPages              236.96         331.93         568.89
Shmem                    1.08           9.31          10.39
KernelStack              4.02           4.31           8.33
PageTables               4.27           3.36           7.63
NFS_Unstable             0.00           0.00           0.00
Bounce                   0.00           0.00           0.00
WritebackTmp             0.00           0.00           0.00
Slab                    65.73          57.61         123.34
SReclaimable            24.79          21.64          46.43
SUnreclaim              40.95          35.96          76.91
AnonHugePages          176.00         244.00         420.00
HugePages_Total      24576.00       24576.00       49152.00
HugePages_Free        4096.00       20480.00       24576.00
HugePages_Surp           0.00           0.00           0.00
```

9.2.4.1.2   NUMA Memory Used by the Hypervisor

Use the **numastat qemu** command to show information about the use of memory by
the QEMU process.

The following **numastat qemu** output shows that all VMs created by QEMU are
allocated huge pages from NUMA node 0.

```
[root@vsr vsr-ws]# numastat qemu
Per-node process memory usage (in MBs) for PID 6062 (qemu-kvm)
                           Node 0          Node 1            Total
                  --------------- --------------- ---------------
Huge                      16384.00            0.00        16384.00
Heap                         99.93            0.00           99.93
Stack                         0.11            0.00            0.11
Private                      16.96            0.15           17.11
--------------- --------------- --------------- ---------------
Total                     16501.00            0.15        16501.15
```

9.2.4.1.3   NUMA Miss Statistics

Use the **numastat** or **numastat -v** (verbose) command to show the following NUMA
statistics:

- Numa Miss — the number of attempted memory allocations to another node that
  were allocated on this node due to low memory on the intended node. Each
  Numa Miss event has a corresponding Numa Foreign event on another node.

• Numa Foreign — the number of memory allocations initially intended for this node that were allocated to another node instead. Each Numa Foreign event has a corresponding Numa Miss event on another node.

The following **numastat** output shows memory usage within NUMA borders.

```
[root@vsr vsr-ws]# numastat -v
Per-node numastat info (in MBs):
                          Node 0          Node 1           Total
                   --------------  --------------  --------------
Numa_Hit                  9727.01        14590.22        24317.23
Numa_Miss                    0.00            0.00            0.00
Numa_Foreign                 0.00            0.00            0.00
Interleave_Hit             128.71          127.19          255.90
Local_Node                5169.49        14452.00        19621.49
Other_Node                4557.52          138.21         4695.73
```

The following **numastat** output shows memory usage that violates NUMA borders:

```
[root@localhost ~]# numastat -v
Per-node numastat info (in MBs):
                          Node 0          Node 1           Total
                   --------------  --------------  --------------
Numa_Hit                   445179         1308573         1753752
Numa_Miss                       0          971897         1971897
Numa_Foreign               971897               0          971897
Interleave_Hit                 80              79             159
Local_Node                 445169         1308506         1753676
Other_Node                     10          971964          971973
```

9.2.4.2 Used and Available Huge Pages (VMs Running)

Use the **cat /proc/meminfo** command to show the following statistics and compare to the results from the same command as used before the VMs were running:

• HugePages_Total — the total number of huge pages
• HugePages_Free — the number of huge pages that are free, after deducting use by the VSR VMs

See output in Used and Available Huge Pages (VMs Not Running).

9.2.4.3 Kernel Messages

Use the **dmesg** command to view kernel messages that may indicate errors.

The following **dmesg** output shows an MTU error, indicated by the "VF max_frame 9216 out of range" line.

```
[root@vsr_hyp ~]# dmesg | tail -10
[1670173.622405] vfio-pci 0000:03:10.0: irq 65 for MSI/MSI-X
[1670173.682574] ixgbe 0000:03:00.1 enp3s0f1: VF Reset msg received from vf 0
[1670173.692442] vfio-pci 0000:03:10.1: irq 61 for MSI/MSI-X
[1670173.692451] vfio-pci 0000:03:10.1: irq 66 for MSI/MSI-X
[1670173.832478] ixgbe 0000:82:00.0 enp130s0f0: VF Reset msg received from vf 0
[1670173.852540] ixgbe 0000:82:00.0 enp130s0f0: VF max_frame 9216 out of range
[1670173.930838] ixgbe 0000:82:00.1 enp130s0f1: VF Reset msg received from vf 0
[1670174.051260] ixgbe 0000:84:00.1 enp132s0f1: VF Reset msg received from vf 0
[1670174.151628] ixgbe 0000:03:00.0 enp3s0f0: VF Reset msg received from vf 0
[1670174.251940] ixgbe 0000:03:00.1 enp3s0f1: VF Reset msg received from vf 0
```

9.2.4.4 MTU Information

Use the **show port** command to view the interface MTU from the VSR perspective.

Use the **ip link show** *device* command to view the interface MTU from the host perspective.

9.2.5 Collecting Information When the VSR Is Running and Under Load

When determining whether a VM is overloaded, consider that the CPU load is divided into the following blocks, which do not directly depend on the other:

- control plane CPU
- data plane CPU

9.2.5.1 VSR Control Plane CPU Utilization

Use the **show system cpu** or **show card a cpu** commands to show the CPU utilization of the VSR control plane.

The following output shows an idle system; output values will be higher for a non-idle system.

```
A:vsr23-1# show card "a" cpu
===============================================================================
Card a CPU Utilization (Sample period: 1 second)
===============================================================================
```

Name	CPU Time (uSec)	CPU Usage	Capacity Usage
BFD	7	~0.00%	~0.00%
BGP	0	0.00%	0.00%
BGP PE-CE	0	0.00%	0.00%
CFLOWD	15	~0.00%	~0.00%
Cards & Ports	1,131	0.05%	0.04%
DHCP Server	4	~0.00%	~0.00%
ETH-CFM	175	~0.00%	0.01%
HQoS Algorithm	39	~0.00%	~0.00%
HQoS Statistics	0	0.00%	0.00%
ICC	148	~0.00%	~0.00%
IMSI Db Appl	12	~0.00%	~0.00%
IOM	6,177	0.30%	0.22%
IP Stack	853	0.04%	0.02%
IS-IS	205	0.01%	0.01%
ISA	561	0.02%	0.02%
LDP	241	0.01%	0.02%
Logging	5	~0.00%	~0.00%
MBUF	0	0.00%	0.00%
MCS	72	~0.00%	0.02%
MPLS/RSVP	600	0.02%	0.03%
MSCP	0	0.00%	0.00%
Management	1,480	0.07%	0.04%
OAM	928	0.04%	0.03%
OSPF	894	0.04%	0.02%
OpenFlow	8	~0.00%	~0.00%
PKI	20	~0.00%	~0.00%
RIP	0	0.00%	0.00%
RTM/Policies	0	0.00%	0.00%
Redundancy	0	0.00%	0.00%
SIM	246	0.01%	0.01%
SNMP Daemon	0	0.00%	0.00%
Security	0	0.00%	0.00%
Services	570	0.02%	0.02%
Stats	0	0.00%	0.00%
Subscriber Mgmt	431	0.02%	0.02%
System	5,903	0.29%	0.18%
Traffic Eng	0	0.00%	0.00%
VRRP	131	~0.00%	~0.00%
WEB Redirect	21	~0.00%	~0.00%
Total	2,003,978	100.00%	
Idle	1,983,062	98.95%	
Usage	20,916	1.04%	
Busiest Core Utilization	11,071	1.10%	

9.2.5.2 VSR Data Plane CPU Utilization

Use the **show card 1 virtual fp** command to show the utilization of the VSR data plane compared to its maximum capacity.

> **Note:** These cores work in PMD mode and are shown in hypervisor as 100% load; the **show card virtual fp** command shows the real effective load.

```
*A:Latency-AA# show card 1 virtual fp
===============================================================================
Card 1 Virtual Forwarding Plane Statistics
===============================================================================
Task                    vCPUs      Average        Maximum
                                   Utilization    Utilization
-------------------------------------------------------------------------------
NIC                     1          39.28 %        39.28 %
Worker                  4          0.05 %         0.05 %
Scheduler               1          0.00 %         0.00 %
```

If the worker tasks are under heavy utilization, as shown in Figure 51, this will create back pressure on the NIC task, which will stop draining packets from the NIC. The following diagram shows this process.

Figure 51   **Worker Tasks Under Heavy Utilization**

sw0571
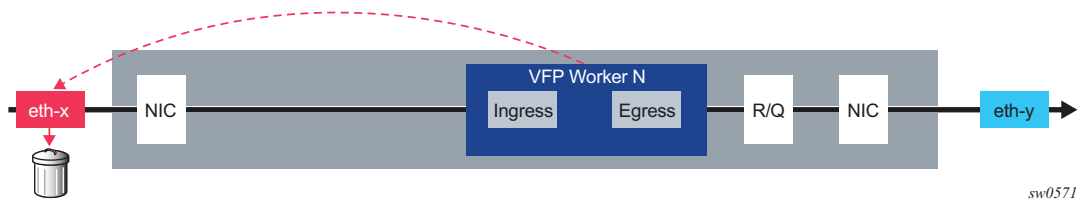
9.2.5.3   Host Machine CPU Utilization (HTOP)

Use the **htop** command to show the utilization of each physical CPU core from the host OS perspective.

The **htop** command is not present by default, so must be installed using the **yum install** or **apt install** commands.

```
[root@vsr vsr-ws]# yum install htop
root@sc-03:~# apt install htop
```

An output sample for **htop** is shown in Figure 52.

Figure 52   **htop Output**
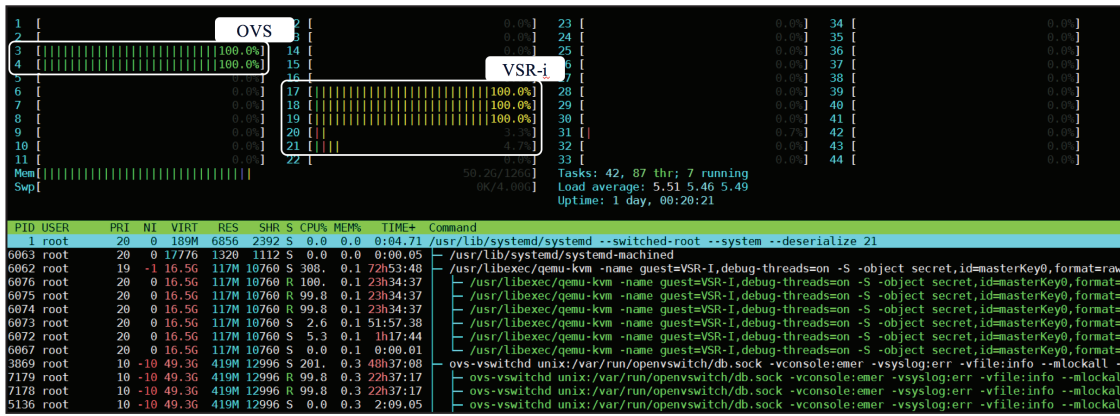
sc0073

9.2.5.4   NIC Packet Drops

9.2.5.4.1   Show the Number of Dropped Packets

Use the **show port 1/***mda***/***port* **detail** command and record the number of "Dropped" packets under the heading "NIC receive overrun".

> **Note:** To show this information, the system must be configured with **config**>**system**>**congestion-management**.

This information is especially important when the port is in PCI passthrough mode because the host does not show any drop statistics for PCI passthrough ports when using the **ethtool -S** command, as described in Show Host-Level NIC Statistics.

> **Note:** Although the port may be shown as up and running, traffic may be silently dropped by the host.

```
A:VSR# show port 1/1/1 detail
===============================================================================
VSR Early Discard Statistics
===============================================================================
                                  Packets                   Octets
-------------------------------------------------------------------------------
NIC receive overrun
        Dropped :                 0                         0
```

```
Ingress capacity exceeded early discards
    FC 1 Dropped :              0                        0
    FC 2 Dropped :              0                        0
    FC 3 Dropped :              0                        0
    FC 4 Dropped :              0                        0
    FC 5 Dropped :              0                        0
    FC 6 Dropped :              0                        0
    FC 7 Dropped :              0                        0
Buffer pool exhaustion early discards
    FC 1 Dropped :              416967890                211819636753
    FC 2 Dropped :              0                        0
    FC 3 Dropped :              161313510                81947263080
    FC 4 Dropped :              0                        0
    FC 5 Dropped :              0                        0
    FC 6 Dropped :              0                        0
    FC 7 Dropped :              0                        0
```

9.2.5.4.2   Show Host-Level NIC Statistics

Use the **ethtool -S** *interface-name* command to show host-level NIC statistics and to check for dropped traffic. The most important statistic in this output is the count for rx_no_dma_resources. Any number greater than 0, or that increments with traffic, indicates that host-level resources are being exhausted.

Figure 53 shows NIC statistics at a 90% load.

Figure 53   **NIC Statistics – 90% Load (ixgbe 4.6.4)**

```
NIC statistics:
     tx_packets: 37
     tx_bytes: 6318
     rx_missed_errors: 141149694
     rx_pkts_nic: 1395946578
     tx_pkts_nic: 969018413
     rx_bytes_nic: 98374173120
     tx_bytes_nic: 62017173588
     lsc_int: 6
     tx_flow_control_xoff: 56030
     rx_no_dma_resources: 426775772
     tx_queue_0_packets: 13
     tx_queue_0_bytes: 4446
     tx_queue_1_packets: 6
     tx_queue_1_bytes: 540
     tx_queue_2_packets: 9
     tx_queue_2_bytes: 558
     tx_queue_3_packets: 9
     tx_queue_3_bytes: 774
     VF 0 Rx Packets: 969165504
     VF 0 Rx Bytes: 58149932076
     VF 0 Tx Packets: 969018422
     VF 0 Tx Bytes: 58141106340
     VF 0 MC Packets: 8
```

sc0072

For example:

```
[root@vsr_hyp ~]# ethtool -S enp130s0f0
NIC statistics:
     rx_packets: 1452519449579
     ...
     rx_no_dma_resources: 1307435840907
```

9.2.5.5 OVS Statistics

Use the **ovs-ofctl dump-ports** command, in combination with the **watch** command,
to monitor statistics for transmitted, received, and dropped packets in OVS.

Note: The output for some commands can vary across different versions of OVS. Refer to
OVS documentation for more information.

To determine traffic statistics for a specified OVS, use the **watch -c -n 1 -d ovs-ofctl dump-ports ovs-sf-1** command. This command can also be used for OVS-DPDK.

In the following **watch** command output, the statistics for Transmit (TX) packets match those of the Receive (RX) packets in port 2, while port 1 also shows TX dropped packets.

```
watch -c -n 1 -d ovs-ofctl dump-ports ovs-sf-1
OFPST_PORT reply (xid=0x2): 3 ports
  port  1: rx pkts=84681434, bytes=6777075390, drop=0, errs=0, frame=0, over=0, crc=
0
          tx pkts=83511830, bytes=7502578448, drop=28094, errs=0, coll=0
  port  2: rx pkts=83511830, bytes=7502578448, drop=0, errs=0, frame=0, over=0, crc=
0
          tx pkts=84681434, bytes=6777075390, drop=0, errs=0, coll=0
  port LOCAL: rx pkts=8, bytes=648, drop=0, errs=0, frame=0, over=0, crc=0
          tx pkts=146146696, bytes=10981617152, drop=0, errs=0, coll=0
```

To determine traffic statistics for all available OVSs, use the **watch -c -n 1 -d ovs-dpctl show -s** command. This command is not applicable for OVS-DPDK.

The following output shows the statistics for dropped TX packets in port 2.

```
watch -c -n 1 -d ovs-dpctl show -s
system@ovs-system:
        lookups: hit:168167606 missed:158 lost:0
        flows: 4
        port 0: ovs-system (internal)
                RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                TX packets:0 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:0  TX bytes:0
        port 1: ovs-sf-1 (internal)
                RX packets:8 errors:0 dropped:0 overruns:0 frame:0
                TX packets:146124798 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:648  TX bytes:10979971640 (10.2 GiB)
        port 2: sf.vsr-01.cpm
                RX packets:84668573 errors:0 dropped:0 overruns:0 frame:0
                TX packets:83499183 errors:0 dropped:28094 aborted:0 carrier:0
                collisions:0
                RX bytes:6776039716 (6.3 GiB)  TX bytes:7501447526 (7.0 GiB)
        port 3: sf.vsr-01.iom-1
                RX packets:83499183 errors:0 dropped:0 overruns:0 frame:0
                TX packets:84668573 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:7501447526 (7.0 GiB)  TX bytes:6776039716 (6.3 GiB)
```

9.2.5.6   Packet Captures

Use a tool such as Wireshark to capture packets at various interconnect points. The interconnect points that support Wireshark packet captures are shown in the following system diagram.

Figure 54 shows the major VSR data plane options.

Figure 54   **Major VSR Data Plane Options**

sw0866
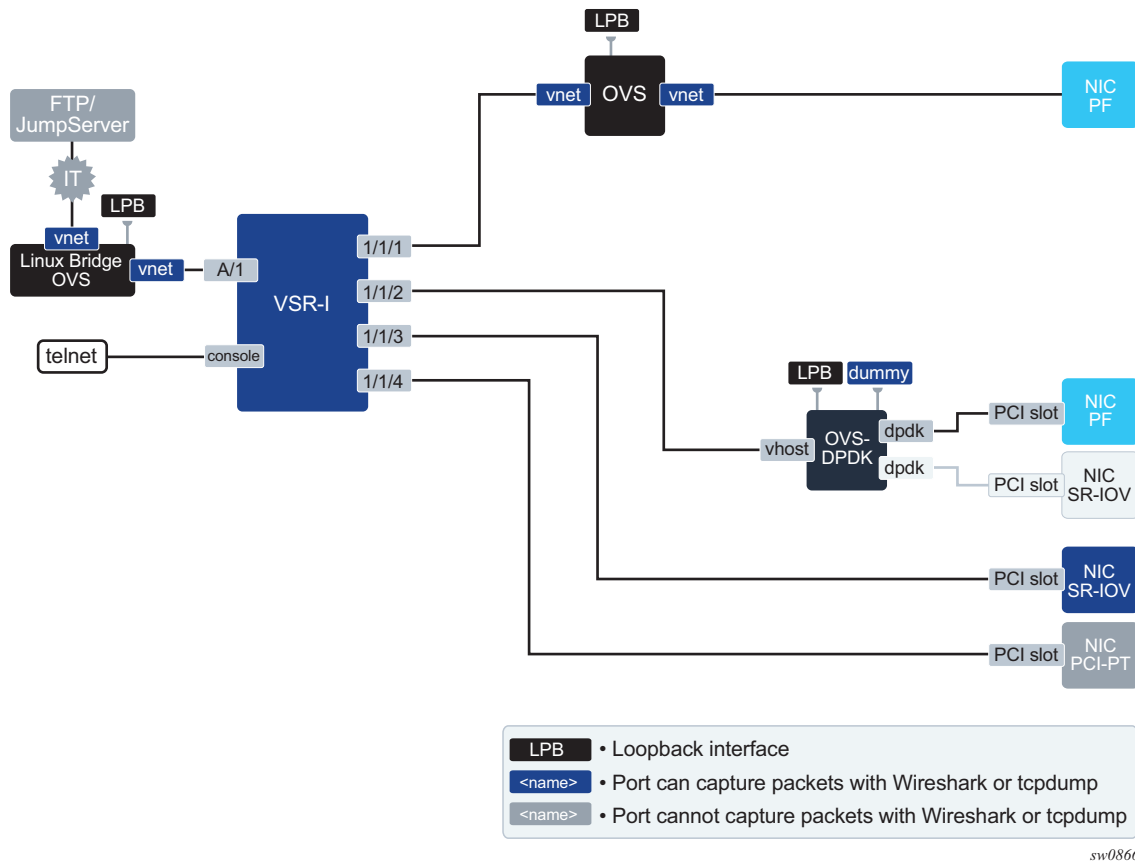
9.2.5.7   Insufficient VM Memory

Avoid memory oversubscription. Check memory consumption using the **show system memory-pools** command to show how much memory is used and how much is remaining.

9.3 Troubleshooting Common Problems

This section provides information about troubleshooting common problems.

9.3.1 vCPUs Not Pinned or Isolated

For CPU or vCPU isolation, ensure that only one VM uses a vCPU and that there is no overlap between VMs and vCPUs.

Review the OpenStack configuration for any errors, as the system may silently ignore mistyped configuration. See Deploying VSR on Linux KVM Hosts Using Libvirt or OpenStack for more information about configuration.

If the vCPUs might not have been pinned or isolated correctly, see the following sections for more information:

- CPU Isolation
- vCPU
- Cputune
- Create Nova Flavors Appropriate for VSR VMs

9.3.2 Insufficient CPU Resources

Insufficient CPU resources can lead to the following general problems:

- excessive packet loss
- lower than expected data plane performance
- control plane instability or performance problems
- system instability

When determining whether CPU capacity is overloaded, consider that the CPU resources for the control plane and data plane are separated.

See the following sections for more CPU resource information:

- vCPU
- Cputune
- CPU

- Sysinfo
- CPU Processor Information
- Adjust Compute Node Resource Allocation
- Create Nova Flavors Appropriate for VSR VMs

9.3.2.1   Control Plane CPU Resources

Control plane CPU resources are overloaded if:

- the output of the **show system cpu** or **show card cpu** commands (VSR Control Plane CPU Utilization) show any task at more than 80% utilization or overall CPU utilization regularly above 99%
- the output of the **htop** command (Host Machine CPU Utilization (HTOP)) shows any of the first *N* pCPU cores (where *N* is the **control-cpu-cores** SMBIOS value) at more than 90% utilization

To remedy control plane CPU resource overload:

- add more vCPUs to the VM (see vCPU)
- allocate more vCPUs to the control plane using the **control-cpu-cores** SMBIOS parameter (see Allocation of vCPUs for Control and Management Tasks)
- enable hyper-threading on the host machine (see Hyper-Threading)

9.3.2.2   Data Plane CPU Resources

Data plane CPU resources are overloaded if the output of the **show card 1 virtual fp** command (VSR Data Plane CPU Utilization) regularly shows any task at more than 99% utilization.

To remedy data plane CPU resource overload:

- add more vCPUs to the VM (see vCPU)
- allocate fewer vCPUs to the control plane using the **control-cpu-cores** SMBIOS parameter (see Allocation of vCPUs for Control and Management Tasks)
- enable hyper-threading on the host machine with the appropriate tuning (see Hyper-Threading)

➡ **Note:** Enabling hyper-threading helps the data plane CPU load only if the vsr-deployment-model is high-packet-touch).

9.3.3   Incorrect Hyper-threading Settings

If hyper-threading might not have been configured correctly, see the following sections for more information:

- Hyper-Threading
- Kernel Parameters
- Cputune
- CPU
- Sysinfo
- Create Nova Flavors Appropriate for VSR VMs
- Instantiating a VSR-I using vSphere Web Client
- Configure CPU Pinning for Deployment on a Hyper-Threaded Host

9.3.4   Incorrect NIC Driver or Firmware Versions in the Host

An incorrect host driver version can cause issues if you are using the ports of the associated NIC in SR-IOV mode. The SR-IOV VF may not bind correctly to the VSR port, resulting in no datapath through that port.

Refer to the *SR OS 20.x.Rx.* Software Release Notes for more information about NIC firmware and driver versions. If the host driver is too old, update it to the recommended version. Because Intel drivers are kernel-dependent, this update requires a compile from the source code. If the host driver is too recent, downgrade it to the recommended version.

➡ **Note:** Kernel or Linux updates may result in a change to the driver version.

See the following sections for more NIC information:

- [VSR Networking](#)
- [NICs](#)
- [Kernel Parameters](#)

9.3.5 Incorrect MTU Settings

MTU settings are present on the host (hypervisor) and guest VM. Generally, MTU configured on a guest VM should be less than or equal to MTU configured on the host; otherwise, packets may be silently dropped at the hypervisor level, which can be difficult to troubleshoot.

Use the following tools to troubleshoot MTU values on SR-IOV, PCI-passthrough, Linux bridge, and OVS.

- Use the **ip link show** *interface-name* command at the host level to check the MTU.
- Use the **show port** *port-name* **detail** SR OS command to check the guest VSR interface MTU.
- Compare the results of the previous checks, if applicable.

Ensure that the guest MTU is lower than the host MTU. See the following sections for more information about MTU settings:

- [Using SR-IOV](#)
- [Sysinfo](#)
- [Enable SR-IOV on OpenStack Controller and Compute Node](#)

9.3.5.1 SR-IOV MTU Settings

Each NIC port used by a VSR in SR-IOV mode must have a minimum MTU of 1500 bytes and the MTU setting in the host must match the MTU setting in VSR; otherwise, packets may be dropped.

9.3.5.1.1 SR-IOV Troubleshooting

MTU settings can result in errors and issues, whether or not trusted mode is enabled with SR-IOV. Ensure that the MTU configured on the NICs will be larger than requested MTU.

In trusted mode, if the requested MTU is bigger than the PF MTU, an error message is shown in the **dmesg** command output, but the VM is not prevented from booting. Unidirectional traffic is observed.

In untrusted mode, if the requested MTU is bigger than the PF MTU, the hypervisor silently ignores a value and traffic with big MTUs is silently dropped. Because this issue does not prevent many protocols (including BGP and RSVP) from working, it may not be easily detectable.

Because multicast traffic does not work with trusted mode disabled, LDP and IGPs that use multicast do not work with untrusted mode.

9.3.5.2   Linux Bridge MTU Settings

The Linux bridge inherits its MTU value from the first virtual or physical interface that is added to it.

→ **Note:** Some MTU configuration commands (such as the **ip link set dev eth0 mtu 9000** command) are not persistent. Due to lack of persistency, if a server using a Linux bridge is rebooted, connectivity may be lost or performance may be severely affected.

Note: Change the MTU settings before starting the VMs.

9.3.6   NUMA Misalignment

If NUMA might not have been configured correctly, see the following sections for more information:

- BIOS Settings
- NUMA
- Kernel Parameters
- vCPU
- Numatune
- Create Nova Flavors Appropriate for VSR VMs
- Deploying the VSR-I vApp using vCloud Director
- Set NUMA Node Affinity

9.3.7   Insufficient Packet Buffer Memory

If the VSR configuration has a lot of queues or if the host vSwitch does not provide queues with sufficient buffer depth, traffic loss may increase, especially if the traffic is bursty. This is because there are not enough packet buffers to absorb the traffic bursts and to service all the queues in a timely manner. If the VSR configuration has a lot of queues, ensure that the **vsr-deployment-model** SMBIOS parameter is set to **queue-scale**.

9.3.7.1   NIC Packet Drops

As data can be written to a hypervisor memory independently of the CPU, it is extensively used by the NIC to write data traffic to a memory for future processing. If the option is disabled, a constant traffic drop of around 0.2%, might be observed after reaching a certain throughput. This drop occurs because the NIC is not capable of writing all incoming traffic to memory and, as a result, drops it. Such dropped traffic indicates that some BIOS options (such as DMA or VT-d or I/OAT) were not set correctly.

9.3.8   Insufficient VM Memory

Correct memory allocation is a key component for VSR operation. Memory consumption must be monitored from a VM perspective; from a hypervisor perspective, all configured memory is allocated to the VM.

VSR requires an allocation of 1 Gbyte for Hugepages. Refer to the *SR OS 20.x.Rx. Software Release Notes* for the minimum amount of memory required for different network functions.

See the following sections for more information about VM memory settings:

- Memory
- Guest Memory Backing

Appendices

-

3HE 15837 AAAA TQZZA 01

Appendix A: VSR Glossary of Key Terms

Table 20   A

Term	Definition
Ansible playbooks	Defined in YAML format and included if commissioning operations are needed on the VNF as part of Mistral workflows or pre/post actions.

Table 21   C

Term	Definition
CentOS	An open source Linux distribution that reuses source code from Red Hat Enterprise Linux.
Config Drive	An OpenStack feature that allows instance-specific configuration data to be written to a read-only virtual disk that is attached to the VM when it boots.
CPU Pinning	A configuration constraint (often expressed as an affinity map), which specifies to the scheduler the (logical) cores that can be used to run a task or set of tasks.

Table 22   D

Term	Definition
DPDK	Data Path Development Kit Open source software (BSD licensed) developed by Intel to improve fast packet processing for NFV data plane applications. DPDK optimizations include poll-mode NIC drivers in Linux user space, huge pages for memory management, and lockless queues.

Table 23   G

Term	Definition
Geneve	The Generic Network Virtualization Encapsulation tunneling protocol.

Table 24   **H**

Term	Definition
Haswell	Intel CPU micro-architecture introduced in 2013 that uses 22-nm process.
Healing	A recovery action that reboots the VM.
HEAT template	An OpenStack HEAT template. The HOT (HEAT Orchestration Template) used by CBAM has access to CBAM built-in parameters and the extensions defined in the VNFD, which can be used to parametrize the resources defined in HOT.
Huge pages	A large block (2MB or 1GB) of physically contiguous virtual memory that has a mapping (in the page table) to physical memory.
Hyper-threading	Intel technology that presents one physical CPU core as two logical processors to the OS.
Hypervisor	Software running on a host machine that creates and manages VMs, and provides the guest O/S in each VM with an abstraction of the physical machine. See also VMM.

Table 25   **I**

Term	Definition
Integrated model	A VSR instance that uses a single VM to support all the functions of one network element.
Intel VT-d	Intel Virtualization Technology for Directed I/O Intel CPU MMU feature that provides hardware assist for mapping a guest virtual address (GVA) to a guest physical address (GPA) to a host physical address (HPA). Avoids the need for VMM to maintain a shadow page table per guest.
Intel VT-x	Intel Virtualization Technology for x86 processors Hardware virtualization support in Intel CPUs that allows guest OS to run natively on x86. Introduces two new CPU modes: VMX root (intended for host/VMM execution) and VMX non-root (intended for guest).

3HE 15837 AAAA TQZZA 01   Issue: 01

Table 26   J

Term	Definition
Javascript helpers	Supported from CBAM as pre/post actions of Mistral workflows (built-in or custom) and as part of Mistral actions.

Table 27   K

Term	Definition
Kernel Space	A block of virtual memory strictly reserved for the OS kernel, kernel extensions and device drivers.
KVM	Kernel-based Virtual Machine Linux kernel module that allows a user space program, such as QEMU, to access the hardware virtualization features of the CPU.

Table 28   L

Term	Definition
L3 Cache	Fast on-chip memory of the CPU that stores frequently accessed data, saving time to access main memory. It is shared by all cores of the CPU.
Libvirt	Open source Linux package that provides a common set of APIs for creating and managing the VMs on one host, independent of hypervisor. Libvirt uses XML files to define the properties of VM instances, networks, and other devices; the **virsh** command line toolset is provided.
Linux Bridge	Software implementation of a bridge that forwards Ethernet frames based on destination MAC address; bridging is performed by a kernel module controlled by the **brctl** userspace program installed with the **bridge-utils** package. A Linux bridge is supported by various Linux OS.

Table 29   M

Term	Definition
MANO	Management and Orchestration A reference architecture defined by ETSI NFV study group that gives generic names to the functional components of a complete NFV solution.

Table 29   **M (Continued)**

Term	Definition
Mistral workflows	Defined in YAML format and included if custom operations (interfaces) are defined in the VNFD. Each workflow includes a set of inputs, which can affect the execution of the workflow.

Table 30   **N**

Term	Definition
NUMA	Non-Uniform Memory Access An optimization for multi-CPU systems where each processor has its own memory.

Table 31   **O**

Term	Definition
OpenStack	An open source cloud orchestration platform (VIM) managed by the non-profit OpenStack Foundation, it includes various components such as Nova (compute), Neutron (networking), Glance (image service), Cinder (block storage), and Dashboard (GUI).
OVA	Open Virtual Application A tar archive of an OVF package.
OVF	Open Virtualization Format A DMTF standard format for packaging software to be run in VMs. An OVF package contains an XML-based OVF descriptor file (.ovf), one or more disk images, and other auxiliary files. The OVF descriptor file specifies HW requirements and lists references to other files in the OVF package.

Table 31   **O (Continued)**

Term	Definition
OVS	Open Virtual Switch
	Open-source software implementation of a multi-layer switch, it supports standard bridging protocols, monitoring protocols (sFlow, NetFlow), and programmatic extensions (OpenFlow, OVSDB).
	Main OVS components are: userspace daemon (ovs-vswitchd), database daemon (ovsdb-server), and kernel module.
	The kernel module implements 'fast path' using a flow cache table populated by ovs-vswitchd. The first packet of a flow goes to ovs-switchd for slow-path processing. ovs-vswitchd communicates with the kernel using the netlink protocol, and with ovsdb-server using the OVSDB protocol.

Table 32   **P**

Term	Definition
Paravirtualization	Technique where the guest and hypervisor coordinate to optimize performance in a virtualized environment.
Prefetching	Transferring data before it is required to optimize performance.

Table 33   **Q**

Term	Definition
QCOW2	A virtual disk image format supported by QEMU.
QEMU	Quick Emulator
	Open source hypervisor typically used with KVM that emulates a broad range of devices including CPUs, disks, PCIe chipsets, USB devices, and serial ports.

Table 34   **R**

Term	Definition
RHEL	Red Hat Enterprise Linux

Table 34   **R (Continued)**

Term	Definition
RSS	Receive Side Scaling A feature supported by some NICs to classify incoming packets into different receive queues based on 5-tuple flow. Each queue has its own interrupt handled by its own core, which may improve receive throughput.

Table 35   **S**

Term	Definition
SMBIOS	System Management BIOS Data structures and access methods for storing and reading BIOS information.
SR-IOV	A PCI-SIG standard that allows a PCIe device to appear as multiple separate PCIe devices, allowing multiple VM vNIC interfaces to share the same physical NIC port for communications.

Table 36   **U**

Term	Definition
Ubuntu	A Debian-based common Linux distribution.
User Space	A block of virtual memory where application software and some drivers execute.

Table 37   **V**

Term	Definition
vFP	A software implementation of a SR-series router forwarding plane, optimized for x86 CPU instructions and memory architecture. The vFP uses the same control APIs as hardware-based IOMs/XCMs.
VIM	Virtualized Infrastructure Manager A MANO component responsible for managing the NFV infrastructure including compute, storage, and network resources. OpenStack and CloudStack are typical VIMs.

Table 37   **V (Continued)**

| Term	Definition
VirtIO	A paravirtualized I/O framework where buffers are transferred between the guest-side VirtIO driver and the host-side VirtIO driver.
VHost-net	A device driver that runs in the host kernel and performs the virtqueue operations of the host-side VirtIO driver. It delivers higher performance than complete emulation of the host-side VirtIO driver in QEMU (avoids system calls from userspace, supports zero-copy TX operation).
VM	Virtual Machine
VMDK	Virtual Machine Disk The virtual disk image format used by VMware VMs.
VMware vSphere	A virtualization product suite sold by VMware, it includes ESXi hypervisor, vCenter server, vSphere Web client, and advanced feature add-ons including vMotion, High Availability, Fault Tolerance, Distributed Switch, Distributed Resource Scheduler.
VNF package	A TOSCA Cloud Service Archive (CSAR) zip file containing all artifacts needed for VNF instantiation: • VNF descriptor (VNFD) • VNFD metadata • HEAT templates (HOT) • Mistral workflows • Ansible playbooks • Javascript helpers
VNFD	Describes the virtual resources required by the VNF (compute, network, and storage) and the lifecycle management operations it supports. References all other VNF artifacts that are part of the VNF package. The template is defined in YAML format and is reusable. The VNFD supports parametrization and facilitates the deployment of multiple VNF instances through a single VNFD/VNF package. When CBAM is integrated with an orchestrator, the VNFD is also the contract between a VNF and the orchestrator that fully describes the expectations of the VNF from the infrastructure.
VNFD - metadata	Provides information to uniquely identify a VNF descriptor (ID, version, and product).
VNFD - requirements	Describes the connectivity requirements (for externally managed provider networks) that are expected to be available to support VNF connectivity.

Table 37   **V (Continued)**

Term	Definition
VNFD - extensions	Modifiable parameters defined per VNF instance.
VNFD - interface	The supported lifecycle operations for a VNF that are implemented either through the CBAM built-in Mistral workflows or by custom Mistral workflows included in the VNF package.
VNFD - node templates	Describes the following virtual resources needed for VNF instantiation: • VMs—virtual deployment units (VDUs) associated with a software image, flavor (vCPU and memory), and storage requirements • internal networks—virtual links • VM connectivity—a set of virtual connection points that are associated with internal or external networks
VNFD - heat mapping	The HEAT mapping section of the VNFD associates the abstract resource descriptions with their corresponding implementations in HOT.
VNFM	VNF Manager The MANO component responsible for lifecycle management of VNF instances. Coordinates with EMS/NMS. This role is provided by CloudBand CBAM for VSR instances.
VxLAN	Virtual eXtensible Local Area Network A method of encapsulating Ethernet frames inside IP/UDP packets to create a tenant-specific overlay network within a data center.

Table 38   **X**

Term	Definition
x2APIC	x2 Advanced Programmable Interrupt Controller An Intel programmable interrupt controller.

Customer Document and Product Support
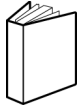
Customer Documentation

[Customer Documentation Welcome Page](#)

Technical Support

[Product Support Portal](#)

Documentation Feedback

[Customer Documentation Feedback](#)