

How to Configure the NFX250

Published
2021-07-12

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

How to Configure the NFX250

Copyright © 2021 Juniper Networks, Inc. All rights reserved.

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at <https://support.juniper.net/support/eula/>. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About This Guide | ix

1

Overview

NFX250 Overview | 2

JDM Architecture Overview | 6

Understanding Disaggregated Junos OS | 7

Understanding Physical and Virtual Components | 9

Disaggregated Junos OS VMs | 12

Virtio and SR-IOV Usage | 14

JDM CLI Overview | 18

Understanding the JDM CLI | 18

Accessing the JDM Shell, JDM CLI, and JCP Prompts in a Disaggregated Junos OS Platform | 19

Accessing the JDM CLI | 20

Accessing the JDM Shell | 20

Accessing the JCP Prompt from the JDM CLI | 20

Accessing the Hypervisor from the JDM CLI | 21

Accessing the ipsec-nm from the JDM CLI | 21

Software Upgrade Path for NFX250 Devices | 21

Junos OS Releases Supported on NFX Series Hardware | 23

2

Installation and Configuration

Performing Initial Software Configuration on an NFX250 Device | 26

Installing Software on NFX250 Devices | 29

Downloading and Installing Software | 30

Downloading Software | 30

Software Installation on NFX250 Network Services Platform | 34

Upgrading an Image on the Disaggregated Junos OS Platform | 35

Reverting the System to the Factory-Default Configuration | 40

Rebooting the System | 40

Configuring JDM User Accounts and Authentication | 41

JDM User Accounts Overview | 41

Configuring JDM User Accounts and Authentication | 42

YANG files on NFX250 Devices | 44

Understanding YANG on NFX250 Devices | 44

Generating YANG Files | 45

Synchronizing Time Using NTP | 48

Configuring Management Interfaces for JDM | 49

JDM Management Interfaces Overview | 50

Configuring the Out-of-Band Management Interface for JDM | 52

Configuring the Out-of-Band Management Interface with IPv4 Addressing for JDM | 52

Configuring the Out-of-Band Management Interface with IPv6 Addressing for JDM | 53

Configuring the In-Band Management Interface for JDM | 53

Configuring the Out-of-Band Management Interface for Hypervisor | 57

Configuring the Out-of-Band Management Interface with IPv4 Addressing for Hypervisor | 57

Configuring the Out-of-Band Management Interface with IPv6 Addressing for Hypervisor | 57

Configuring Remote Access to JDM | 58

Configuring SSH Service and NETCONF-Over-SSH Connections for Remote Access to the Disaggregated Junos OS Platform | 58

Configuring HTTP Access to the Disaggregated Junos OS Platform | 59

Configuring HTTPS Access to the Disaggregated Junos OS Platform | 59

Configuring Enhanced Orchestration and Hugepages | 60

Enhanced Orchestration | 60

Hugepages | 61

Managing Virtual Network Functions Using JDM | 64

Understanding Virtual Network Functions | 65

Prerequisites to Onboard Virtual Network Functions on NFX250 Devices | 67

Prerequisites for VNFs | 67

Managing the VNF Life Cycle | 68

Planning Resources for a VNF | 68

Managing the VNF Image | 70

Preparing the Bootstrap Configuration | 71

Launching a VNF | 71

Allocating Resources for a VNF | 72

Managing VNF States | 78

Managing VNF MAC Addresses | 79

Managing MTU | 79

Accessing a VNF from JDM | 80

Viewing List of VNFs | 81

Displaying the VNF Details | 81

Deleting a VNF | 82

Creating the vSRX VNF on the NFX250 Platform | 82

Configuring the vMX Virtual Router as a VNF on NFX250 | 85

Virtual Route Reflector on NFX250 Overview | 87

Configuring vRR as a VNF on NFX250 | 89

Configuring vRR VNF on NFX250 in Standard Orchestration Mode | 89

Configuring vRR VNF on NFX250 in Enhanced Orchestration Mode | 102

Configuring Cross-connect | 105

Configuring Analyzer VNF and Port-mirroring | 107

Configuring Service Chaining Using JDM | 109

Understanding Service Chaining on Disaggregated Junos OS Platforms | 109

Configuring Service Chaining Using VLANs | 110

Configuring Service Chaining Using DHCP Services on VLANs | 110

Example: Configuring Service Chaining Using VLANs on NFX250 Network Services Platform | 111

Requirements | 112

Overview | 112

Configuration | 114

Example: Configuring Service Chaining Using SR-IOV on NFX250 Network Services Platform | 118

Requirements | 119

Overview | 119

Configuration | 121

ADSL2 and ADSL2+ Interfaces on NFX250 Devices | 126

ADSL Interface Overview | 126

Example: Configuring ADSL SFP Interface on NFX250 Devices | 127

Requirements | 128

Overview | 128

Configuration | 128

Results | 130

VDSL2 Interfaces on NFX250 Devices | 131

VDSL Interface Overview | 131

VDSL2 Network Deployment Topology | 132

VDSL2 Interface Support on NFX Series Devices | 134

Example: Configuring VDSL SFP Interface on NFX250 Devices | 136

Requirements | 136

Overview | 136

Configuration | 137

Results | 139

3

Monitoring and Troubleshooting JDM

Configuring SNMP on JDM | 141

Configuring SNMP Community | 141

Configuring SNMP System Parameters | 141

Configuring SNMP v3 | 142

Configuring SNMP Traps | 143

Querying SNMP MIBs | 143

Managing Traps | 144

Managing the Log Files and Core Files | 144

Viewing and Managing Centralized Log Files | 145

Enabling Centralized Logging | 145

Viewing Log Messages | 146

Managing Core Files | 146

Viewing Core Files | 147

Recovering the Root Password for NFX250 | 147

4

Virtual Network Functions Configuration Statements and Operational Commands

cross-connect | 152

features | 155

host-os forwarding-options analyzer | 156

hugepages | 158

image | 160

init-descriptor | 162

interfaces | 163

mac-address | 166

mapping | 167

memory | 169

mtu | 171

no-autostart | 173

offloads | 174

pci-address | 176

size | 177

storage | 178

type | 181

virtual-cpu | 182

virtual-network-functions | 184

`vjunos0` | 189

`vnf-name` | 191

`show virtual-network-functions` | 195

`show vlans` | 201

About This Guide

Use this guide to perform initial provisioning, configure Junos OS features, chain multiple virtualized network functions, monitor, and manage the NFX250 devices using the Juniper Device Manager (JDM).

RELATED DOCUMENTATION

[Release Notes: Junos® OS 15.1X53-D45 for NFX250 Network Services Platform](#)

[Release Notes: Junos OS Release 15.1X53-D40 for NFX Series](#)

1

CHAPTER

Overview

[NFX250 Overview](#) | 2

[JDM Architecture Overview](#) | 6

[JDM CLI Overview](#) | 18

[Software Upgrade Path for NFX250 Devices](#) | 21

[Junos OS Releases Supported on NFX Series Hardware](#) | 23

NFX250 Overview

IN THIS SECTION

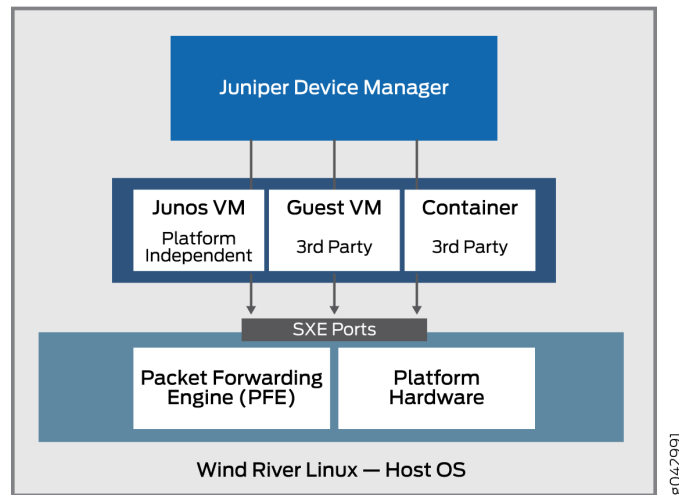
- [Benefits of NFX250 Network Services Platform | 3](#)
- [NFX250 Models | 4](#)
- [NFX250 Components | 5](#)

The Juniper Networks NFX250 Network Services Platform is a secure, automated, software-driven customer premised equipment (CPE) platform that delivers virtualized network and security services on demand. An integral part of Juniper's fully automated Cloud CPE solution suite for NFV, this high-performance virtualized services platform helps service providers improve overall operational efficiency and service agility while empowering enterprise customers with immediate access to custom-designed managed services. Simultaneously supporting multiple Juniper and third-party VNFs on a single device and providing built-in, dynamic, policy-based routing, the NFX250 addresses the needs of small to midsize businesses as well as large multinational or distributed enterprises with a single, highly scalable solution.

NFX250 Network Services Platform are Juniper Network's secure, automated, software-driven customer premises equipment (CPE) devices that deliver virtualized network and security services on demand. Leveraging Network Functions Virtualization (NFV) and built on the Juniper Cloud CPE solution, NFX250 enables service providers to deploy and service chain multiple, secure, high-performance virtualized network functions (VNFs) as a single device. This automated, software-driven solution dynamically provisions new services on demand.

Figure 1 on page 3 illustrates the important position JDM occupies in the overall architecture.

Figure 1: Position of the Juniper Device Manager



Benefits of NFX250 Network Services Platform

The NFX250 is an integrated branch router and switch with a multicore CPU that enables it to run multiple Virtual Network Functions (VNFs). The NFX250 Network Services Platform provides these benefits:

- Simultaneously supports multiple Juniper and third-party VNFs on a single device, providing built-in, dynamic, and policy-based routing
- Provides an open framework that supports industry standards, protocols, and seamless API integration
- Supports a variety of flexible deployments. A distributed services deployment model ensures high availability, performance, and compliance
- Incorporates many advanced security features. Secure Boot feature safeguards device credentials, automatically authenticates system integrity, verifies system configuration, and enhances overall platform security
- Modular software architecture provides high performance and scalability for routing, switching, and security enhanced by carrier-class reliability
- Automated configuration eliminates complex device setup and delivers a plug-and-play experience

- High performance simplifies network topologies and operations

NFX250 Models

The NFX250 device is available in four models.

Product Number	Specifications	Features
NFX250-S1	<p>2.0 GHz 6-core Intel CPU</p> <p>16 GB of memory and 100 GB of solid-state drive (SSD) storage</p> <p>Eight 1-GbE network ports, two 1-GbE RJ-45 ports which can be used as either access ports or as uplinks, two SFP ports, two SFP+ ports, one Management port, and two Console ports</p>	Basic Layer 2/Layer 3
NFX250-S2	<p>2.0 GHz 6-core Intel CPU</p> <p>32 GB of memory and 400 GB of SSD storage</p> <p>Eight 1-GbE network ports, two 1-GbE RJ-45 ports which can be used as either access ports or as uplinks, two SFP ports, two SFP+ ports, one Management port, and two Console ports</p>	Basic Layer 2/Layer 3

(Continued)

Product Number	Specifications	Features
NFX250-LS1	<p>1.6 GHz 4-core Intel CPU</p> <p>16 GB of memory and 100 GB of solid-state drive (SSD) storage</p> <p>Eight 1-GbE network ports, two 1-GbE RJ-45 ports which can be used as either access ports or as uplinks, two SFP ports, two SFP+ ports, one Management port, and two Console ports</p>	<p>Supports up to 100 MBPS throughput Secure Router functionality for the following features:</p> <ul style="list-style-type: none"> • IPSec VPN • NAT • Stateful Firewall • Routing services – BGP, OSPF, DHCP, IPv4 and IPv6
NFX250-S1E	<p>2.0 GHz 6-core Intel CPU</p> <p>16 GB of memory and 200 GB of solid-state drive (SSD) storage</p> <p>Eight 1-GbE network ports, two 1-GbE RJ-45 ports which can be used as either access ports or as uplinks, two SFP ports, two SFP+ ports, one Management port, and two Console ports</p>	<p>Supports up to 100 MBPS throughput Secure Router functionality for the following features:</p> <ul style="list-style-type: none"> • IPSec VPN • NAT • Stateful Firewall • Routing services – BGP, OSPF, DHCP, IPv4 and IPv6

NFX250 Components

The NFX250 consists of the following key components:

- *Juniper Device Manager*: The Juniper Device Manager (JDM) is a low-footprint Linux container that provides these functions:

- Virtual Machine (VM) lifecycle management
- Device management and isolation of host OS from user installations
- NIC, single-root I/O virtualization (SR-IOV), and virtual input/output (VirtIO) interface provisioning
- Support for the Network Service Orchestrator module to connect to Network Activator
- Inventory and resource management
- Internal network and image management
- Service chaining—provides building blocks such as virtual interfaces and bridges for users to implement service chaining policies
- Virtual console access to VNFs including vSRX and vjunos
- *Junos Control Plane*: Junos Control Plane (JCP) is the Junos VM running on the hypervisor. You can use JCP to configure the network ports of the NFX250 device, and JCP runs by default as vjunos0 on NFX250. You can log on to JCP from JDM using the SSH service and the command-line interface (CLI) is the same as Junos.

RELATED DOCUMENTATION

| [Performing Initial Software Configuration on an NFX250 Device](#) | 26

JDM Architecture Overview

IN THIS SECTION

- [Understanding Disaggregated Junos OS](#) | 7
- [Understanding Physical and Virtual Components](#) | 9
- [Disaggregated Junos OS VMs](#) | 12
- [Virtio and SR-IOV Usage](#) | 14

Understanding Disaggregated Junos OS

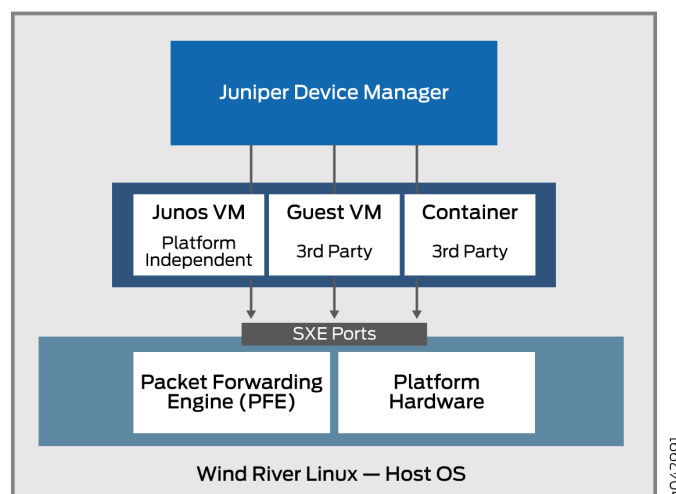
Many network equipment vendors have traditionally bound their software to purpose-built hardware and sold customers the bundled and packaged software–hardware combination. However, with the disaggregated Junos OS architecture, Juniper Network devices are now aligned with networks that are cloud-oriented, open, and rely on more flexible implementation scenarios.

The basic principle of the disaggregated Junos OS architecture is decomposition (*disaggregation*) of the tightly bound Junos OS software and proprietary hardware into virtualized components that can potentially run not only on Juniper Networks hardware, but also, on white boxes or bare-metal servers. In this new architecture, the Juniper Device Manager (JDM) is a virtualized root container that manages software components.

The JDM is the only root container in the disaggregated Junos OS architecture (there are other industry models that allow more than one root container, but the disaggregated Junos OS architecture is not one of them). The disaggregated Junos OS is a *single-root* model. One of the major functions of JDM is to prevent modifications and activities on the platform from impacting the underlying host OS (usually Linux). As the root entity, the JDM is well-suited for that task. The other major function of JDM is to make the hardware of the device look as much like a traditional Junos OS–based physical system as possible. This also requires some form of root capabilities.

Figure 2 on page 7 illustrates the important position JDM occupies in the overall architecture.

Figure 2: Position of the Juniper Device Manager



A VNF is a consolidated offering that contains all the components required for supporting a fully virtualized networking environment. A VNF has network optimization as its focus.

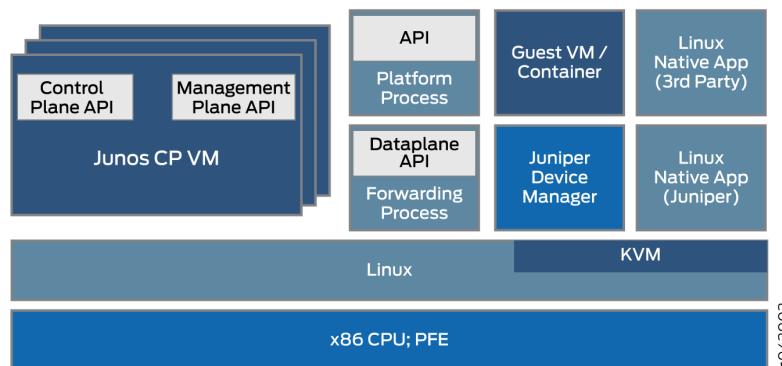
JDM enables:

- Management of guest virtualized network functions (VNFs) during their life cycle.
- Installation of third-party modules.
- Formation of VNF service chains.
- Management of guest VNF images (their binary files).
- Control of the system inventory and resource usage.

Note that some implementations of the basic architecture include a Packet Forwarding Engine as well as the usual Linux platform hardware ports. This allows better integration of the Juniper Networks data plane with the bare-metal hardware of a generic platform.

The disaggregated Junos OS architecture enables JDM to handle virtualized network functions such as a firewall or Network Address Translation (NAT) functions. The other VNFs and containers integrated with JDM can be Juniper Networks products or third-party products as native Linux applications. The basic architecture of the disaggregated Junos OS is shown schematically in [Figure 3 on page 8](#).

Figure 3: Basic Disaggregated Junos OS Architecture



NOTE: There are multiple ways to implement the basic disaggregated Junos OS architecture on various platforms. Details can vary greatly. This topic describes the overall architecture.

The virtualization of the simple software process running on fixed hardware poses several challenges in the area of interprocess communication. How does, for example, a VNF with a NAT function work with a firewall running as a container on the same device? After all, there might be only one or two external Ethernet ports on the whole device, and the processes are still internal to the device. One benefit is the fact that the interfaces between these virtualized processes are often virtualized themselves, perhaps as *SXE ports*, which means that you can configure a type of MAC-layer bridge between processes directly,

or between a process and the host OS and then between the host OS and another process. This supports the chaining of services as traffic enters and exits the device.

JDM provides users with a familiar Junos OS CLI and handles all interactions with underlying Linux kernel to maintain the “look and feel” of a Juniper Networks device.

Some of the benefits of the disaggregated Junos OS are:

- The whole system can be managed like managing a server platform.
- Customers can install third-party applications, tools, and services, such as Chef, Wireshark, or Quagga, in a virtual machine (VM) or container.
- These applications and tools can be upgraded by using typical Linux repositories and are independent of Junos OS releases.
- Modularity increases reliability because faults are contained within the module.
- The control and data planes can be programmed directly through APIs.

Understanding Physical and Virtual Components

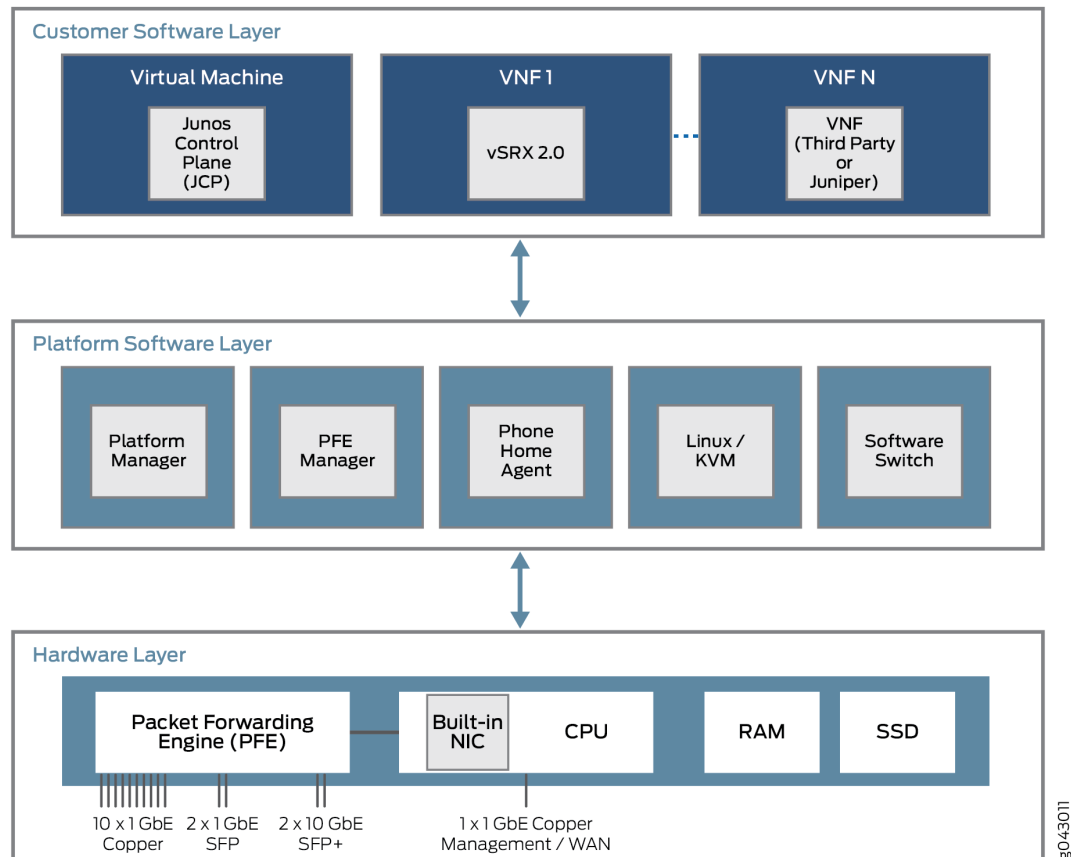
In the disaggregated Junos OS Network Functions Virtualization (NFV) environment, device components might be physical or virtual. The same physical-virtual distinction can be applied to interfaces (ports), the paths that packets or frames take through the device, and other aspects such as CPU cores or disk space.

The disaggregated Junos OS specification includes an architectural model. The architectural model of a house can have directions for including a kitchen, a roof, and a dining room, and can represent various kinds of dwellings; from a seaside cottage to a palatial mansion. All these houses look very different, but still follow a basic architectural model and share many characteristics.

Similarly, in the case of the disaggregated Junos OS architectural models, the models cover vastly different types of platforms, from simple customer premises equipment (CPE) to complex switching equipment installed in a large data center, but have some basic characteristics that the platforms share.

What characteristics do these platforms share? All disaggregated Junos OS platforms are built on three layers. These layers and some possible content are shown in [Figure 4 on page 10](#).

Figure 4: Physical and Virtual Layers in the Disaggregated Junos OS



The lowest layer is the hardware layer. In addition to memory (RAM) and disk space (SSD), the platform hardware has a multi-core CPU with an external NIC port used for management. In some cases, there will be a single NIC port used for the control and data plane, but that port can also be used to communicate with a Packet Forwarding Engine for user traffic streams.

The platform software layer sits on top of the hardware layer. All platform-dependent functions take place here. These functions can include a software switching function for various virtual components to bridge traffic between them. A Linux or kernel-based virtual machine (KVM) runs the platform, and, in some models, a phone home agent contacts a vendor or service provider device to perform autoconfiguration tasks. The phone home agent is particularly preferred for smaller CPE platforms.

Above the platform software layer is the customer software layer, which performs various platform-independent functions. Some of the components might be Juniper Networks virtual machines, such as a virtual SRX device (vSRX) or the Junos Control Plane (JCP). The JCP works with the JDM to make the

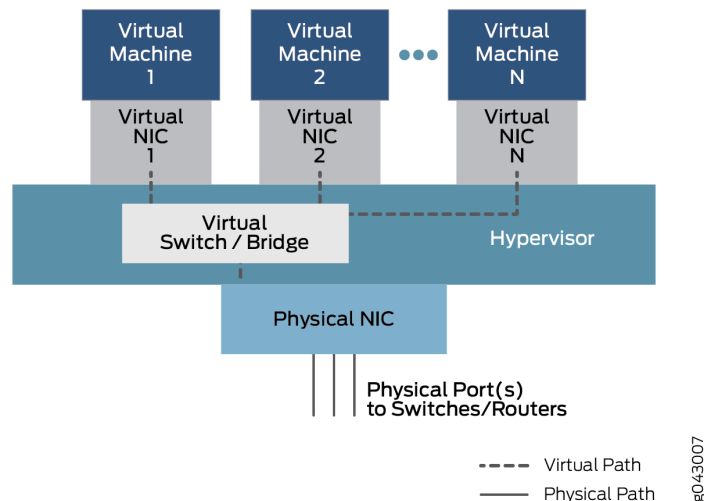
device resemble a dedicated Juniper Networks platform, but one with a lot more flexibility. Much of this flexibility comes from the ability to support one or more VNFs that implement a virtualized network function (VNF). These VNFs consist of many types of tasks, such as Network Address Translation (NAT), specialized Domain Name System (DNS) server lookups, and so on.

Generally, there are a fixed number of CPU cores, and a finite amount of disk space. But in a virtual environment, resource allocation and use is more complex. Virtual resources such as interfaces, disk space, memory, or cores are parceled out among the VNFs running at the time, as determined by the VNF image.

The VNFs, whether virtual machines (VMs) or containers, which share the physical device are often required to communicate with each other. Packets or frames enter a device through a physical interface (a port) and are distributed to some initial VNF. After some processing of the traffic flow, the VNF passes the traffic over to another VNF if configured to do so, and then to another, before the traffic leaves the physical device. These VNFs form a data plane service chain that is traversed inside the device.

How do the VNFs, which are isolated VMs or containers, pass traffic from one to the other? The service chain is configured to pass traffic from a physical interface to one or more internal virtual interfaces. Therefore, there are virtual NICs associated with each VM or Container, all connected by a virtual switch or bridge function inside the device. This generic relationship, which enables communication between physical and virtual interfaces is shown in [Figure 5 on page 11](#).

Figure 5: Physical and Virtual Component Communication



In this general model, which can have variations in different platforms, data enters through a port on the physical NIC and is bridged through the virtual switch function to Virtual Machine1 through Virtual NIC 1, based on destination MAC address. The traffic can also be bridged through another configured virtual interface to Virtual Machine2 or more VNFs until it is passed back to a physical port and exits the device.

For configuration purposes, these interfaces might have familiar designations such as ge-0/0/0 or fxp0, or new designations such as sxe0 or hsxe0. Some might be *real*, but internal ports (such as sxe0), and some might be completely virtual constructs (such as hsxe0) needed to make the device operational.

Disaggregated Junos OS VMs

Cloud computing enables applications to run in a virtualized environment, both for end-user server functions and network functions needed to connect scattered endpoints across a large data center, or even among multiple data centers. Applications and network functions can be implemented by virtualized network functions (VNFs). What are the differences between these two types of packages and why would someone use one type or the other?

Both VNFs and containers allow the multiplexing of hardware with tens or hundreds of VNFs sharing one physical server. This allows not only rapid deployment of new services, but also extension and migration of workloads at times of heavy use (when extension can be used) or physical maintenance (when migration can be used).

In a cloud computing environment, it is common to employ VNFs to do the heavy work on the massive server farms that characterize big data in modern networks. Server virtualization allows applications written for different development environments, hardware platforms, or operating systems to run on generic hardware that runs an appropriate software suite.

VNFs rely on a hypervisor to manage the physical environment and allocate resources among the VNFs running at any particular time. Popular hypervisors include Xen, KVM, and VMWare ESXi, but there are many others. The VNFs run in the user space on top of the hypervisor and include a full implementation of the VM application's operating system. For example, an application written in the C++ language and compiled and run on Microsoft Windows operating system can be run on a Linux operating system using the hypervisor. In this case, Windows is a guest operating system.

The hypervisor provides the guest operating system with an emulated view of the hardware of the VNFs. Among other resources such as disk space or memory, the hypervisor provides a virtualized view of the network interface card (NIC) when endpoints for different VMs reside on different servers or hosts (a common situation). The hypervisor manages the physical NICs and exposes only virtualized interfaces to the VNFs.

The hypervisor also runs a virtual switch environment, which allows the VNFs at the VLAN frame layer to exchange packets inside the same box, or over a (virtual) network.

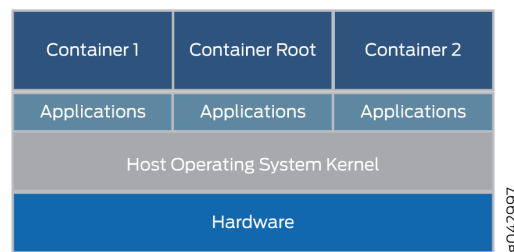
The biggest advantage of VNFs is that most applications can be easily ported to the hypervisor environment and run well without modification.

The biggest drawback is that, often the resource-intensive overhead of the guest operating system must include a complete version of the operating system even if the function of the entire VNF is to provide a simple service such as a domain name system (DNS).

Containers, unlike VNFs, are purpose-built to be run as independent tasks in a virtual environment. Containers do not bundle an entire operating system inside like VNFs do. Containers can be coded and bundled in many ways, but there are also ways to build standard containers that are easy to maintain and extend. Standard containers are much more open than containers created in a haphazard fashion.

Standard Linux containers define a unit of software delivery called a standard container. Instead of encapsulating the whole guest operating system, the standard container encapsulates only the application and any dependencies required to perform the task the application is programmed to perform. This single runtime element can be modified, but then the container must be rebuilt to include any additional dependencies that the extended function might need. The overall architecture of containers is shown in [Figure 6 on page 13](#).

Figure 6: Containers–Overall Architecture



The containers run on the host OS kernel and not on the hypervisor. The container architecture uses a container engine to manage the underlying platform. If you still want to run VNFs, the container can package up a complete hypervisor and guest OS environment as well.

Standard containers include:

- A configuration file.
- A set of standard operations.
- An execution environment.

The name *container* is borrowed from the shipping containers that are used to transport goods around the world. Shipping containers are standard delivery units that can be loaded, labelled, stacked, lifted, and unloaded by equipment built specifically to handle the containers. No matter what is inside, the container can be handled in a standard fashion, and each container has its own user space that cannot be used by other containers. Although [Docker](#) is a popular container management system to run containers on a physical server, there are alternatives such as Drawbridge or Rocket to consider.

Each container is assigned a virtual interface. Container management systems such as Docker include a virtual Ethernet bridge connecting multiple virtual interfaces and the physical NIC. Configuration and environment variables in the container determine which containers can communicate with each other, which can use the external network, and so on. External networking is usually accomplished with NAT although there are other methods because, containers often use the same network address space.

The biggest advantage of containers is that they can be loaded on a device and executed much faster than VNFs. Containers also use resources much more sparingly— you can run many more containers than VNFs on the same hardware. This is because containers do not require a full guest operating system or boot time. Containers can be loaded and run in milliseconds, not tens of seconds. However, the biggest drawback with containers is that they have to be written specifically to conform to some standard or common implementation, whereas VNFs can be run in their native state.

Virtio and SR-IOV Usage

IN THIS SECTION

- [Understanding Virtio Usage | 14](#)
- [Understanding SR-IOV Usage | 16](#)
- [Comparing Virtio and SR-IOV | 17](#)

You can enable communication between a Linux-based virtualized device and a Network Functions Virtualization (NFV) module either by using virtio or by using suitable hardware and single-root I/O virtualization (SR-IOV). Each method has distinct characteristics.

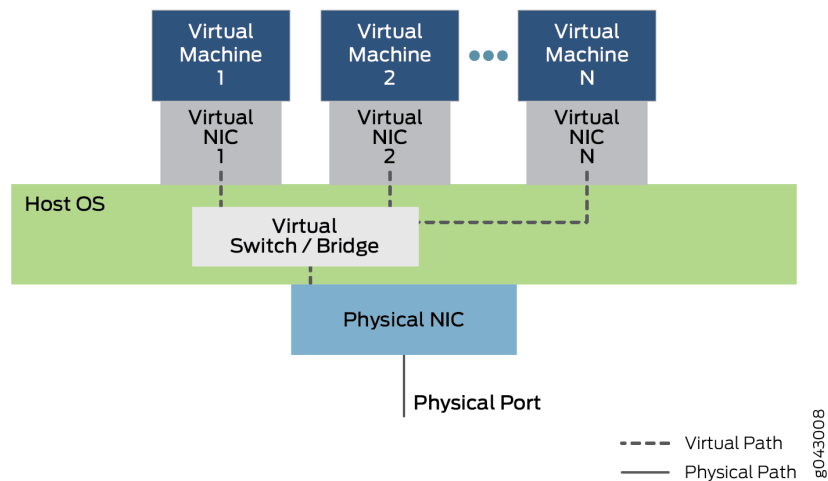
Understanding Virtio Usage

When a physical device is virtualized, both physical NIC interfaces and external physical switches as well as the virtual NIC interfaces and internal virtual switches coexist. So when the isolated VNFs in the device, each with their own memory and disk space and CPU cycles, attempt to communicate with each other, the multiple ports, MAC addresses, and IP addresses in use pose a challenge. With the virtio library, traffic flow between the isolated virtual functions becomes simpler and easier.

Virtio is part of the standard Linux libvirt library of useful virtualization functions and is normally included in most versions of Linux. Virtio is a software-only approach to inter-VNF communication. Virtio provides a way to connect individual virtual processes. The bundled nature of virtio makes it possible for any Linux-run device to use virtio.

Virtio enables VNFs and containers to use simple internal bridges to send and receive traffic. Traffic can still arrive and leave through an external bridge. An external bridge uses a virtualized internal NIC interface on one end of the bridge and a physical external NIC interface on the other end of the bridge to send and receive packets and frames. An internal bridge, of which there are several types, links two virtualized internal NIC interfaces by bridging them through a virtualized internal switch function in the host OS. The overall architecture of virtio is shown in [Figure 7 on page 15](#).

Figure 7: VNF Bridging with Virtio



[Figure 7 on page 15](#) shows the inner structure of a server device with a single physical NIC card running a host OS (the outer cover of the device is not shown). The host OS contains the virtual switch or bridge implemented with virtio. Above the OS, several virtual machines employ virtual NICs that communicate through virtio. There are multiple virtual machines running, numbered 1 to N in the figure. The standard “dot dot dot” notation indicates possible virtual machines and NICs not shown in the figure. The dotted lines indicate possible data paths using virtio. Note that traffic entering or leaving the device does so through the physical NIC and port.

[Figure 7 on page 15](#) also shows traffic entering and leaving the device through the internal bridge. Virtual Machine 1 links its virtualized internal NIC interface to the physical external NIC interface. Virtual Machine 2 and Virtual Machine N link internal virtual NICs through the internal bridge in the host OS. Note that these interface might have VLAN labels associated with them, or internal interface names. Frames sent across this internal bridge between VNFs never leave the device. Note the position of the bridge (and virtualized switch function) in the host OS. Note the use of simple bridging in the device. These bridges can be configured either with *regular* Linux commands or the use of CLI configuration statements. Scripts can be used to automate the process.

Virtio is a virtualization standard for disk and network device drivers. Only the guest device driver (the devices driver for the virtualized functions) needs to *know* that it is running in a virtual environment.

These drivers cooperate with the hypervisor and the virtual functions get performance benefits in return for the added complication. Virtio is architecturally similar to, but not the same as, Xen paravirtualized device drivers (drivers added to a guest to make them faster when running on Xen). VMWare's Guest Tools are also similar to virtio.

Note that much of the traffic is concentrated on the host OS CPU—more explicitly, on the virtualized internal bridges. Therefore, the host CPU must be able to perform adequately as the device scales.

Understanding SR-IOV Usage

When a physical device is virtualized, both physical network interface card (NIC) interfaces and external physical switches as well as the virtual NIC interfaces and internal virtual switches coexist. So when the isolated virtual machines (VMs) or containers in the device, each with their own memory and disk space and CPU cycles, attempt to communicate with each other, the multiple ports, MAC addresses, and IP addresses in use pose a challenge.

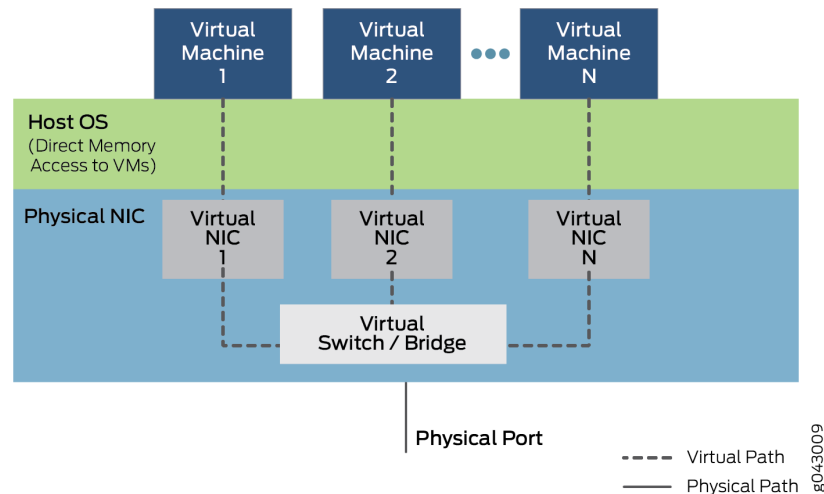
SR-IOV extends the concept of virtualized functions right down to the physical *NIC*. The single physical card is divided into up to 16 partitions per physical NIC port that correspond to the virtual functions running at the higher layers. Communication between these virtual functions are handled the same way that communications between devices with individual *NIC*s are usually handled: with a bridge. SR-IOV includes a set of standard methods for creating, deleting, enumerating, and querying the SR-IOV NIC switch, as well as the standard parameters that can be set.

The *single-root* part of SR-IOV refers to the fact that there is really only one *primary* piece of the *NIC* controlling all operations. An SR-IOV-enabled *NIC* is just a standard Ethernet port providing the same physical *bit-by-bit function* of any network card.

However, the SR-IOV also provides several virtual functions, which are accomplished by simple queues to handle input and output tasks. Each VNF running on the device is mapped to one of these NIC partitions so that VNFs themselves have direct access to NIC hardware resources. The NIC also has a simple Layer 2 sorter function, which classifies frames into traffic queues. Packets are moved directly to and from the network virtual function to the VM's memory using direct memory access (DMA),

bypassing the hypervisor completely. The role of the NIC in the SR-IOV operation is shown in [Figure 8 on page 17](#).

Figure 8: VNF Communication Using SR-IOV



The hypervisor is still involved in the assignment of the VNFs to the virtual network functions, and in the management of the physical card, but not in the transfer of the data inside the packets. Note that VNF-to-VNF communication is performed by Virtual NIC 1, Virtual NIC 2, and Virtual NIC N. There is also a portion of the NIC (not shown) that keeps track of all the virtual functions and the sorter to shuttle traffic among the VNFs and external device ports.

Note that the ability to support SR-IOV is dependent on the platform hardware, specifically the NIC hardware, and the software of the VNFs or containers to employ DMA for data transfer. Partitionable NICs, and the internal bridging required, tend to be more expensive, because of which, their use can increase the cost on smaller devices by an appreciable amount. Rewriting VNFs and containers is not a trivial task either.

Comparing Virtio and SR-IOV

Virtio is part of the standard libvirt library of helpful virtualization functions and is normally included in most versions of Linux. Virtio adopts a software-only approach. SR-IOV requires software written in a certain way and specialized hardware, which means an increase in cost, even with a simple device.

Generally, using virtio is quick and easy. Libvirt is part of every Linux distribution and the commands to establish the bridges are well-understood. However, virtio places all of the burden of performance on the host OS, which normally bridges all the traffic between VNFs, into and out of the device.

Generally, SR-IOV can provide lower latency and lower CPU utilization—in short, almost native, non-virtual device performance. But VNF migration from one device to another is complex because the VNF is dependent on the NIC resources on one machine. Also, the forwarding state for the VNF resides in the Layer 2 switch built into the SR-IOV NIC. Because of this, forwarding is no longer quite as flexible because the rules for forwarding are coded into the hardware and cannot be changed often.

While support for virtio is nearly universal, support for SR-IOV varies by NIC hardware and platform. The Juniper Networks NFX250 Network Services Platform supports SR-IOV capabilities and allows 16 partitions on each physical NIC port.

Note that a given VNF can use either virtio or SR-IOV, or even both methods simultaneously, if supported.

NOTE: Virtio is the recommended method for establishing connection between a virtualized device and an NFV module.

RELATED DOCUMENTATION

| [Configuring JDM User Accounts and Authentication](#) | 41

JDM CLI Overview

IN THIS SECTION

- [Understanding the JDM CLI](#) | 18
- [Accessing the JDM Shell, JDM CLI, and JCP Prompts in a Disaggregated Junos OS Platform](#) | 19

Understanding the JDM CLI

Junos Device Manager (JDM) can be configured using the JDM CLI. In most cases, you are logged into the JDM CLI by default when you access a disaggregated Junos OS platform.

The JDM CLI is similar to the Junos OS CLI in look and feel. It provides the same value-added facilities as the Junos OS CLI, which include:

- Separate configuration and command modes
- Commit check
- Configuration save, restore, and rollback
- NETCONF and YANG support

The JDM CLI is based on the Junos OS CLI and follows many of its processes and procedures. Like Junos OS, the JDM CLI has an operational mode and configuration mode. You use the `configure` command to go from operational mode to configuration mode, and the `exit` command to exit configuration mode. Many operational mode commands—such as `show` and `request` commands—are available in the JDM CLI and the Junos OS CLI. Many configuration commands available in the Junos OS CLI are also available in the JDM CLI, and are often entered using the same command at the same hierarchy level.

If you are placed in the JDM shell for any reason, such as if you logged in to the disaggregated Junos OS platform as the root user, enter the `cli` command from the JDM shell prompt to get to the JDM CLI prompt:

```
root# cli
root@jdm>
```

Accessing the JDM Shell, JDM CLI, and JCP Prompts in a Disaggregated Junos OS Platform

IN THIS SECTION

- [Accessing the JDM CLI | 20](#)
- [Accessing the JDM Shell | 20](#)
- [Accessing the JCP Prompt from the JDM CLI | 20](#)
- [Accessing the Hypervisor from the JDM CLI | 21](#)
- [Accessing the ipsec-nm from the JDM CLI | 21](#)

This topic describes how to access the JDM shell, JDM CLI, and JCP prompts in a disaggregated Junos OS platform.

It contains the following sections:

Accessing the JDM CLI

You are in operational mode in the JDM CLI if you see the `@jdm>` prompt. The JDM CLI also includes the configuration prompt `@jdm#`, which can be accessed by entering the `configure` command at the JDM CLI operational mode prompt.

By default, most logins to a disaggregated Junos OS platform take the user to the operational mode in the JDM CLI prompt.

The JDM CLI prompt can also be accessed from the JDM shell.

To access the JDM CLI from the JDM shell, enter the `cli` command at the JDM shell prompt:

```
root@jdm:~# cli
root@jdm>
```

Accessing the JDM Shell

By default, you are placed in the JDM shell when you login to the console port as the root user. The JDM shell uses the `~#` prompt.

To access the JDM shell from the JDM CLI, enter the **start shell** command at the JDM CLI prompt:

```
root@jdm> start shell
jdm:~#
```

Accessing the JCP Prompt from the JDM CLI

To access the JCP prompt from the JDM CLI, enter the **ssh vjunos0** command at the JDM CLI prompt:

```
root@jdm> ssh vjunos0
The authenticity of host 'vjunos0 (192.168.1.2)' can't be established.

ECDSA key fingerprint is 18:83:1f:95:88:db:1b:75:9f:07:ce:2c:4a:45:a3:b0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vjunos0,192.168.1.2' (ECDSA) to the list of known hosts.
Password:
Last login: Thu Oct  8 09:47:42 2015
--- JUNOS 15.1I20150824_0501 built 2015-08-24 05:13:01 UTC
```

```
root@RE:0% cli
root>
```

Accessing the Hypervisor from the JDM CLI

To access the Hypervisor from the JDM CLI, enter the **ssh hypervisor** statement at the JDM CLI prompt:

```
root@jdm> ssh hypervisor
Last login: Sun Jan 18 15:01:55 2015 from jdm
```

NOTE: Only a root user can use this option.

Accessing the ipsec-nm from the JDM CLI

To access the ipsec-nm from the JDM CLI, enter the **ssh ipsec-nm** statement at the JDM CLI prompt:

```
root@jdm> ssh ipsec-nm
Last login: Sun Jan 18 15:01:55 2015 from jdm

root@ipsec-nm: % cli
root@ipsec-nm>
```

RELATED DOCUMENTATION

| [Configuring JDM User Accounts and Authentication](#) | 41

Software Upgrade Path for NFX250 Devices

[Table 1 on page 22](#) provides details of the software upgrade path for NFX250 devices.

Table 1: NFX250 Software Upgrade Path

From Junos OS Release	Target / Intermediate Junos OS Release	Recommended Junos OS Release
15.1X53-D47.4	15.1X53-D470.6	18.1R1.9 (intermediate) -> 18.4R1.8
15.1X53-D47.4	15.1X53-D497.1	
15.1X53-D47.4	18.1R1.9	18.4R1.8
15.1X53-D470.6	15.1X53-D472	18.1R1.9 (intermediate) -> 18.4R1.8
15.1X53-D470.6	18.1R1.9	18.4R1.8
15.1X53-D472	15.1X53-D473.1	18.1R1.9 (intermediate) -> 18.4R1.8
15.1X53-D472	18.1R1.9	18.4R1.8
15.1X53-D473.1	18.1R1.9	18.4R1.8
15.1X53-D497.1	18.1R1.9	18.4R1.8
17.3R1.10	18.1R1.9	18.4R1.8
17.4R3.4	18.1R1.9	18.4R1.8
18.1R1.9	-	18.4R1.8
18.2R1.9	-	18.4R1.8
18.3R1.9	-	18.4R1.8

Junos OS Releases Supported on NFX Series Hardware

The [Table 2 on page 23](#) provides details of Junos OS software releases supported on the NFX Series devices.

NOTE: Support for Linux bridge mode on NFX250 devices ended in Junos OS Release 18.4.

NOTE: Support for nfx-2 software architecture on NFX250 devices ended in Junos OS Release 19.1R1.

Table 2: Supported Junos OS Releases on NFX Series Devices

NFX Series Platform	Supported Junos OS Release	Software Package	Software Downloads Page
NFX150	18.1R1 or later	nfx-3 jinstall-host-nfx-3-x86-64- <i><release-number></i> -secure-signed.tgz install-media-host-usb-nfx-3-x86-64- <i><release-number></i> -secure.img	NFX150 Software Download Page
NFX250	15.1X53-D45, 15.1X53-D47, 15.1X53-D470, and 15.1X53-D471	nfx-2 jinstall-host-nfx-2-flex-x86-64- <i><release-number></i> -secure-signed.tgz install-media-host-usb-nfx-2-flex-x86-64- <i><release-number></i> -secure.img	NFX250 Software Download Page

Table 2: Supported Junos OS Releases on NFX Series Devices *(Continued)*

NFX Series Platform	Supported Junos OS Release	Software Package	Software Downloads Page
	17.2R1 through 19.1R1		
	19.1 R1 or later	nfx-3 jinstall-host-nfx-3-x86-64-<release-number>-secure-signed.tgz install-media-host-usb-nfx-3-x86-64-<release-number>-secure.img	NFX250 Software Download Page
NFX350	19.4 R1 or later	nfx-3 jinstall-host-nfx-3-x86-64-<release-number>-secure-signed.tgz install-media-host-usb-nfx-3-x86-64-<release-number>-secure.img	NFX350 Software Download Page

RELATED DOCUMENTATION

[NFX250 Overview](#) | 2

2

CHAPTER

Installation and Configuration

Performing Initial Software Configuration on an NFX250 Device | 26

Installing Software on NFX250 Devices | 29

Configuring JDM User Accounts and Authentication | 41

YANG files on NFX250 Devices | 44

Synchronizing Time Using NTP | 48

Configuring Management Interfaces for JDM | 49

Configuring Remote Access to JDM | 58

Configuring Enhanced Orchestration and Hugepages | 60

Managing Virtual Network Functions Using JDM | 64

Configuring Service Chaining Using JDM | 109

ADSL2 and ADSL2+ Interfaces on NFX250 Devices | 126

VDSL2 Interfaces on NFX250 Devices | 131

Performing Initial Software Configuration on an NFX250 Device

Before you begin connecting and configuring an NFX250 device, set the following parameter values on the console server or PC:

- Baud Rate—9600
- Flow Control—None
- Data—8
- Parity—None
- Stop Bits—1
- DCD State—Disregard

You must perform the initial configuration of the NFX250 device through the console port using the Juniper Device Manager (JDM) command-line interface (CLI).

NOTE: Note that there are changes in the CLI commands. The CLI configuration commands for Release 15.1X53-D40 may not be applicable for this release.

To connect and configure the device from the console:

1. Connect the console port to a laptop or PC using the supplied RJ-45 cable and RJ-45 to DB-9 adapter. The console (**CON**) port is located on the management panel of the device.

NOTE: See the procedure after Step 11 for details on the Network Service Orchestrator process.

2. The Juniper Device Manager (JDM) command-line interface (CLI) displays; log in as **root**. There is no password. If the software booted before you connected to the console port, you might need to press the Enter key for the prompt to appear.

```
login: root
```

3. Start the CLI.

```
root@jdm% cli
```

4. Enter configuration mode.

```
root@jdm> configure
```

5. Add a password to the root administration user account.

```
[edit]
root@jdm# set system root-authentication plain-text-password
New password: password
Retype new password: password
```

6. (Optional) Configure the name of the device. If the name includes spaces, enclose the name in quotation marks (" ").

```
[edit]
root@jdm# set system host-name host-name
```

7. Configure the IP address and prefix length for the device management interface.

```
[edit]
root@jdm# # set interfaces jmgmt0 unit 0 family inet address address/prefix length
```

jmgmt0 is the out-of-band management network interface in JDM.

To configure an IPV6 address, run the root@jdm# set interfaces jmgmt0 unit 0 family inet6 address *v6_address*.

NOTE: jmgmt0 is located on front panel port of the NFX250 device.

8. Configure the default gateway.

```
[edit]
root@jdm# set routing-options static route default next-hop address
```

9. Commit the configuration to activate it on the device.

```
[edit]  
root@jdm# commit
```

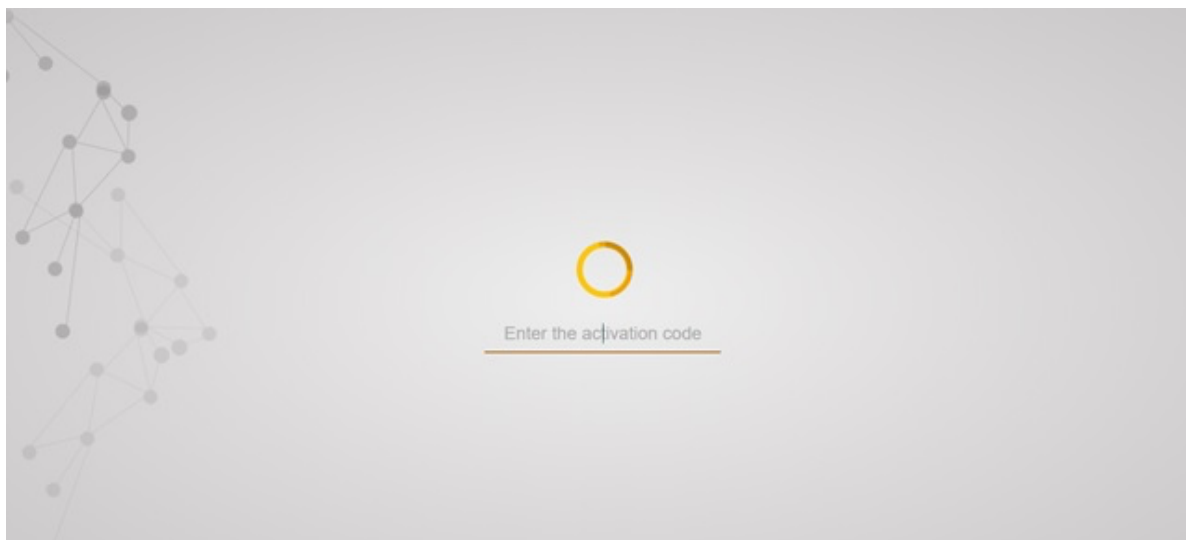
If Network Service Orchestrator module is configured, this client connects to the Network Activator as soon as the device is switched on, and provisions the initial configuration and the latest software image and, if the image on the NFX250 device is not the latest.

Network Activator is responsible for the bare-minimum bootstrapping of the NFX250. After successful configuration and software upgrade, the device reboots and the Network Activator configuration is removed.

To complete the configuration of the Network Service Orchestrator module process:

1. Connect to any front panel WAN port (see [Figure 10 on page 29](#)).
2. Open web browser and enter the IP address 10.10.10.1.
3. Enter the authentication code in the Web page that is displayed.

Figure 9: Network Service Orchestrator module

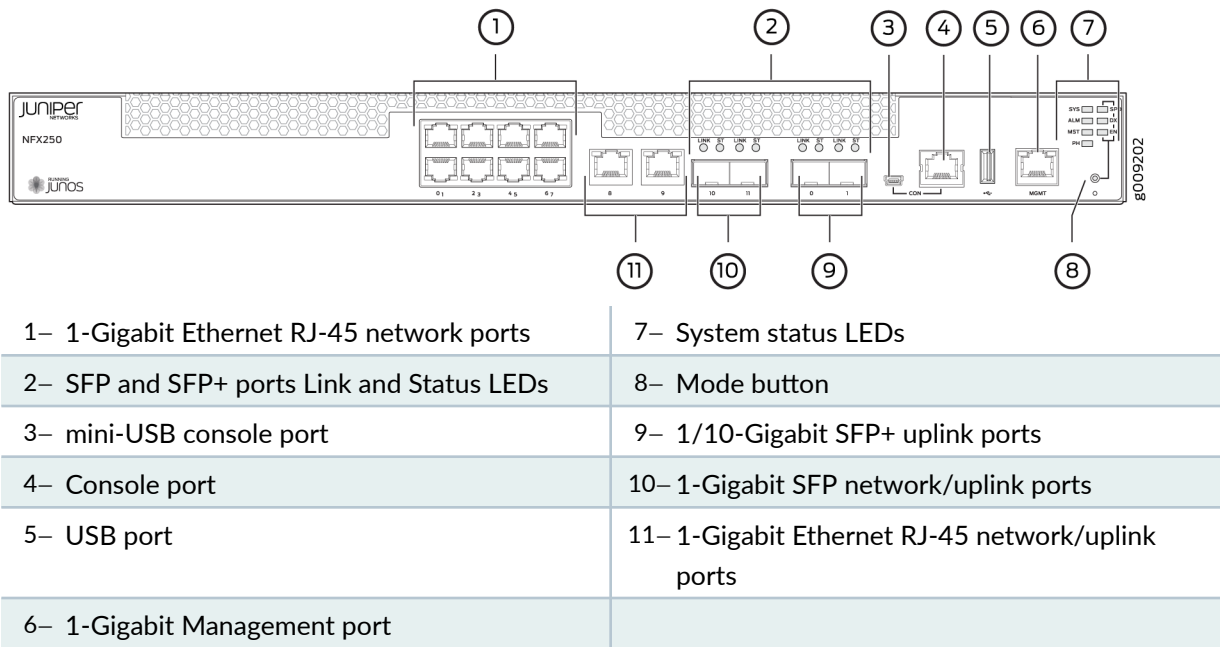


Once the process is complete, a confirmation message is displayed. Click Logs to display details of the bootstrapping process. Refer to [Captive Portal Log Messages](#) for the list of log messages that are displayed.

NOTE: You can also use the CLI to provide the authentication code:

```
root@jdm> test phone-home server-authentication-code code
```

Figure 10: NFX250 Front Panel Components



RELATED DOCUMENTATION

[NFX250 Overview](#) | 2

Installing Software on NFX250 Devices

IN THIS SECTION

- [Downloading and Installing Software](#) | 30
- [Software Installation on NFX250 Network Services Platform](#) | 34

- Upgrading an Image on the Disaggregated Junos OS Platform | 35
- Reverting the System to the Factory-Default Configuration | 40
- Rebooting the System | 40

Downloading and Installing Software

IN THIS SECTION

- Downloading Software | 30

Downloading Software

IN THIS SECTION

- Downloading Software by Using a Browser | 30
- Installing Software by Using the Command-Line Interface | 32

You can download the software package you need in one of two ways:

Downloading Software by Using a Browser

You download the software package you need from the Juniper Networks Support website at <https://www.juniper.net/support/>.

NOTE: To access the download section, you must have a service contract and an access account. If you need help obtaining an account, complete the registration form at the Juniper Networks website: <https://userregistration.juniper.net/entitlement/setupAccountInfo.do>.

To download the software image:

1. From your browser, go to <https://www.juniper.net/support/>.

The Online Support (CSC) page opens.

2. Click the **Download Software** link.

The Download Software page appears.

3. Select the software package that you want to download. You can select software that supports a specific platform or technology.
4. On the page that appears, click the **Software** tab and select the Junos OS installation package to download.
5. Log in with your username and password.
6. On the Download Software page that appears, the following options are available:
 - If you want to download the software on your local host, click the **CLICK HERE** link and save the file to your system. If you want to place the file on a remote system, you must make sure that the file can be accessible by the router, switch, or services gateway by using HTTP, FTP, or SCP. Proceed with the installation. See "[Installing Software by Using the Command-Line Interface](#)" on [page 32](#) for more details.
 - If you want to download the software on your device, use the following procedure to download and install the software on the device.
 - a. Click **Copy** to copy the generated URL generated to the clipboard.

NOTE: The URL string generated remains active only for 15 minutes.

- b. Log in to your device.
- c. In operational mode, enter the file copy *"URL" destination* command.

In the command, paste the copied URL string (for *URL*) and then enter */var/tmp* (as the destination on your hard disk).

Example:

```
user@host> file copy "https://cdn.juniper.net/software/ittest/software_target/agileEcotTest/
Dev_Binary_Build.tar?
SM_USER=user1=1507622971_dce164fa854b4a27550c254eef950dd8" /var/tmp
```


NOTE: Ensure that the URL string is enclosed within quotation marks. Also ensure that there is sufficient free space available on the device.

The software image is downloaded on your device.

- d. Install the software by using the `request system software add package-name` command.

Example:

```
user@host> request system software add /var/tmp/ junos-install-mx-x86-32-17.3R1.10.tgz
```

Your software is installed on the device.

SEE ALSO

| [Junos Platforms - Download Software](#)

Installing Software by Using the Command-Line Interface

Download the software package you need from the Juniper Networks Support website at <https://www.juniper.net/support/>, and place the package on a local system. You can then transfer the downloaded package to the device using either the router or switch command-line interface, or the local system command-line interface.

NOTE: To access the download section, you must have a service contract and an access account. If you need help obtaining an account, complete the registration form at the Juniper Networks website: <https://userregistration.juniper.net/entitlement/setupAccountInfo.do>.

Before you transfer the software package, ensure that the FTP service is enabled on the device. Enable the FTP service using the `set system services ftp` command:

```
user@host# set system services ftp
```

To transfer the software package using the device command-line interface:

1. From the router or switch command line, initiate an FTP session with the local system (host) where the package is located by using the `ftp` command:

```
user@host> ftp host
```

host is the hostname or address of the local system.

2. Log in with your customer support-supplied username and password:

```
User Name: username  
331 Password required for username.  
Password: password
```

After your credentials are validated, the FTP session opens.

3. Navigate to the software package location on the local system, and transfer the package by using the `get` command:

```
user@host> get installation-package
```

Following is an example of an *installation-package* name: **junos-install-mx-x86-32-17.3R1.10.tgz**

4. Close the FTP session by using the `bye` command:

```
user@host> bye  
Goodbye
```

To transfer the package by using the local system command-line interface:

1. From the local system command line, initiate an FTP session with the device using the `ftp` command:

```
user@host> ftp host
```

host is the hostname or address of the router or switch.

2. Log in with your customer support-supplied username and password:

```
User Name: username  
331 Password required for username.  
Password: password
```

After your credentials are validated, the FTP session opens.

3. Navigate to the software package location on the local system, and transfer the package by using the `put` command:

```
user@host> put installation-package
```

Following is an example of an *installation-package* name: **junos-install-mx-x86-32-17.3R1.10.tgz**

4. Close the FTP session by using the `bye` command:

```
user@host> bye
Goodbye
```

SEE ALSO

[Junos Platforms - Download Software](#)

Software Installation on NFX250 Network Services Platform

This topic lists the commands to be used for installing a software package and upgrading an image on NFX250 Network Services Platform and rebooting the NFX250 platform. It also lists the commands to be used for formatting and reverting the system to factory state.

To install a new package on the NFX250 Network Services Platform:

```
[edit]
user@jdm> request system software add package [reboot]
```

Reboot is an option to reboot the device after installing the new software package.

Replace *package* with the following path:

For a software package in a local directory on the platform—**/var/tmp/package.tgz**

To reboot the platform:

```
[edit]  
user@jdm> request system reboot
```

To format the system by deleting all user data, configuration details, and reinstall current software on the NFX250 Network Services platform:

```
user@jdm> request system zeroize
```

To format the system by deleting all user data, configuration details, and to upgrade the software on the NFX250 Network Services platform:

```
user@jdm> request system software add package clean-install
```

NOTE: The zeroize and clean-install commands work only for primary installation and do not work for backup installation.



CAUTION: The zeroize and clean-install commands might remove all user installed software packages, VNF files of the user, and so on. After completing these operations, you must fetch these information and reinstall the software. You might require a console access to configure the basic remote network access if the system is in factory state.

Upgrading an Image on the Disaggregated Junos OS Platform

- To upgrade the images of JCP, JDM, and the host OS on the disaggregated Junos OS platform:

```
user@jdm>request system software add jinstall reboot
```

For example:

```

user@jdm>request system software add /var/tmp/jinstall-nfx-2-flex-15.1X53-D40.3.secure-domestic-
signed.tgz reboot | no-more
System software upgrade in progress, please wait...
Pushing Junos image package to the host...
Installing /var/tmp/install-media-nfx-2-junos-15.1X53-D45.3.secure.tgz
Extracting the package ...
total 1191772
-rw-r--r-- 1 20607 758 313873261 Nov 22 09:18 jinstall-nfx-2-junos-15.1X53-D45.3.secure-linux.tgz
-rw-r--r-- 1 20607 758 906487459 Nov 22 09:18 jinstall-nfx-2-junos-15.1X53-D45.3.secure-app.tgz

=====
Host OS upgrade is FORCED
Current Host kernel version : 3.14.61-rt58-WR7.0.0.13_ovp
Package Host kernel version : 3.14.61-rt58-WR7.0.0.13_ovp
Current Host version       : 3.0.2
Package Host version       : 3.0.2
Min host version required for applications: 2.2.0
=====

Validate linux image...
upgrade_platform: -----
upgrade_platform: Parameters passed:
upgrade_platform: silent=0
upgrade_platform: package=/var/tmp/tmp.LKb5WwiFu8junos_cli_upg/jinstall-nfx-2-junos-15.1X53-
D45.3.secure-linux.tgz
upgrade_platform: clean install=0
upgrade_platform: on primary  =0
upgrade_platform: clean upgrade=0
upgrade_platform: Need reboot after staging=1
upgrade_platform: -----
upgrade_platform:
upgrade_platform: Checking input /var/tmp/tmp.LKb5WwiFu8junos_cli_upg/jinstall-nfx-2-junos-15.1X53-
D45.3.secure-linux.tgz ...
upgrade_platform: Input package /var/tmp/tmp.LKb5WwiFu8junos_cli_upg/jinstall-nfx-2-junos-15.1X53-
D45.3.secure-linux.tgz is valid.
Secure Boot is enforced.
ALLOW:usr/secureboot/grub/BOOTX64.EFI
ALLOW:boot/bzImage-intel-x86-64.bin
ALLOW:boot/initramfs.cpio.gz
Setting up Junos host applications for installation ...

```

```

Installing Host OS ...
upgrade_platform: -----
upgrade_platform: Parameters passed:
upgrade_platform: silent=0
upgrade_platform: package=/var/tmp/jinstall-nfx-2-junos-15.1X53-D45.3.secure-linux.tgz
upgrade_platform: clean install=0
upgrade_platform: on primary    =0
upgrade_platform: clean upgrade=0
upgrade_platform: Need reboot after staging=0
upgrade_platform: -----
upgrade_platform:
upgrade_platform: Checking input /var/tmp/jinstall-nfx-2-junos-15.1X53-D45.3.secure-linux.tgz ...
upgrade_platform: Input package /var/tmp/jinstall-nfx-2-junos-15.1X53-D45.3.secure-linux.tgz is valid.
Secure Boot is enforced.
ALLOW:usr/secureboot/grub/BOOTX64.EFI
ALLOW:boot/bzImage-intel-x86-64.bin
ALLOW:boot/initramfs.cpio.gz
upgrade_platform: Staging the upgrade package - /var/tmp/jinstall-nfx-2-junos-15.1X53-D45.3.secure-
linux.tgz..
./
./bzImage-intel-x86-64.bin
./bzImage-intel-x86-64.bin.psig
./grub/
./grub/grub.conf
./grub/grub.efi
./initramfs.cpio.gz
./initramfs.cpio.gz.psig
./linux.checksum
./version.txt
./upgrade_platform
./host-version
./platform_info
./initrd.cpio.gz
bzImage-intel-x86-64.bin: OK
initramfs.cpio.gz: OK
version.txt: OK
upgrade_platform: Checksum verified and OK...
1528703 blocks
upgrade_platform: Staging of /var/tmp/jinstall-nfx-2-junos-15.1X53-D45.3.secure-linux.tgz completed
upgrade_platform: System need *REBOOT* to complete the upgrade
upgrade_platform: Run upgrade_platform with option -r | --rollback to rollback the upgrade

Host OS upgrade staged. Reboot the system to complete installation!

```

```

Rebooting ...
System going down for reboot in 30 seconds...
System reboot in progress...
Shutting down virtual-machines...
Waiting for virtual-machines to shutdown, retry = 0
Waiting for virtual-machines to shutdown, retry = 1
Waiting for virtual-machines to shutdown, retry = 2
No virtual-machines active now.
Rebooting the system...
INIT: Sending processes the TERM signal

{master:0}
root@nfx250-m-p2a-sys11-jdm> Stopping OpenBSD Secure Shell server: sshdstopped /usr/sbin/sshd (pid 4169)
.
Unmount Junos cgroup... Done
Stopping atd: OK
Stopping domain name service: named.
Unmounting cgroups...Done
Stopping system message bus: dbus.
stopping DNS forwarder and DHCP server: dnsmasq... stopped /usr/bin/dnsmasq (pid 9449 9448)
done.
Stopping docker:
Stopping HOSTAP Daemon: no /usr/sbin/hostapd found; none killed
hostapd.
Shutting down irqbalance: no irqbalance found; none killed
done
Stopping ntpd: done
stopping rsyslogd ... done
Stopping internet superserver: xinetd.

Waiting for sanlock to stop: Success

Clearing ebtables rule sets: filter nat broute done. ok
Stopping crond: OK
Stopping S.M.A.R.T. daemon: smartd.
Stopping network management services: snmpd snmptrapd libvirtMib_subagent.
* Stopping virtualization library daemon: libvirtd
Deconfiguring network interfaces... done.
Stopping tcsh: OK
Stopinit: Failed to release D-Bus name: Did not receive a reply. Possible causes include: the remote
application did not send a reply,
the message bus security policy blocked the reply, the reply timeout expired, or the network connection

```

```

was broken.
ping redis-server...
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
Rebooting... RE-FPGA-DRV: reboot notifier called with 0x0001
RE-FPGA-DRV: Please standby while rebooting.
.
..
..
..
.

Booting from Flash A

FPGA Reset Reason = 0x80

Primary BIOS version CBDE_SFP_00.21_01.01

Total Memory Size = 16GB

Checking Primary BIOS code integrity...Passed!
Press Esc for boot options
ME is in normal operational state

Booting HDD00.1 (StorFly VSFBM6CC100G-JUN)...

Secure boot is enforced
Welcome to GRUB!

Secure Grub2 Diskboo

```


Reverting the System to the Factory-Default Configuration

To revert the system to factory-default configuration:

```
user@jdm# load factory-default  
warning: activating factory configuration
```

Rebooting the System

To reboot the system:

```
user@jdm>request system reboot
```

For example:

```
user@jdm>request system reboot  
Reboot the system ? [yes,no] (no) yes  
System reboot operation started, please wait...  
System going down for reboot in 30 seconds...  
System reboot in progress...  
Shutting down virtual-machines...  
  
. . . .
```

NOTE: The time taken to reboot the system depends on the number of active VNFs. The system is rebooted only after all the active VNFs are shut down.

RELATED DOCUMENTATION

[NFX250 Overview](#) | 2

Configuring JDM User Accounts and Authentication

IN THIS SECTION

- [JDM User Accounts Overview | 41](#)
- [Configuring JDM User Accounts and Authentication | 42](#)

JDM User Accounts Overview

IN THIS SECTION

- [Root Account | 41](#)
- [Auto Login for Junos VNFs | 42](#)
- [Other User Accounts | 42](#)
- [User Authentication | 42](#)

On a disaggregated Junos OS platform, all computing elements are separate compute entities, and their user accounts and passwords are managed separately. For example, JDM user accounts, including the root user account, are completely separate from the Junos VM user accounts.

Root Account

In the factory-default configuration, the JDM is set up with a root user account. However, there is no password set for the account. You must configure a root password as part of the initial configuration. If the initial configuration of the platform is performed through the phone home feature, the configuration must contain the root password setting. Until you configure a root password, you cannot access some of the user prompts and you cannot commit a configuration by using the JDM CLI.

You can set the root password only from the JDM CLI. You cannot set or change the root password from the JDM shell. The JDM root password is automatically propagated to the JDM shell.

Auto Login for Junos VNFs

When Junos VNFs such as JCP and vSRX are present on an NFX250 device, JDM auto login account allows you to login to Junos VNF without a password.

To configure auto login to JDM:

```
root@jdm> request setup jdm-auto-login
```

To login to Junos VNF from JDM:

```
root@jdm> ssh jdm-sysuser@vjunos0
```

Other User Accounts

You can create user accounts other than the root account in the JDM. To do this, you must use the JDM CLI. You cannot use the JDM shell to create user accounts.

The JDM supports the same features for user accounts as does Junos OS. That is, the JDM supports login classes, custom password requirements, limits on the number of login attempts, and so on.

User Authentication

The JDM supports two of the three methods for user authentication that Junos OS supports: local password authentication and TACACS+ authentication. It does not support RADIUS authentication.

Configuring JDM User Accounts and Authentication

You create user accounts and configure authentication for those accounts in JDM the same way you do in Junos OS. This topic provides some brief guidance on how to configure user accounts and authentication. For more details, consult the Junos OS documentation.

- To set the JDM root password:

```
root@jdm# set system root-authentication plain-text-password
```

You must use the JDM CLI to set the root password. You cannot set the root password using the JDM shell.

- To create a new JDM user account:

```
root@jdm# set system login user user-name class class-name authentication plain-text-password
```

You cannot create JDM user accounts from the JDM shell.

- To configure SSH keys for a user to enable SSH without a password:

```
root@jdm# set system login user user-name load-key-file URL-to-ssh-key-file
```

- To configure TACAS+ authentication for user accounts:

```
root@jdm# set system tacplus-server server-address secret password
```

NOTE: TACACS+ is used to support SSH authentication, and once configured, TACACS+ configuration is applicable for both, JDM and host SSH authentication. On the host, TACACS+ is used to authenticate SSH requests only for the root account and when requested from outside the device.

Optionally, you can specify the TACACS+ authentication server port number and the timeout period. To do so:

```
root@jdm# set system tacplus-server server-address port port-number
```

```
root@jdm# set system tacplus-server server-address timeout period
```

NOTE: By default, the TACACS+ port number is set to 49, and the timeout period is set to 5 seconds.

You must also configure the user name along with the class of the user locally on JDM:

```
root@jdm# set system login user user-name
root@jdm# set system login user user-name class super-user
```

- To allow users to log in to the NFX250 device as a root user:

```
root@jdm# root-login allow
```

- To prevent users from logging in to the NFX250 device as a root user:

```
root@jdm# root-login deny
```

- To allow users to log in to the NFX250 device as a root user through an authentication method (for example, RSA authentication) that does not require a password:

```
root@jdm# root-login deny-password
```

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

YANG files on NFX250 Devices

IN THIS SECTION

- [Understanding YANG on NFX250 Devices | 44](#)
- [Generating YANG Files | 45](#)

Understanding YANG on NFX250 Devices

YANG is a standards-based, extensible data modeling language that is used to model the configuration and operational state data, remote procedure calls (RPCs), and server event notifications of network devices. The NETMOD working group in the IETF originally designed YANG to model network

management data and to provide a standard for the content layer of the Network Configuration Protocol (NETCONF) model. However, YANG is protocol independent, and YANG data models can be used independent of the transport or RPC protocol and can be converted into any encoding format supported by the network configuration protocol.

Juniper Networks provides YANG modules that define the Junos OS configuration hierarchy and operational commands and Junos OS YANG extensions. You can generate the modules on the device running Junos OS.

YANG uses a C-like syntax, a hierarchical organization of data, and provides a set of built-in types as well as the capability to define derived types. YANG stresses readability, and it provides modularity and flexibility through the use of modules and submodules and reusable types and node groups.

A YANG module defines a single data model and determines the encoding for that data. A YANG module defines a data model through its data, and the hierarchical organization of and constraints on that data. A module can be a complete, standalone entity, or it can reference definitions in other modules and submodules as well as augment other data models with additional nodes.

A YANG module defines not only the syntax but also the semantics of the data. It explicitly defines relationships between and constraints on the data. This enables you to create syntactically correct configuration data that meets constraint requirements and enables you to validate the data against the model before uploading it and committing it on a device.

YANG uses modules to define configuration and state data, notifications, and RPCs for network operations in a manner similar to how the Structure of Management Information (SMI) uses MIBs to model data for SNMP operations. However, YANG has the benefit of being able to distinguish between operational and configuration data. YANG maintains compatibility with SNMP's SMI version 2 (SMIv2), and you can use libsmi to translate SMIv2 MIB modules into YANG modules and vice versa. Additionally, when you cannot use a YANG parser, you can translate YANG modules into YANG Independent Notation (YIN), which is an equivalent XML syntax that can be read by XML parsers and XSLT scripts.

For information about YANG, see [RFC 6020](#), *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, and related RFCs.

For more information, see [YANG Modules Overview](#), [Using Juniper Networks YANG Modules](#), and [show system schema](#).

Generating YANG Files

You can generate YANG files for JDM and JCP on NFX250 devices.

To generate YANG files for JDM:

1. Log in to the NFX device using SSH or console:

```
login: root
```

2. Start the CLI:

```
root@jdm:~# cli
{master:0}
root@jdm>
```

3. Create a temporary directory to store the generated YANG files:

```
{master:0}
root@jdm> file make-directory /var/third-party/jdm_yang
{master:0}
root@jdm> file list /var/third-party/jdm_yang
/var/third-party/jdm_yang:
{master:0}
root@jdm>
```

4. Generate YANG files for JDM:

```
{master:0}
root@jdm> show system schema module all format yang output-directory /var/third-party/jdm_yang
```

5. Verify whether YANG files are generated in the specified target directory:

```
{master:0}
root@jdm> file list /var/third-party/jdm_yang
/var/third-party/jdm_yang:

jdm-conf-root@2020-01-01.yang
jdm-rpc-bridges@2020-01-01.yang
jdm-rpc-clear@2020-01-01.yang
jdm-rpc-cli@2020-01-01.yang
jdm-rpc-connections@2020-01-01.yang
...Output truncated...
```

To generate YANG files for JCP:

1. Log in to the JCP CLI:

```
{master:0}
root@jdm> ssh jdm-sysuser@vjunos0
{master:0}
jdm-sysuser>
```

2. Create a temporary directory to store the generated YANG files:

```
{master:0}
jdm-sysuser> file make-directory /var/tmp/jcp_yang
{master:0}
jdm-sysuser> file list /var/tmp/jcp_yang
/var/tmp/jcp_yang:
{master:0}
jdm-sysuser>
```

3. Generate YANG files for JCP:

```
{master:0}
root> show system schema module all format yang output-directory /var/tmp/jcp_yang
```

4. Verify whether YANG files are generated in the specified target directory:

```
{master:0}
root> file list /var/tmp/jcp_yang
/var/tmp/jcp_yang:

junos-common-types@2020-01-01.yang
junos-qfx-conf-access-profile@2020-01-01.yang
junos-qfx-conf-access@2020-01-01.yang
junos-qfx-conf-accounting-options@2020-01-01.yang
junos-qfx-conf-applications@2020-01-01.yang
...Output truncated...
```

5. Copy the generated YANG files from JCP to JDM:

- a. Exit the JCP CLI to return to the JDM CLI:

```
{master:0}
jdm-sysuser> exit
Connection to vjunos0 closed.
{master:0}
root@jdm>
```

- b. Navigate to JDM shell and copy the generated JCP YANG files from JCP to JDM:

```
{master:0}
root@jdm> start shell
jdm:~# scp -r jdm-sysuser@vjunos0:/var/tmp/jcp_yang /var/third-party/
junos-qfx-conf-access-profile@2020-01-01.yang          100% 923      0.9KB/s   00:00
junos-qfx-conf-access@2020-01-01.yang                 100% 179KB 178.9KB/s 00:00
junos-qfx-conf-accounting-options@2020-01-01.yang     100% 20KB  20.4KB/s  00:00
junos-qfx-conf-applications@2020-01-01.yang          100% 11KB  10.6KB/s  00:00
...Output truncated...
jdm:~#
```

6. Copy the generated JDM and JCP YANG files from the NFX device to the YANG based tools or orchestrators by using the `scp` or `file copy` command.

Synchronizing Time Using NTP

You can synchronize time on the following components of the NFX platform using Network Time Protocol (NTP):

- Junos Control Plane (JCP) - JCP runs the NTP server, and synchronizes time using the external NTP servers that are configured. JCP acts as the NTP server for the host.
- Host (hypervisor) - The host runs the NTP server and client, and synchronizes time using the Junos Control Plane (JCP) NTP server through JDM. The host, in turn, acts as the NTP server for the virtual network functions (VNFs).
- VNF - This is optional. VNFs run the NTP client, and synchronize the time using either JCP, hypervisor, or any external server that is configured.

To set the date and time using NTP:

1. Configure the NTP server and set the date on JCP:

```
root# set system ntp server <ip-address>
root# exit
root> set date ntp
```

Commit the configuration.

2. Once the NTP server has been configured on JCP, you can set the date and time on the host using JDM:

```
root@jdm> set date ntp
```

3. (Optional) Set the local time zone to match the location of the device and to present the time in the correct local format. Universal Coordinated Time (UTC) is the default. Many administrators prefer to keep all their devices configured to use the UTC time zone. This approach has the benefit of allowing you to easily compare the time stamps of logs and other events across a network of devices in many different time zones.

```
root@jdm# set time zone time-zone
```

NOTE: If the VNFs are not running the NTP clients, reboot the system to synchronize the date and time on all VNFs.

RELATED DOCUMENTATION

| [Understanding Virtual Network Functions](#) | 65

Configuring Management Interfaces for JDM

IN THIS SECTION

● [JDM Management Interfaces Overview](#) | 50

- [Configuring the Out-of-Band Management Interface for JDM | 52](#)
- [Configuring the In-Band Management Interface for JDM | 53](#)
- [Configuring the Out-of-Band Management Interface for Hypervisor | 57](#)

JDM Management Interfaces Overview

IN THIS SECTION

- [Console Interface | 50](#)
- [Out-of-Band Management Interface | 50](#)
- [In-Band Management Interface | 51](#)

You can access JDM through the system console, a dedicated out-of-band management interface, or an in-band management interface.

Console Interface

On disaggregated Junos OS platforms that host Juniper Device Manager (JDM), you are placed in either the JDM shell prompt or the JDM CLI when you first connect to a device using the device console port. You can access the JDM CLI or other VMs, including the Junos Control Plane (JCP) CLI that is used to manage Junos OS software, from this prompt. See *Accessing the JDM Shell, JDM CLI, and JCP Prompts in a Disaggregated Junos OS Platform*.

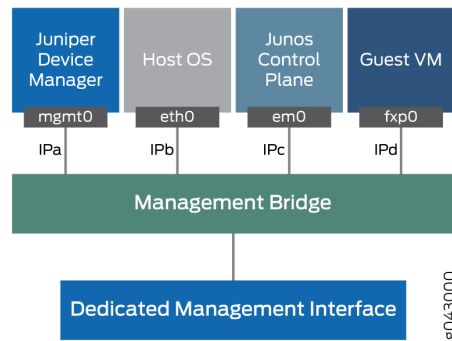
Out-of-Band Management Interface

The JDM out-of-band management interface is named `jmgmt0`. The `jmgmt0` interface is directly connected to the dedicated Ethernet management port on the disaggregated Junos OS platform.

The `jmgmt0` interface in a disaggregated Junos OS platform is analogous to the `em0`, `me0`, or `fxp0` interfaces on a Juniper Networks switch or a router running traditional Junos OS software. To use `jmgmt0` as a management port, you must configure a logical interface (`jmgmt0.0`) on it with a valid IP address. You can then connect to the management interface over the network using utilities such as SSH or Telnet. SNMP can be used on the management interface to gather statistics.

The dedicated Ethernet management port on the platform is shared by other compute entities. For example, JCP uses the dedicated Ethernet management port for its out-of-band management interface, em0, as shown in [Figure 11 on page 51](#).

Figure 11: Out-of-band Management Interface



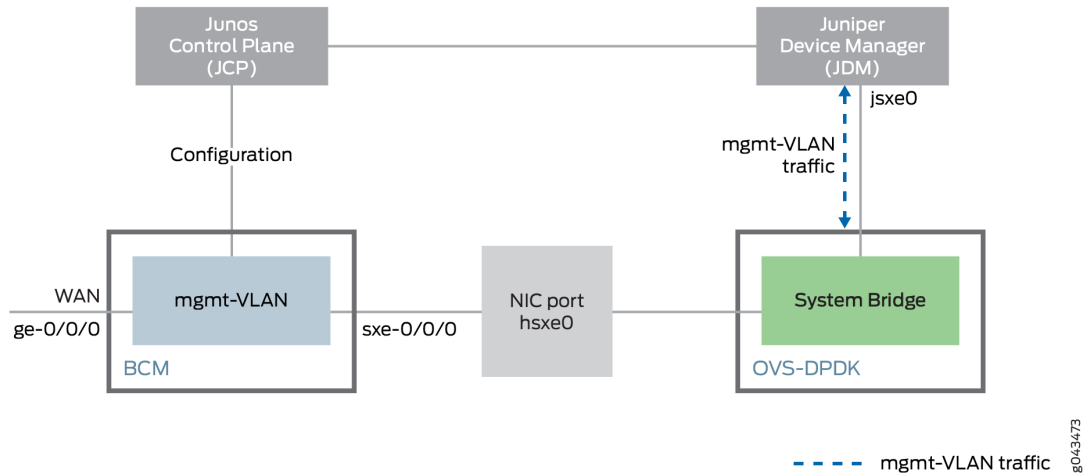
In-Band Management Interface

JDM has an interface—jsxe0—that can be used as in-band management interface. Unlike the out-of-band management interface jmgmt0, this interface is not directly connected to a physical port. You must connect jsxe0 to a physical interface through VLAN bridging—that is, you must configure both the physical interface and jsxe0 to be in the same management VLAN.

[Figure 12 on page 52](#) illustrates how a network port is bridged to jsxe0. In this figure, ge-0/0/0 is the network port being used for in-band management. Interface sxe-0/0/0 is a JCP interface. Both ge-0/0/0

and sxe-0/0/0 are managed by JCP, and are configured in JCP to be part of the management VLAN, mgmt-vlan. JDM interface jsxe0 is also configured to part of the mgmt-vlan.

Figure 12: In-Band Management Interface Network



Configuring the Out-of-Band Management Interface for JDM

IN THIS SECTION

- [Configuring the Out-of-Band Management Interface with IPv4 Addressing for JDM | 52](#)
- [Configuring the Out-of-Band Management Interface with IPv6 Addressing for JDM | 53](#)

This topic discusses how to configure an out-of-band management interface for JDM in a disaggregated Junos OS platform.

On a disaggregated Junos OS platform, the out-of-band management interface for JDM is named mgmt0. The mgmt0 interface has a direct connection to the dedicated Ethernet management port on the front panel of the device.

Configuring the Out-of-Band Management Interface with IPv4 Addressing for JDM

To configure the management interface with IPv4 addressing:

1. Configure the logical interface and the IP address:

```
root@jdm# set interfaces jmgmt0 unit 0 family inet address ipv4-address/mask
```

2. Set the default route:

```
root@jdm# set routing-options static route 0.0.0.0/0 nexthop ipv4-address
```

Configuring the Out-of-Band Management Interface with IPv6 Addressing for JDM

To configure the management interface with IPv6 addressing:

1. Configure the logical interface and the IP address:

```
root@jdm# set interfaces jmgmt0 unit 0 family inet6 address ipv6-address/mask
```

2. Set the default route:

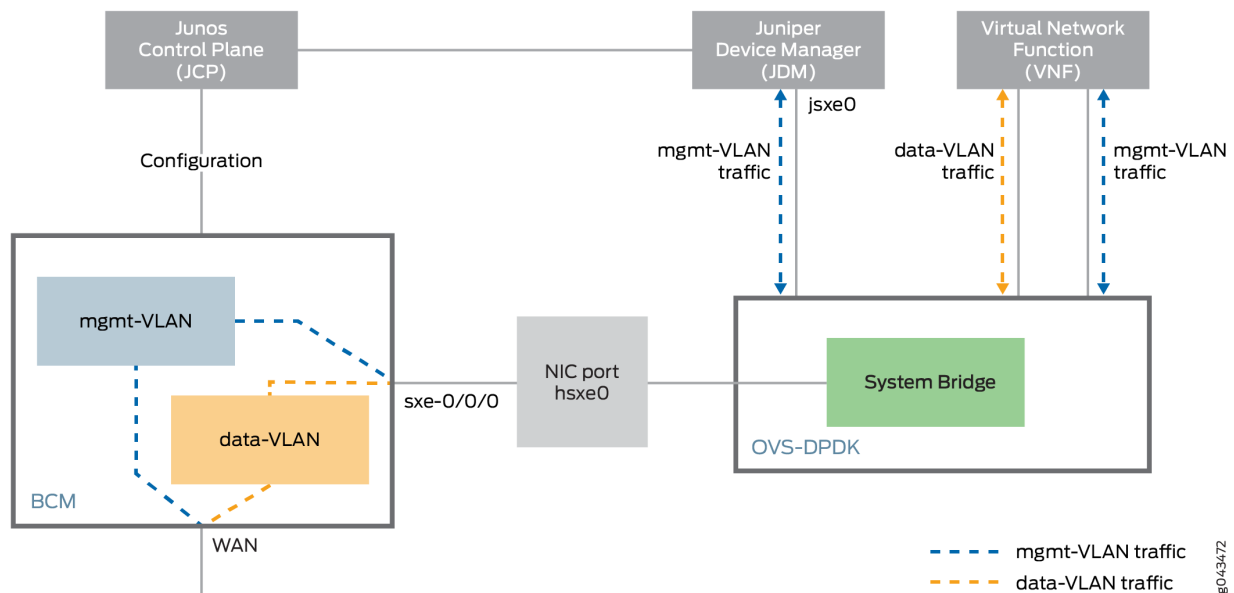
```
root@jdm# set routing-options static route ::/0 nexthop ipv6-address
```

Configuring the In-Band Management Interface for JDM

JDM provides an internal interface—`jsxe0`—that can be used for in-band management. This internal interface is not directly connected to a physical interface. You must link `jsxe0` to a physical interface through VLAN bridging—that is, you must configure both the physical interface and `jsxe0` to be in the same management VLAN. See [Figure 13 on page 54](#).

JCP, and not JDM manages the physical network interfaces and the service interfaces; therefore, you must first configure the `sxe-0/0/0` and `sxe-0/0/1` internal interfaces using the JCP CLI before you can manage the `jsxe0` interface using the JDM CLI.

Figure 13: In-Band Management Interface Example



NOTE: Choose the management VLAN ID to ensure that only the management traffic is directed to JDM.

To configure `jsxe0` as an in-band management interface:

1. Log in to the JCP CLI and enter configuration mode:

```
root@jdm> ssh jdm-sysuser@vjunos0

root@RE:0% cli
{master:0}
{master:0}[edit]
```

2. Configure the physical network port as a trunk port:

```
[edit]
root# set interfaces interface-name unit 0 family ethernet-switching interface-mode trunk
```

For example:

```
[edit]
root# set interfaces ge-0/0/0 unit 0 family ethernet-switching interface-mode trunk
```

3. Configure a JCP service port as a trunk port:

```
root# set interfaces service-interface-name unit 0 family ethernet-switching interface-mode trunk
```

For example:

```
[edit]
root# set interfaces sxe-0/0/0 unit 0 family ethernet-switching interface-mode trunk
```

4. Configure the management VLAN and add the physical network interface and the service interface as members of the VLAN:.

```
[edit]
root@jcp# set vlans mgmt-vlan vlan-id vlan-id
root@jcp# set interfaces service-interface-name unit 0 family ethernet-switching vlan members mgmt-vlan
root@jcp# set interfaces interface-name unit 0 family ethernet-switching vlan members mgmt-vlan
```

For example:

```
[edit]
root@jcp# set vlans inband_mgmt vlan-id 4040
root@jcp# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members inband_mgmt
root@jcp# set interfaces ge-0/0/9 unit 0 family ethernet-switching vlan members inband_mgmt
```

5. Exit the JCP CLI to return to the JDM CLI (*hostname@jdm>* prompt):

```
[edit]
root# exit
```



```

Exiting configuration mode
root> exit
root% exit
logout
Connection to vjunos0 closed.
root@jdm>

```

6. Configure the jsxe0 interface as a trunk interface with membership in the management VLAN, and configure the management IP address on the interface:

```

root@jdm# set interfaces jsxe0 vlan-tagging
root@jdm# set interfaces jsxe0 unit logical-unit-number vlan-id mgmt-vlan-id family inet address mgmt-
ip-address/prefix-length

```

Example: Configuring the In-Band Management Interface on JDM

```

root@jdm> ssh jdm-sysuser@vjunos0

root@RE:0% cli
{master:0}
{master:0}[edit]
root@jdm# set interfaces jsxe0 vlan-tagging
root@jdm# set interfaces jsxe0 unit 4040 vlan-id 4040
root@jdm# set interfaces jsxe0 unit 4040 family inet address 10.20.30.40/24
root@jdm# set interfaces jsxe0 unit 4040 family inet6 address 2001::10:20:30:40/64

```

Example: Configuring the In-Band Management Interface on JCP

```

root@jdm> ssh jdm-sysuser@vjunos0

root@RE:0% cli
{master:0}
{master:0}[edit]
root@jcp# set vlans inband_mgmt vlan-id 4040
root@jcp# set interfaces ge-0/0/9 description "WAN interface to PE"
root@jcp# set interfaces ge-0/0/9.0 family ethernet-switching interface-mode trunk
root@jcp# set interfaces ge-0/0/9.0 family ethernet-switching vlan members inband_mgmt
root@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching interface-mode trunk
root@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching vlan members inband_mgmt

```

Configuring the Out-of-Band Management Interface for Hypervisor

IN THIS SECTION

- [Configuring the Out-of-Band Management Interface with IPv4 Addressing for Hypervisor | 57](#)
- [Configuring the Out-of-Band Management Interface with IPv6 Addressing for Hypervisor | 57](#)

This topic discusses how to configure an out-of-band management interface for Hypervisor in a disaggregated Junos OS platform.

On a disaggregated Junos OS platform, the out-of-band management interface for Hypervisor is named eth0br. The eth0br interface has a direct connection to the dedicated Ethernet management port on the front panel of the device.

Configuring the Out-of-Band Management Interface with IPv4 Addressing for Hypervisor

To configure the management interface with IPv4 addressing:

1. Configure the logical interface and the IP address:

```
root@jdm# set host-os interfaces eth0br unit 0 family inet address ipv4-address/mask
```

2. Set the default route:

```
root@jdm# set host-os routing-options static route 0.0.0.0/0 next-hop gateway-ipv4-address
```

Configuring the Out-of-Band Management Interface with IPv6 Addressing for Hypervisor

To configure the management interface with IPv6 addressing:

1. Configure the logical interface and the IP address:

```
root@jdm# set host-os interfaces eth0br unit 0 family inet6 address ipv6-address/mask
```

2. Set the default route:

```
root@jdm# set host-os routing-options static route ::/0 next-hop gateway-ipv6-address
```

RELATED DOCUMENTATION

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

Configuring Remote Access to JDM

IN THIS SECTION

- [Configuring SSH Service and NETCONF-Over-SSH Connections for Remote Access to the Disaggregated Junos OS Platform | 58](#)
- [Configuring HTTP Access to the Disaggregated Junos OS Platform | 59](#)
- [Configuring HTTPS Access to the Disaggregated Junos OS Platform | 59](#)

Configuring SSH Service and NETCONF-Over-SSH Connections for Remote Access to the Disaggregated Junos OS Platform

You can configure the disaggregated Junos OS platform to accept NETCONF sessions over SSH as an access service.

To do so:

1. Access the system services SSH configuration:

```
[edit]
user@jdm#set system services netconf ssh
```

2. Configure the TCP port used for NETCONF-over-SSH connections:

```
[edit system services netconf ssh]  
user@jdm#user@jdm# port port-number
```

The configured port only accepts NETCONF-over-SSH connections. Regular SSH connections to the port are ignored.

Configuring HTTP Access to the Disaggregated Junos OS Platform

You can configure HTTP access to the disaggregated Junos OS platform.

To do so:

1. Access the system services HTTP configuration:

```
[edit]  
user@jdm#edit system services http
```

2. Set the HTTP port for incoming connections:

```
[edit system services http]  
user@jdm# set port port
```

Configuring HTTPS Access to the Disaggregated Junos OS Platform

You can configure HTTPS access to the disaggregated Junos OS platform.

To do so:

1. Access the HTTPS configuration:

```
[edit]  
user@jdm#edit system services https
```

2. Set the HTTPS port for incoming connections:

```
[edit system services https]
user@jdm# set port port-number
```

RELATED DOCUMENTATION

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

[Configuring JDM User Accounts and Authentication | 41](#)

Configuring Enhanced Orchestration and Hugepages

IN THIS SECTION

- [Enhanced Orchestration | 60](#)
- [Hugepages | 61](#)

Enhanced Orchestration

Enhanced orchestration mode enables you to easily manage VNFs and service chains without requiring the VNF XML descriptor files. By default, this mode is ON and this is the recommended mode.

To enable enhanced orchestration:

```
[edit system services]
user@jdm# set enhanced-orchestration
```

NOTE: Ensure that you reboot the system after enabling the enhanced orchestration mode.

Hugepages

IN THIS SECTION

- [Pre-allocating Hugepages | 62](#)
- [Allocating Hugepages for a VNF | 64](#)
- [Troubleshooting Hugepages | 64](#)

Hugepages in NFX250 devices are contiguous memory blocks of 2 MB and 1 GB sizes, which are used for virtual memory management. Due to memory fragmentation, the system might not have sufficient memory to assign the required amount of hugepages when a new VNF is launched. This might result in the VNF either failing to launch or switching to the shutdown state during configuration. Hence, to launch VNFs on NFX250 devices, you must pre-allocate memory using hugepages before configuring the VNFs.

NOTE: Hugepages must be enabled for all VNFs that use OVS for service chaining.

When pre-allocating hugepages, ensure that there is sufficient memory for system use as insufficient memory might cause the system to become unresponsive. The system, which consists of the JCP, JDM, and Hypervisor, requires around 6 to 7 GB of memory. Only the remaining memory can be used by the VNFs.

To view the current state of memory and hugepages available, issue the following command at the JDM CLI prompt:

```
user@jdm> show system visibility memory
```

The default configuration comes with one hugepage of 1 GB size that is used by the OVS, and hugepages of 2 MB size for system use.

[Table 3 on page 62](#) lists the maximum hugepage memory that can be reserved for the various NFX250 models.

Table 3: Recommended Hugepage Memory for the NFX250 Devices

Model	Memory	Maximum Hugepage Memory (GB)	Maximum Hugepage Memory (GB) for CSO-SDWAN
NFX250-S1	16 GB	8	-
NFX250-S2, NFX250-S1E	32 GB	24	13
NFX250-LS1	16 GB	8	-

Pre-allocating Hugepages

To pre-allocate hugepages, issue the following command at the JDM CLI prompt:

```
user@jdm# set system memory hugepages page-size page-size page-count page-count
```

where:

- *page-size* can be one of the following values:
 - 1024 for 1 GB hugepage
 - 2 for 2 MB hugepage
- *page-count* is the number of hugepages

The *page-size* and *page-count* values depend on the size and total number of hugepages required by all the VNFs that will be launched in the system. The values are also limited by the available memory on the device.

We recommend that you reboot the system after configuring hugepages to pre-allocate hugepages during bootup.

NOTE: By default, OVS uses 1 GB hugepage from the allocated set of hugepages.

The following sample output shows a configuration with both 1 GB and 2 MB hugepages configured. A total of 8 hugepages of 1 GB size are configured, in which one hugepage is used by the OVS and the

remaining seven hugepages can be used for the VNFs. There are 376 hugepages of 2 MB size that can be used for VNFs. The remaining hugepages (500-376=124) are reserved for system use.

```

-----
Virtual Memory:
-----
Total      (KiB): 15949136
Used       (KiB): 12690344
Available  (KiB): 4687452
Free       (KiB): 3258792
Percent Used    : 70.60

Swap Memory:
-----
Total (KiB): 0
Used  (KiB): 0
Free  (KiB): 0
Percent Used: 0.00

Memory Limits:
-----
User VNFs Total Memory Limit (KiB): 9437184

Huge Pages:
-----
Total 1GiB Huge Pages:      8
Free 1GiB Huge Pages:      7
Configured 1GiB Huge Pages: 8
Total 2MiB Huge Pages:     500
Free 2MiB Huge Pages:      376
Configured 2MiB Huge Pages: 500

```

Note that the value of Configured 1 GB Huge Pages includes the hugepage that the system allocates for the OVS.

NOTE: On NFX250 NG devices, the value of the Total 1GiB Huge Pages will be one more than the total hugepages reserved through the CLI (Configured 1GiB Huge Pages).

Allocating Hugepages for a VNF

To allocate hugepages for a VNF, issue the following command:

```
user@jdm# set virtual-network-functions vnf-name memory features hugepages [page-size page-size]
```

Troubleshooting Hugepages

If the number of hugepages configured results in insufficient memory for system use, the following error message appears when you access the JDM CLI after rebooting the device:

```
fatal error - could not reserve address space in "getmem.c"
```

To reconfigure the hugepages:

1. Establish an SSH connection to the hypervisor:

```
user@jdm# ssh hypervisor
```

2. Determine the number of hugepages configured:

```
cat /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

3. Reset the number of hugepages to 0:

```
echo 0 > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

4. Reboot the device.

5. Access the JDM CLI and reconfigure the number of hugepages.

Managing Virtual Network Functions Using JDM

IN THIS SECTION

- [Understanding Virtual Network Functions | 65](#)
- [Prerequisites to Onboard Virtual Network Functions on NFX250 Devices | 67](#)
- [Managing the VNF Life Cycle | 68](#)
- [Creating the vSRX VNF on the NFX250 Platform | 82](#)

- [Configuring the vMX Virtual Router as a VNF on NFX250 | 85](#)
- [Virtual Route Reflector on NFX250 Overview | 87](#)
- [Configuring vRR as a VNF on NFX250 | 89](#)
- [Configuring Cross-connect | 105](#)
- [Configuring Analyzer VNF and Port-mirroring | 107](#)

Understanding Virtual Network Functions

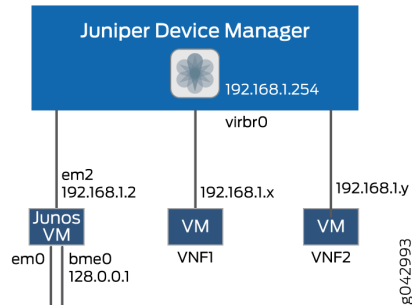
Virtualized network functions (VNFs) include all virtual entities that can be launched and managed from the Juniper Device Manager (JDM). Currently, virtual machines (VMs) are the only VNF type that is supported.

There are several components in a JDM environment:

- **JDM**—Manages the life cycle for all service VMs. JDM also provides a CLI with configuration persistence or the ability to use NETCONF for scripting and automation.
- **Primary Junos OS VM**—A *system VM* that is the primary virtual device. This VM is always present when the system is running.
- **Other Junos OS VMs**—These VMs are *service VMs* and are activated dynamically by an external controller. A typical example of this type of VM is a vSRX instance.
- **Third-party VNFs**—JDM supports the creation and management of third-party VMs such as Ubuntu Linux VMs.

The JDM architecture provides an internal network that connects all VMs to the JDM as shown in [Figure 14 on page 66](#).

Figure 14: Network Connections Between JDM and the VMs



The JDM can reach any VNF using an internal network (192.0.2.1/24).

NOTE: Up to Junos OS Release 15.1X53-D470, the liveliness IP is in 192.168.1.0/24 subnet. In all later Junos OS Releases, the liveliness IP is in 192.0.2.0/24 subnet.

A VNF can own or share management ports and NIC ports in the system.

All VMs run in isolation and a state change in one VM does not affect another VM. When the system restarts, the service VMs are brought online as specified in the persistent configuration file. When you gracefully shut down the system, all VMs including the Junos VMs are shut down.

[Table 4 on page 66](#) provides a glossary of commonly used VNF acronyms and terms.

Table 4: VNF Glossary

Term	Definition
JCP	Junos Control Plane (also known as the primary Junos OS VM)
JDM	Juniper Device Manager
NFV	Network Functions Virtualization

Table 4: VNF Glossary (*Continued*)

Term	Definition
VM	Virtual Machine
VNF	Virtualized Network Function

Prerequisites to Onboard Virtual Network Functions on NFX250 Devices

IN THIS SECTION

- [Prerequisites for VNFs | 67](#)

You can onboard and manage Juniper VNFs and third-party VNFs on NFX devices through the Junos Control Plane (JCP).

The number of VNFs that you can onboard on the device depends on the availability of system resources such as the number of CPUs and system memory.

Before you onboard the VNFs, it is recommended to check the available system resources such as CPUs, memory, and storage for VNFs. For more information, see ["Managing the VNF Life Cycle " on page 68](#).

Prerequisites for VNFs

To instantiate VNFs, the NFX devices support:

- KVM based hypervisor deployment
- OVS or Virtio interface drivers
- raw or qcow2 VNF file types
- (Optional) SR-IOV
- (Optional) CD-ROM and USB configuration drives
- (Optional) Hugepages for memory requirements

Managing the VNF Life Cycle

IN THIS SECTION

- [Planning Resources for a VNF | 68](#)
- [Managing the VNF Image | 70](#)
- [Preparing the Bootstrap Configuration | 71](#)
- [Launching a VNF | 71](#)
- [Allocating Resources for a VNF | 72](#)
- [Managing VNF States | 78](#)
- [Managing VNF MAC Addresses | 79](#)
- [Managing MTU | 79](#)
- [Accessing a VNF from JDM | 80](#)
- [Viewing List of VNFs | 81](#)
- [Displaying the VNF Details | 81](#)
- [Deleting a VNF | 82](#)

You can use the JDM CLI to manage the VNF. Additionally, *libvirt* software offers extensive virtualization features. To ensure that you are not limited by the CLI, JDM provides an option to operate VNF using an XML descriptor file. Network Configuration Protocol (NETCONF) supports all VNF operations. Multiple VNFs can co-exist in a system and you can configure multiple VNFs using either an XML descriptor file or an image.

NOTE: Ensure that VNF resources that are specified in the XML descriptor file do not exceed the available system resources.

This topic covers the life-cycle management of a VNF.

Planning Resources for a VNF

IN THIS SECTION

- [Purpose | 69](#)

Purpose

Before launching a VNF, it is important to check the system inventory and confirm that the resources required by the VNF are available. The VNF must be designed and configured properly so that its resource requirements do not exceed the available capacity of the system.

NOTE:

- The output of the `show system inventory` command displays only the current snapshot of system resource usage. When you start a VNF, the resource usage might be less than what was available when you installed the VNF package.
- Before starting a VNF, you must check the system resource usage.

NOTE: Some of the physical CPUs are reserved by the system. Except for the following physical CPUs, all others are available for user-defined VNFs:

Table 2 provides the list of physical CPUs that are reserved for NFX250-LS1.

Table 5: Physical CPU Allocation for NFX250-LS1

CPU Core	Allocation
0	Host, JDM, and JCP
4	Host bridge
7	IPSec

Table 3 provides the list of physical CPUs that are reserved for NFX250-S1, NFX250-S2, and NFX250-S1E devices.

Table 6: Physical CPU Allocation for NFX250

CPU Core	Allocation
0	Host, JDM, and JCP

Table 6: Physical CPU Allocation for NFX250 (Continued)

CPU Core	Allocation
6	Host bridge
7	IPSec

For more information, see the following:

- [show system inventory hardware cpu](#)
- [show system inventory hardware memory](#)
- [show system inventory hardware network](#)
- [show system inventory hardware storage](#)
- [show system inventory software vnf](#)
- [show system visibility jcp](#)
- [show system visibility jdm](#)
- [show system visibility jcp](#)
- [show system visibility vnf](#)
- [show system visibility storage](#)
- [show system visibility network](#)
- [show system visibility memory](#)
- [show system visibility cpu](#)
- [show system visibility host](#)

Managing the VNF Image

To load a VNF image on the device from a remote location, use the `file-copy` command. Alternatively, you can use the `NETCONF` command `file-put`, to load a VNF image.

NOTE: You must save the VNF image in the `/var/third-party/images` directory.

Preparing the Bootstrap Configuration

You can bootstrap a VNF by attaching either a CD or a USB storage device that contains a bootstrap-config ISO file.

A bootstrap configuration file must contain an initial configuration that allows the VNF to be accessible from an external controller, and accepts SSH, HTTP, or HTTPS connections from an external controller for further runtime configurations.

An ISO disk image must be created offline for the bootstrap configuration file as follows:

```
user@jdm>request genisoimage bootstrap-config-filename iso-filename
```

Launching a VNF

You can launch a VNF by configuring the VNF name, and specifying either the path to an XML descriptor file or to an image.

While launching a VNF with an image, two VNF interfaces are added by default. These interfaces are required for management and internal network. For those two interfaces, the target Peripheral Component Interconnect (PCI) addresses, such as 0000:00:03:0 and 0000:00:04:0 are reserved.

To launch a VNF using an XML descriptor file:

```
user@jdm# set virtual-network-functions vnf-name init-descriptor file-path
user@jdm# commit
```

To launch a VNF using an image:

```
user@jdm# set virtual-network-functions vnf-name image file-path
user@jdm# commit
```

To specify a UUID for the VNF:

```
user@jdm# set virtual-network-functions vnf-name [uuid vnf-uuid]
```

uuid is an optional parameter, and it is recommended to allow the system to allocate a UUID for the VNF.

NOTE:

- You cannot change the init-descriptor or image configuration after saving and committing the init-descriptor and image configuration. To change the init-descriptor or image for a VNF, you must delete and create a VNF again.
- Commit checks are applicable only for VNF configurations that are based on image specification through JDM CLI, and not for VNF configurations that are based on init-descriptor XML file.

NOTE: For creating VNFs using image files, ensure the following:

- You must use unique files for image, disk, USB that are used within a VNF or across VNFs except for an iso9660 type file, which can be attached to multiple VNFs.
- A file specified as image in raw format should be a block device with a partition table and a boot partition.
- A file specified as image in qcow2 format should be a valid qcow2 file.

Allocating Resources for a VNF

IN THIS SECTION

- [Specifying CPU for VNF | 72](#)
- [Allocating Memory for a VNF | 73](#)
- [Configuring VNF Storage Devices | 74](#)
- [Configuring VNF Interfaces and VLANs | 76](#)

This topic covers the process of allocating various resources to a VNF.

Specifying CPU for VNF

To specify the number of virtual CPUs that are required for a VNF, type the following command:

```
user@jdm# set virtual-network-functions vnf-name virtual-cpu count 1-4
```

To pin a virtual CPU to a physical CPU, type the following command:

```
user@jdm# set virtual-network-functions vnf-name virtual-cpu vcpu-number physical-cpu pcpu-number
```

The physical CPU number can either be a number or a range. By default, a VNF is allocated with one virtual CPU that is not pinned to any physical CPU.

NOTE: You cannot change the CPU configuration of a VNF when the VNF is in **running** state. Restart the VNF for changes to take effect.

To enable hardware-virtualization or hardware-acceleration for VNF CPUs, type the following command:

```
user@jdm# set virtual-network-functions vnf-name virtual-cpu features hardware-virtualization
```

Allocating Memory for a VNF

To specify the maximum primary memory that the VNF can use, enter the following command:

```
user@jdm# set virtual-network-functions vnf-name memory size size
```

By default, 1 GB of memory is allocated to a VNF.

NOTE: You cannot change the memory configuration of a VNF if the VNF is in **running** state. Restart the VNF for changes to take effect.

To allocate hugepages for a VNF, type the following command:

```
user@jdm# set virtual-network-functions vnf-name memory features hugepages [page-size page-size]
```

page-size is an optional parameter. Possible values are 1024 for a page size of 1GB and 2 for a page size of 2 MB. Default value is 1024 hugepages.

NOTE: Configuring hugepages is recommended only if the enhanced orchestration mode is enabled. If the enhanced orchestration mode is disabled and if VNF requires hugepages, the VNF XML descriptor file should contain the XML tag with hugepages configuration.

NOTE: For VNFs that are created using image files, there is a maximum limit of the total memory that can be configured for all user-defined VNFs including memory based on hugepages and memory not based on hugepages.

[Table 7 on page 74](#) lists the maximum hugepage memory that can be reserved for the various NFX250 models.

Table 7: Recommended Hugepage Memory for the NFX250 Devices

Model	Memory	Maximum Hugepage Memory (GB)	Maximum Hugepage Memory (GB) for CSO-SDWAN
NFX250-S1	16 GB	8	-
NFX250-S1E	16 GB	24	13
NFX250-S2	32 GB	24	13
NFX250-LS1	16 GB	8	-

Configuring VNF Storage Devices

To add a virtual CD or to update the source file of a virtual CD, enter the following command:

```
user@jdm# set virtual-network-functions vnf-name storage device-name type cdrom source file file-name
```

To add a virtual USB storage device, enter the following command:

```
user@jdm# set virtual-network-functions vnf-name storage device-name type usb source file file-name
```

To attach an additional hard disk, enter the following command:

```
user@jdm# set virtual-network-functions vnf-name storage device-name type disk [bus-type virtio | ide]
[file-type raw | qcow2] source file file-name
```

To delete a virtual CD, USB storage device, or a hard disk from the VNF, enter the following command:

```
user@jdm# delete virtual-network-functions vnf-name storage device-name
```

NOTE:

- After attaching or detaching a CD from a VNF, you must restart the device for changes to take effect. The CD detach operation fails if the device is in use within the VNF.
- VNF supports one virtual CD, one virtual USB storage device, and multiple virtual hard disks.
- You can update the source file in a CD or USB storage device while the VNF is in **running** state.
- You must save the source file in the **/var/third-party** directory and the file must have read and write permission for all users.

NOTE: For VNFs created using image files, ensure the following:

- A file specified as a hard disk in raw format should be a block device with a partition table.
- A file specified as a hard disk in qcow2 format should be a valid qcow2 file.
- A file specified as USB should be a block device with a partition table, or an iso9660 type file.
- A file specified as CD-ROM should be a block device of type iso9660.
- If a VNF has an image specified with bus-type=ide, it should not have any device attached with name hda.

- If a VNF has an image specified with `bus-type=virtio`, then it should not have any device attached with name `vda`.

Configuring VNF Interfaces and VLANs

You can create a VNF interface and attach it to a physical NIC port, management interface, or VLANs.

1. To attach a VNF interface to a physical interface by using the SR-IOV virtual function:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name mapping physical-interface-name virtual-function [vlan-id vlan-id]
```

`vlan-id` is optional and it is the port VLAN ID.

2. To create VLAN:

```
user@jdm# set host-os vlans vlan-name vlan-id vlan-id
```

3. To attach a VNF interface to a VLAN:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name mapping vlan members list-of-vlans [mode trunk|access]
```

NOTE:

- The interfaces attached to the VNF are persistent across VNF restarts.
- If the VNF supports hot-plugging, you can attach the interfaces when the VNF is in **running** state. Otherwise, add the interfaces, and then restart the VNF.
- To map interfaces to VLAN, you must enable the **memory features hugepages** command option.
- You cannot change the mapping of VNF interface when the VNF is in **running** state.

4. To map virtual interfaces with physical interfaces:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name mapping peer-interfaces
```

Mapping of virtual interfaces and physical interfaces (ge-0/0/n and xe-0/0/n) makes sure that the state of the virtual interface matches with the state of the physical interface to which it is mapped. For example, if a physical interface is down and virtual interface is up, the virtual interface will be brought down within 5 seconds of detection. One or more virtual interface can be mapped to one or more physical interfaces.

5. To connect VNF interfaces to the internal management network:

NOTE: Before connecting VNF interfaces to the internal management network, you must configure the VNFs by using the `set virtual-network-function vnf-name no_default_interface` command.

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name management internal
user@jdm# set virtual-network-functions vnf-name interfaces interface-name management out-of-band
```

Any of the VNF interfaces including eth0 and eth1 can have internal or out-of-band attribute management. However, only one VNF interface out of all interfaces that are connected can have either out-of-band-management or internal-management. You cannot specify both the attribute values to the same VNF interface. For example, eth5 can have management internal while eth0 can have management out-of-band.

6. To specify the target PCI address for a VNF interface:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name pci-address target-pci-address
```

You can use the target PCI address to rename or reorganize interfaces within the VNF.

For example, a Linux-based VNF can use udev rules within the VNF to name the interface based on the PCI address.

NOTE:

- The target PCI-address string should be in the following format:

0000:00:<slot>:0, which are the values for domain:bus:slot:function. The slot should be different for each VNF interface. The values for domain, bus, and function should be zero.
- You cannot change the target PCI-address of VNF interface when the VNF is in **running** state.

7. To delete a VNF interface:

```
user@jdm# delete virtual-network-functions vnf-name interfaces interface-name
user@jdm# commit
```

NOTE:

- To delete an interface, you must stop the VNF, delete the interface, and start the VNF.
- After attaching or detaching a virtual function, you must restart the VNF for changes to take effect.
- eth0 and eth1 are reserved for default VNF interfaces that are connected to the internal network and out-of-band management network. Therefore, the configurable VNF interface names start from eth2.
- Within a VNF, the interface names can be different, based on guest OS naming convention. VNF interfaces that are configured in JDM might not appear in the same order within the VNF.
- You must use the target PCI addresses to map to the VNF interfaces that are configured in JDM and name them accordingly.

Managing VNF States

By default, the VNF is autostarted on committing the VNF config.

1. To disable an autostart of a VNF on a VNF config commit:

```
user@jdm# set virtual-network-functions vnf-name no-autostart
```

2. To manually start a VNF:

```
user@jdm> request virtual-network-functions vnf-name start
```

3. To stop a VNF:

```
user@jdm> request virtual-network-functions vnf-name stop
```

4. To restart a VNF:

```
user@jdm> request virtual-network-functions vnf-name restart
```

Managing VNF MAC Addresses

VNF interfaces that are defined, either using a CLI or specified in an init-descriptor XML file, are assigned a globally-unique and persistent MAC address. A common pool of 64 MAC addresses is used to assign MAC addresses. You can configure a MAC address other than that available in the common pool, and this address will not be overwritten.

1. To configure a specific MAC address for a VNF interface:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name mac-address mac-address
```

2. To delete the MAC address configuration of a VNF interface:

```
user@jdm# delete virtual-network-functions vnf-name interfaces interface-name mac-address mac-address
```

NOTE:

- To delete or modify the MAC address of a VNF interface, you must stop the VNF, make the necessary changes, and then start the VNF.
- The MAC address specified for a VNF interface can be either a system MAC address or a user-defined MAC address.
- The MAC address specified from the system MAC address pool must be unique for VNF interfaces.

Managing MTU

The maximum transmission unit (MTU) is the largest data unit that can be forwarded without fragmentation. You can configure either 1500 bytes or 2048 bytes as the MTU size. The default MTU value is 1500 bytes.

NOTE: MTU configuration is supported only on VLAN interfaces.

1. To configure MTU on a VNF interface:

```
user@jdm# set virtual-network-functions vnf-name interfaces interface-name mtu size
```

NOTE: You must restart the VNF after configuring MTU if the VNF does not support hot-plugging functionality.

2. To delete MTU of a VNF interface:

```
user@jdm# delete virtual-network-functions vnf-name interfaces interface-name mtu
```

NOTE: After the deletion of MTU, the MTU of VNF interface is reset to 1500 bytes.

NOTE:

- MTU size can be either 1500 bytes or 2048 bytes.
- The maximum number of VLAN interfaces on the OVS that can be configured in the system is 20.
- The maximum size of the MTU for a VNF interface is 2048 bytes.

Accessing a VNF from JDM

You can access a VNF from JDM using either SSH or a VNF console.

1. To access a VNF using SSH:

```
user@jdm> ssh vnf-name
```

2. To access a VNF using a virtual console:

```
user@jdm> request virtual-network-functions vnf-name console
```

NOTE:

- Use **ctrl-]** to exit the virtual console.
- Do not use Telnet session to run the command.

Viewing List of VNFs

To view the list of VNFs:

```
user@jdm> show virtual-network-functions
```

ID	Name	State	Liveliness
3	vjunos0	running	alive
-	vsrx	shut off	down

The **Liveliness** output field of a VNF indicates whether the IP address of the VNF is reachable or not reachable from JDM. The default IP address of the liveliness bridge 192.0.2.1/24.

Displaying the VNF Details

To display VNF details:

```
user@jdm> show virtual-network-functions vnf-name
```

Virtual Machine Information

```
-----
Name:          vsrx
IP Address:    192.0.2.4
Status:        Running
Liveliness:    Up
VCPUs:         1
Maximum Memory: 2000896
Used Memory:   2000896
```

Virtual Machine Block Devices

```
-----
```

Target	Source
hda	/var/third-party/images/vsrx/media-srx-ffp-vsrx-vmdisk-15.1-2015-05-29_X_151_X49.qcow2
hdf	/var/third-party/test.iso

Deleting a VNF

To delete a VNF:

```
user@jdm# delete virtual-network-functions vnf-name
```

NOTE: The VNF image remains in the disk even after you delete the VNF.

Creating the vSRX VNF on the NFX250 Platform

vSRX is a virtual security appliance that provides security and networking services in virtualized private or public cloud environments. It can be run as a virtual network function (VNF) on the NFX250 platform. For more details on vSRX, see the product documentation page on the Juniper Networks website at <https://www.juniper.net/>.

To activate the vSRX VNF from the Juniper Device Manager (JDM) command-line interface:

1. Allocate hugepages memory:

```
set virtual-network-functions vsrx memory size 4194304
set virtual-network-functions vsrx memory features hugepages
```

2. Define VLANs required for vSRX VNF interfaces. For example:

```
set host-os vlans v1 vlan-id 2614
set host-os vlans v2 vlan-id 2615
set host-os vlans v3 vlan-id 2714
set host-os vlans v4 vlan-id 2715
```

3. Define any glue VLANs required for the vSRX VNF interfaces. For example:

```
set host-os vlans gluebr0 vlan-id 2814
set host-os vlans gluebr1 vlan-id 2815
```

4. Define vSRX VNF with vSRX image. For example:

```
set virtual-network-functions vsrx image /var/third-party/images/vsrx.qcow2
```

5. (Optional) Create the vSRX VNF with groups that contain custom configuration. For example:

```
set virtual-network-functions vsrx apply-groups junos-vsrx
```

6. Map the vSRX VNF interfaces to VLANs or glue-VLANs. For example:

```
set virtual-network-functions vsrx interfaces eth2 mapping vlan members v1
set virtual-network-functions vsrx interfaces eth3 mapping vlan members v2
set virtual-network-functions vsrx interfaces eth4 mapping vlan members v3
set virtual-network-functions vsrx interfaces eth5 mapping vlan members v4
set virtual-network-functions vsrx interfaces eth6 mapping vlan members gluebr0
set virtual-network-functions vsrx interfaces eth7 mapping vlan members gluebr1
```

7. Specify a mode for the vSRX VNF interfaces. The interface mode can be either access or trunk mode. For example:

```
set virtual-network-functions vsrx interfaces eth2 mapping vlan mode trunk
```

8. Specify the maximum transmission unit (MTU) size for the media in bytes for vSRX VNF interfaces. MTU size can be either 1500 bytes or 2048 bytes. For example:

```
set virtual-network-functions vsrx interfaces eth2 mtu 1500
```

9. Specify the target PCI address for the VNF interface. For example:

```
set virtual-network-functions vsrx interfaces eth2 pci-address pci-address
```

10. At the CLI prompt, enter the **commit** command to activate the vSRX VNF.

```
[edit]
root# commit
```

11. Attach the ISO to vSRX as a CD-ROM device and start vSRX.

```
[edit]
root@jdm# set virtual-network-functions vsrx storage hdb type cdrom source file /var/third-party/iso/
testcd/bootstrapconf.iso
```

NOTE: If a vSRX instance is running, you must restart the instance so that the new configuration is applied from the CD-ROM.

12. (Optional) To create the vSRX VNF with a custom bootstrap configuration, create an ISO image with the configuration file **juniper.conf**.

```
[edit]
root@jdm> request genisoimage /var/third-party/iso/testcd/juniper.conf /var/third-party/iso/testcd/
bootstrapconf.iso
ISO image "/var/third-party/iso/testcd/bootstrapconf.iso" successfully generated from "/var/third-
party/iso/testcd/juniper.conf".
```

NOTE: Ensure that the configuration file is named juniper.conf.

13. Verify if the vSRX VNF initiated correctly. You can use JDM cli or Linux virsh commands to verify.

```
[edit]
root@jdm# run show virtual-network-functions
```

ID	Name	State	Liveliness
2	vjunos0	running	alive
12	vsrx	running	alive
7433	jdm	running	alive

Using the Linux virsh command

```
[edit]
root@jdm> start shell
jdm:~# virsh list
```

ID	Name	State

```

2    vjunos0          running
3    vsrx             running

```

You can see that the vSRX VNF is active.

14. SSH connection to vSRX works only if the liveliness in the show output shows the status alive, that is if bootstrap iso config was used to enable DHCP on fxp0 interface of vSRX to get the internal management IP address). If the liveliness status for vSRX VNF is down, refer to [Configuring the Internal Management IP Address of vSRX VNF](#).

To log on to the vSRX VNF, enter the command `run ssh vsrx`.

15. (Optional) Verify the vSRX VNF details.

```

root@jdm> show virtual-network-functions vsrx
Virtual Machine Information
-----
Name:          vsrx
IP Address:    192.168.1.4
Status:        Running
Liveliness:    Up
VCPUs:         2
Maximum Memory: 4000768
Used Memory:   4000768
Virtual Machine      Block Devices
-----
Target  Source
-----
hda     /var/third-party/images/vsrx.qcow2
hdf     /var/third-party/iso/testcd/bootstrapconf.iso

```

Configuring the vMX Virtual Router as a VNF on NFX250

The vMX router is a virtual version of the Juniper MX Series 5G Universal Routing Platform. To quickly migrate physical infrastructure and services, you can configure vMX as a virtual network function (VNF) on the NFX250 platform. For more details on the configuration and management of vMX, see [vMX Overview](#).

Before you configure the VNF, check the system inventory and confirm that the required resources are available. vMX as VNF must be designed and configured so that its resource requirements do not exceed the available capacity of the system. Ensure that a minimum of 20 GB space is available on NFX250.

To configure vMX as VNF on NFX250 using the Juniper Device Manager (JDM) command-line interface (CLI):

1. Download the nested image available at **vmx-nested-< release>.qcow2**.
2. Define VLANs required for the vMX VNF interfaces. For example:

```
user@host# set host-os vlans v1 vlan-id 2614
user@host# set host-os vlans v2 vlan-id 2615
```

3. Define the glue VLANs required for the vMX VNF interfaces. For example:

```
user@host# set host-os vlans gluebr0 vlan-id 2614
user@host# set host-os vlans gluebr1 vlan-id 2615
```

4. Define vMX for VNF with the vMX image. For example:
 user@host# set virtual-network-functions vmx image /var/third-party/images/vmx-nested-<release>.qcow2
5. Specify the maximum primary memory that the VNF can use. For optimal performance, it is recommended to configure with at least 5 GB memory.
 user@host# set virtual-network-functions vmx memory size < n>
6. Specify the number of cores per CPU in a virtual machine. For vMX VNF, you need a minimum of 4 virtual CPU cores.
 user@host# set virtual-network-functions vmx virtual-cpu count < n> features hardware-virtualization
7. Add an additional data drive that stores the configuration parameters.
 user@host# set virtual-network-functions vmx storage vdc type disk file-type vmx-nested-<release>.qcow2
8. Map the vMX VNF interfaces to VLANs or glue-VLANs.
 user@host# set virtual-network-functions vmx interfaces eth2 description wan0

 user@host# set virtual-network-functions vmx interfaces eth2 mapping vlan members < vlan>

 user@host# set virtual-network-functions vmx interfaces eth3 description wan1

 user@host# set virtual-network-functions vmx interfaces eth3 mapping vlan members < vlan>
9. At the CLI prompt, enter the **commit** command to activate vMX VNF.
 user@host# commit
10. Verify if the vMX VNF has been configured correctly on NFX250.

```
root@jdm# run show virtual-network-functions
```

ID	Name	State	Liveliness

3	vjunos0	running	alive
10	vmx	running	alive
11341	jdm	running	alive

If you use virsh, enter

```
root@jdm# virsh list
```

ID	Name	State

2	vjunos0	running
3	vmx	running

This shows that the vMX VNF is active.

11. Verify if the vMX VNF has been configured correctly on NFX250.

```
root@jdm# run show virtual-network-functions
```

To upgrade the vMX VNF, deactivate the VNF configuration and select the new image copied to the `/var/third-party/images/vmx-nested-< release>.qcow2` location. Reactivate the VNF configuration again.

12. For in-band management network connections, the assigned management port is fxp0. For out-of-band management, ge-0/0/0 is used, and ge-0/0/1 is used for WAN interfaces.

Virtual Route Reflector on NFX250 Overview

IN THIS SECTION

- [Benefits of vRR | 88](#)
- [Software Requirements for vRR on NFX250 | 88](#)

The virtual Route Reflector (vRR) feature allows you to implement route reflector capability using a general purpose virtual machine that can be run on a 64-bit Intel-based blade server or appliance. Because a route reflector works in the control plane, it can run in a virtualized environment. A virtual route reflector on an Intel-based blade server or appliance works the same as a route reflector on a router, providing a scalable alternative to full mesh internal BGP peering.

Starting in Junos OS Release 17.3R1, you can implement the virtual route reflector (vRR) feature on the NFX250 Network Services platform. The Juniper Networks NFX250 Network Services Platform comprises the Juniper Networks NFX250 devices, which are Juniper Network's secure, automated, software-driven customer premises equipment (CPE) devices that deliver virtualized network and security services on demand. NFX250 devices use the Junos Device Manager (JDM) for virtual machine (VM) lifecycle and device management, and for a host of other functions. The JDM CLI is similar to the Junos OS CLI in look and provides the same added-value facilities as the Junos OS CLI.

NOTE: Starting in Junos OS Release 20.1R1, both standard orchestration (SO) and enhanced orchestration (EO) modes are supported for vRR. It is recommended to instantiate vRR VNF in EO mode.

Benefits of vRR

vRR has the following benefits:

- **Scalability:** By implementing the vRR feature, you gain scalability improvements, depending on the server core hardware on which the feature runs. Also, you can implement virtual route reflectors at multiple locations in the network, which helps scale the BGP network with lower cost. The maximum routing information base (RIB) scale with IPv4 routes on NFX250 is 20 million.
- **Faster and more flexible deployment:** You install the vRR feature on an Intel server, using open source tools, which reduces your router maintenance.
- **Space savings:** Hardware-based route reflectors require central office space. You can deploy the virtual route reflector feature on any server that is available in the server infrastructure or in the data centers, which saves space.

For more information about vRR, refer [Virtual Route Reflector \(vRR\) Documentation](#).

Software Requirements for vRR on NFX250

The following software components are required to support vRR on NFX250:

- **Juniper Device Manager:** The Juniper Device Manager (JDM) is a low-footprint Linux container that supports Virtual Machine (VM) lifecycle management, device management, Network Service

Orchestrator module, service chaining, and virtual console access to VNFs including vSRX, vjunos, and now vRR as a VNF.

- **Junos Control Plane:** Junos Control Plane (JCP) is the Junos VM running on the hypervisor. You can use JCP to configure the network ports of the NFX250 device, and JCP runs by default as vjunos0 on NFX250. You can log on to JCP from JDM using the SSH service and the command-line interface (CLI) is the same as Junos.

Configuring vRR as a VNF on NFX250

IN THIS SECTION

- [Configuring vRR VNF on NFX250 in Standard Orchestration Mode | 89](#)
- [Configuring vRR VNF on NFX250 in Enhanced Orchestration Mode | 102](#)

Configuring vRR VNF on NFX250 in Standard Orchestration Mode

IN THIS SECTION

- [Configuring Junos Device Manager \(JDM\) for vRR | 89](#)
- [Verifying that the Management IP is Configured | 91](#)
- [Verifying that the Default Routes are Configured | 91](#)
- [Configuring Junos Control Plane \(JCP\) for vRR | 92](#)
- [Launching vRR | 96](#)
- [Enabling Liveliness Detection of vRR VNF from JDM | 99](#)

Configuring Junos Device Manager (JDM) for vRR

By default, the Junos Device Manager (JDM) virtual machine comes up after NFX250 is powered on. By default, enhanced orchestration mode is enabled on JDM. While configuring vRR, disable enhanced orchestration mode, remove the interfaces configuration, and reboot the NFX device.

To configure the Junos Device Manager (JDM) virtual machine for vRR, perform the following steps:

1. In configuration mode, at the [edit] hierarchy level, disable enhanced orchestration. By default, enhanced orchestration mode is enabled on JDM.

```
[edit ]
user@jdm# delete system services enhanced-orchestration
```

2. Delete interface configuration.

```
[edit ]
user@jdm# delete interfaces
```

3. Set the JDM root password.

```
[edit ]
user@jdm# set system root-authentication plain-text-password
```

4. Commit the configuration using the `commit` command and reboot the system for the configuration to take effect.

```
[edit ]
user@jdm# commit
user@jdm# run request reboot
```

5. After the system reboots, the default bridges configuration is available on JDM. Configure the JDM root password, management port IP, and add default routes.

NOTE: After the system reboots, if the groups **groups1604-configs** is not present in the configuration, include it so the default bridges configuration is available on JDM.

```
[edit ]
user@jdm# set system root-authentication encrypted-password
user@jdm# set system services ssh
user@jdm# set interface eth0 vlan-id vlan-id family inet address address
user@jdm# set route destination address next-hop address
```

Verifying that the Management IP is Configured

IN THIS SECTION

- [Purpose | 91](#)
- [Action | 91](#)

Purpose

Ensure that the management IP address has been configured accurately.

Action

From configuration mode, enter the `show interface` command.

```
user@jdm# show interface eth0
```

```
vlan-id 0 {  
  family {  
    inet {  
      address 10.48.14.18/22;  
    }  
  }  
}
```

Verifying that the Default Routes are Configured

IN THIS SECTION

- [Purpose | 92](#)
- [Action | 92](#)

Purpose

Ensure that the default routes are configured for DNS and gateway access.

Action

From configuration mode, enter the `show route` command.

```
user@jdm# show route
```

```
destination 172.16.0.0/12 next-hop 10.48.15.254;
destination 192.168.0.0/16 next-hop 10.48.15.254;
destination 207.17.136.0/24 next-hop 10.48.15.254;
destination 10.0.0.0/10 next-hop 10.48.15.254;
destination 10.64.0.0/10 next-hop 10.48.15.254;
destination 10.128.0.0/10 next-hop 10.48.15.254;
destination 10.192.0.0/11 next-hop 10.48.15.254;
destination 10.224.0.0/12 next-hop 10.48.15.254;
destination 10.240.0.0/13 next-hop 10.48.15.254;
destination 10.248.0.0/14 next-hop 10.48.15.254;
destination 10.252.0.0/15 next-hop 10.48.15.254;
destination 10.254.0.0/16 next-hop 10.48.15.254;
destination 66.129.0.0/16 next-hop 10.48.15.254;
destination 10.48.0.0/15 next-hop 10.48.15.254;
```

Configuring Junos Control Plane (JCP) for vRR

By default, the Junos Control Plane (JCP) VM comes up after NFX250 is powered on. The JCP virtual machine controls the front panel ports in the NFX250 device. VLANs provide the bridging between the virtual route reflector VM interfaces and the JCP VMs using sxe ports. The front panel ports are configured as part of the same VLAN bridging of the VRR ports. As a result, packets are transmitted or received using these bridging ports between JCP instead of the vRR VNF ports.

To configure JCP for vRR, perform the following steps:

1. In the operational mode, connect to the JCP virtual machine.

```
user@jdm>ssh vjunos0
user@jcp> configure
user@jcp#
```

2. Configure the front panel ports with the same VLAN bridging of vRR VNF ports. In this example, the front panel ports, ge-0/0/1, ge-0/0/10, and xe-0/0/12 are mapped with vRR VNF interfaces, em1, em2, and em3. The ge-0/0/1 (front panel port) maps to the sxe-0/0/0 (internal interface) which maps to em1 (vRR VNF interface). They are all part of the same VLAN (VLAN ID 100).

```
root#set interfaces ge-0/0/1 unit 0 family ethernet-switching interface-mode trunk
root# set interfaces ge-0/0/10 unit 0 family ethernet-switching interface-mode trunk
root# set interfaces xe-0/0/12 unit 0 family ethernet-switching interface-mode trunk
```

```
root# set interfaces sxe-0/0/0 unit 0 family ethernet-switching interface-mode trunk
root# set interfaces sxe-0/0/1 unit 0 family ethernet-switching interface-mode trunk
```

3. Configure the VLANs and add the physical interface and the service interface as members of the same VLAN. In this example, we have 3 VLANs (100, 101, and 102).

```
root@jcp# set vlans vlan100 vlan-id 100
root@jcp# set vlans vlan101 vlan-id 101
root@jcp# set vlans vlan102 vlan-id 102
root@jcp#set interfaces ge-0/0/1 unit 0 family ethernet-switching interface-mode trunk vlan members
vlan100
root@jcp# set interfaces ge-0/0/10 unit 0 family ethernet-switching interface-mode trunk vlan members
vlan101
root@jcp# set interfaces xe-0/0/12 unit 0 family ethernet-switching interface-mode trunk vlan members
vlan102
root@jcp# set interfaces sxe-0/0/0 unit 0 family ethernet-switching interface-mode trunk vlan members
100
root@jcp# set interfaces sxe-0/0/1 unit 0 family ethernet-switching interface-mode trunk vlan members
101-102
```

4. Configure MTU:

NOTE: The maximum MTU that you can configure on JCP and vRR VNF interfaces is 1518 bytes.

```
root@jcp# set interfaces ge-0/0/1 mtu 1518
root@jcp# set interfaces ge-0/0/1 unit 0 family ethernet-switching interface-mode trunk
root@jcp# set interfaces ge-0/0/1 unit 0 family ethernet-switching vlan members vlan100
root@jcp# set interfaces ge-0/0/1 unit 0 family ethernet-switching vlan members default
root@jcp# set interfaces ge-0/0/1 unit 0 storm-control default
```

5. Verify that you have configured the mapping of interfaces accurately.

```
sxe-0/0/0 {
    ether-options {
        no-flow-control;
    }
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members [ 100 default vlan-2 ];
            }
        }
    }
}
ge-0/0/1 {
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members [ default vlan100 ];
            }
            storm-control default;
        }
    }
}
sxe-0/0/1 {
    ether-options {
        no-flow-control;
    }
}
```

```

    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members [ 101-102 vlan-3 ];
            }
        }
    }
}
ge-0/0/10 {
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members [ default vlan101 ];
            }
            storm-control default;
        }
    }
}
xe-0/0/12 {
    unit 0 {
        family ethernet-switching {
            interface-mode trunk;
            vlan {
                members [ vlan-2 vlan102 ];
            }
            storm-control default;
        }
    }
}
vlangs{
..
    vlan100 {
        vlan-id 100;
    }
    vlan101 {
        vlan-id 101;
    }
    vlan102 {
        vlan-id 102;
    }
}

```



```
}
}
```

Launching vRR

You can launch the vRR VNF as a virtualized network function (VNF) using the XML configuration templates that are part of the vRR image archive.

1. To launch the vRR VNF, use the `virsh` command and specify the VM name.

```
root$ virsh define vrr.xml
root$ virsh start vrr
```

Where, `vrr` is the `virsh` domain name specified in the `vrr.xml`.

2. To create a VRR VNF of size 24GB with 2 virtual CPUs and 2 VNF interfaces (em2, em3) as shown in this example, you can use this sample configuration.

NOTE: To create a vRR VNF, use the default memory allocation mode and not hugepages.

```
<domain type='kvm' id='10'>
<!-- Assign VRR VM instance name -->
<name>template_vrr</name>
<!-- Assign Memory required for this VRR VM instance -->
<memory unit='KiB'>25769803</currentMemory>
<currentMemory unit='KiB'>25165824</currentMemory>
<memoryBacking>
<locked/>
</memoryBacking>
<!-- Assign required virtual CPU for VRR VM instance, here 2 vcpu is assigned -->
<vcpu placement='static' cpuset='0-1'>2</vcpu>
  <cputune>
    <vcupin vcpu='0' cpuset='0' />
    <vcupin vcpu='1' cpuset='1' />
  </cputune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <sysinfo type='smbios'>
    <bios>
```

```

    <entry name='vendor'>Juniper</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Juniper</entry>
    <entry name='product'>VRR</entry>
    <entry name='version'>16.1</entry>
  </system>
</sysinfo>
<os>
  <type arch='x86_64' machine='pc-i440fx-1.7'>hvm</type>
  <boot dev='hd' />
  <smbios mode='sysinfo' />
</os>
<features>
  <acpi />
  <apic />
</features>
<cpu mode='host-model'>
  <model fallback='allow' />
  <topology sockets='1' cores='4' threads='1' />
</cpu>
<clock offset='utc' />
<devices>
  <emulator>/usr/bin/kvm</emulator>
<!-- Provide VRR VM instance image location -->
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='writethrough' />
    <source file='/var/opt/test/nikramas/forsriov/fin_val/16.1R5.7/junos-x86-64-16.1R5.7.img' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0' />
  </disk>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>
<!-- em0 - management interface with the associated bridge -->
  <interface type='bridge'>
    <source bridge='eth0br' />
    <target dev='vnet5' />
    <model type='virtio' />
    <alias name='net0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
  </interface>
</devices>

```

```

    </interface>
<!-- em1 - internal data interface with the associated bridge -->
    <interface type='bridge'>
        <source bridge='virbr0' />
        <target dev='vnet6' />
        <model type='virtio' />
        <alias name='net1' />
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0' />
    </interface>
<!-- em2 - external data interface with the associated bridge -->
    <interface type='bridge'>
        <source bridge='sxe0br' />
        <target dev='vnet7' />
        <model type='virtio' />
        <alias name='net2' />
        <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0' />
    </interface>
<!-- em3 - external data interface with the associated bridge -->
    <interface type='bridge'>
        <source bridge='sxe1br' />
        <target dev='vnet8' />
        <model type='virtio' />
        <alias name='net3' />
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' />
    </interface>
    <serial type='pty'>
        <source path='/dev/pts/0' />
        <target port='0' />
        <alias name='serial0' />
    </serial>
    <console type='pty' tty='/dev/pts/0'>
        <source path='/dev/pts/0' />
        <target type='serial' port='0' />
        <alias name='serial0' />
    </console>
    <memballoon model='none'>
        <alias name='balloon0' />
    </memballoon>
</devices>
    <seclabel type='none' model='none' />
</domain>

```

Enabling Liveliness Detection of vRR VNF from JDM

Liveliness of a VNF indicates if the IP address of the VM is accessible to the Junos Device Manager (JDM). If the liveliness of the VM is down, it implies that the VM is not reachable from JDM. You can view the liveliness of VMs using the `show virtual-machines` command. By default, the liveliness of vRR VNF is shown as down. Before creating the vRR VNF, it is recommended that you enable liveliness detection in JDM.

To enable liveliness detection of the vRR VNF from JDM, perform the following steps:

1. To verify that liveliness detection of vRR VNF from JDM, issue the following command:

NOTE: By default, the liveliness of vRR VNF is shown as down. You must enable liveliness detection of vRR VNF from the JDM.

```
user@jdm> show virtual-machines
```

ID	Name	State	Liveliness
2	vjunos0	running	alive
18	VRR	running	down
8366	jdm	running	alive

2. Create a dummy interface with internal bridge, `virbr0`, by modifying the network interface settings for the vRR VNF interface. stanza of the VM template like as shown below. The PCI details like bus, slot, function information can be based on your existing interfaces arbitrary running next number, especially the slot number.

This is a sample network interface setting.

```
<interface type='bridge'>
  <source bridge='eth0br' />
<model type='e1000' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
</interface>
```

You must change the settings as follows:

```
<interface type='bridge'>
  <source bridge='virbr0' />
<model type='e1000' />
```

```
<address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0' />
</interface>
```

When you are modifying the settings make sure that:

- The interface type is 'bridge'.
 - The model type is e1000 to prevent problems with VLAN subinterfaces.
 - The PCI resource for the address is unique for this VM.
3. To identify the MAC address associated with the virbr0 interface, use the `virsh dumpxml vrr-vm-name` command.

```
user@jdm:/var/third-party# virsh dumpxml VRR

...
<interface type='bridge'>
<mac address='52:54:00:c4:fe:8d' />
  <source bridge='virbr0' />
<model type='e1000' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0' />
</interface>
...
```

This is the MAC address assigned by the vRR VNF to the virbr0 interface.

4. To assign an IP address to the vRR VNF interface connected to virbr0 interface, you must use an IP, that is part of the internal network. In this example, the MAC address, 52:54:00:c4:fe:8d, assigned by the vRR VNF is associated with the em4 interface of the vRR VNF. So, we must configure the em4 interface with the IP address as shown in this step.

```
user@> set interfaces em4 unit 0 family inet address 192.168.1.10/24
user@# run show interfaces em4
Physical interface: em4, Enabled, Physical link is Up
  Interface index: 130, SNMP ifIndex: 154
  Type: Ethernet, Link-level type: Ethernet, MTU: 1514, Speed: 1000mbps
  Device flags   : Present Running
  Interface flags: SNMP-Traps
  Link type      : Full-Duplex
  Current address: 52:54:00:c4:fe:8d, Hardware address: 52:54:00:c4:fe:8d
  Last flapped   : 2017-07-11 07:42:38 UTC (00:04:42 ago)
    Input packets : 40
    Output packets: 4
```

```

Logical interface em4.0 (Index 69) (SNMP ifIndex 155)
  Flags: Up SNMP-Traps 0x4000000 Encapsulation: ENET2
  Input packets : 37
  Output packets: 4
  Protocol inet, MTU: 1500
  Max nh cache: 100000, New hold nh limit: 100000, Curr nh cnt: 0,
  Curr new hold cnt: 0, NH drop cnt: 0
  Flags: Sendbroadcast-pkt-to-re, Is-Primary
  Addresses, Flags: Is-Default Is-Primary
    Local: 192.168.1.10

```

The MAC address assigned by the vRR VNF in this example is associated with em4 interface.

5. In Junos Device Manager (JDM), update the file `/etc/hosts` with the IP address and vRR VNF name.

NOTE: When you update the `/etc/hosts` file, include space between the IP address and the vRR VNF name. Do not include tab spaces.

```

user@jdm> cat /etc/hosts
127.0.0.1      localhost
::1           localhost ip6-localhost ip6-loopback
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters
192.168.1.1    hypervisor
192.168.1.2    vjunos0
192.168.1.3    vjunos1
192.168.1.4    vsrx
192.168.1.254  jdm nfx-vrr-jdm
128.0.0.254    jdm nfx-vrr-jdm
192.168.1.5    ipsec-nm
192.168.1.10   VRR

```

6. Ping the IP address of the vRR VNF from JDM to verify that the internal bridge virbr0 is accessible from JDM.

```

user@jdm> ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.316 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=0.379 ms

```

```
^C
--- 192.168.1.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.316/0.347/0.379/0.036 ms
```

7. To verify that liveness detection of vRR VNF from JDM, issue the following command:

```
user@jdm> show virtual-machines
```

ID	Name	State	Liveliness
2	vjunos0	running	alive
18	VRR	running	alive
8366	jdm	running	alive

Now, the liveness status of vRR VNF is shown as alive.

Configuring vRR VNF on NFX250 in Enhanced Orchestration Mode

Before you configure the vRR VNF, check the system inventory and confirm that the required resources are available by using `show system visibility` command. vRR as VNF must be designed and configured so that its resource requirements do not exceed the available capacity of the system.

You can instantiate the vRR VNF in Enhanced Orchestration (EO) mode by using the JDM CLI configuration and without using the XML descriptor file. EO mode uses Open vSwitch (OVS) as NFV backplane for bridging the interfaces.

To activate the vRR VNF from the Juniper Device Manager (JDM) CLI:

1. Download the **qcow2.img** vRR image to the `/var/third-party/images/` folder.
2. Define the vRR VNF. For example:

```
user@host# set virtual-network-functions VRR-1 image /var/third-party/images/junos-x86-64-18.4R1-S2.4.img
```

3. Define VLANs required for vSRX VNF interfaces. For example:

```
user@host# set host-os vlans vlan10 vlan-id 10
user@host# set host-os vlans vlan20 vlan-id 20
```

4. Allocate hugepages as memory for the vRR VNF. For example:

```
user@host# set system memory hugepages page-size 1024 page-count 8
```

5. Specify the number of virtual CPUs required for the vRR VNF. It is recommended that at least two virtual CPUs are assigned to a vRR VNF. For example:

```
user@host# set virtual-network-functions VRR-1 virtual-cpu count 2
```

6. Connect a virtual CPU to a physical CPU. For example:

```
user@host# set virtual-network-functions VRR-1 virtual-cpu 0 physical-cpu 8
user@host# set virtual-network-functions VRR-1 virtual-cpu 1 physical-cpu 9
```

7. Configure the maximum transmission unit (MTU) on the vRR interface (eth2). For example:

```
user@host# set virtual-network-functions VRR-1 interfaces eth2 mtu 2048
```

8. Configure the LAN-side internal-facing interface as a trunk port and add it to the LAN-side VLAN. For example:

```
user@host# set virtual-network-functions VRR-1 interfaces eth2 mapping vlan mode trunk
user@host# set virtual-network-functions VRR-1 interfaces eth2 mapping vlan members vlan10
```

9. Configure the maximum transmission unit (MTU) on the vRR interface (eth3). For example:

NOTE: MTU size can be either 1500 bytes or 2048 bytes.

```
user@host# set virtual-network-functions VRR-1 interfaces eth3 mtu 2048
```

10. Configure the LAN-side internal-facing interface as a trunk port and add it to the LAN-side VLAN. For example:

```
user@host# set virtual-network-functions VRR-1 interfaces eth3 mapping vlan mode trunk
user@host# set virtual-network-functions VRR-1 interfaces eth3 mapping vlan members vlan20
```


11. Specify the memory allocation for the vRR VNF. It is recommended that at least 4GB memory is allocated to the vRR VNF. For example:

```
user@host# set virtual-network-functions VRR-1 memory size 4194304
```

12. Configure hugepages for memory requirements. For example:

```
user@host# set virtual-network-functions VRR-1 memory features hugepages
```

13. Commit the configuration to activate vRR VNF. For example:

```
user@host# commit
```

After you commit the configuration, VNF takes some time to boot. The first interface (em0) is automatically given an IP address by DHCP from JDM for liveliness.

14. Verify that the VNF is up. For example:

```
user@host> show virtual-network-functions brief
```

ID	Name	State	Liveliness
-			
1	vjunos0	Running	alive
5	VRR-1	Running	alive
9887	jdm	Running	alive

15. (Optional) Verify the vRR VNF details. For example:

```
user@host> show virtual-network-functions VRR-1 detail
```

Virtual Network Function Information

-

```
Id:          5
Name:        VRR-1
State:       Running
Liveliness:  alive
IP Address:  192.0.2.100
VCPUs:      2
Maximum Memory: 4194304 KiB
Used Memory:  45015 KiB
Used 1G Hugepages: 2
```

```
Used 2M Hugepages: 0
Error:             None
```

Configuring Cross-connect

The **Cross-connect** feature enables traffic switching between any two OVS interfaces such as VNF interfaces or physical interfaces such as `hsxe0` and `hsxe1` that are connected to the OVS. You can bidirectionally switch either all traffic or traffic belonging to a particular VLAN between any two OVS interfaces.

NOTE: This feature does not support unidirectional traffic flow.

The **Cross-connect** feature supports the following:

- Unconditional cross-connect between two VNF interfaces for all network traffic.
- VLAN-based traffic forwarding between VNF interfaces support the following functions:
 - Provides an option to switch traffic based on a VLAN ID.
 - Supports network traffic flow from trunk to access port.
 - Supports network traffic flow from access to trunk port.
 - Supports VLAN PUSH, POP, and SWAP operations.

To configure cross-connect:

1. Configure VLANs:

```
[edit]
user@jdm#set host-os vlans vlan-name vlan-id vlan-id
user@jdm#set host-os vlans v10 vlan-id 10
user@jdm#set host-os vlans v20 vlan-id 20
user@jdm#set host-os vlans v30 vlan-id 30
user@jdm#set host-os vlans v40 vlan-id 40
```

2. Configure VNFs:

```

user@jdm# set virtual-network-functions vnf-name interfaces interface-name mapping vlan members list
user@jdm# set virtual-network-functions vnf1 interfaces eth1 mapping vlan members v10
user@jdm# set virtual-network-functions vnf1 interfaces eth1 mapping vlan members v30
user@jdm# set virtual-network-functions vnf1 interfaces eth1 mapping vlan members v40
user@jdm# set virtual-network-functions vnf1 interfaces eth1 mapping vlan mode trunk
user@jdm# set virtual-network-functions vnf2 interfaces eth2 mapping vlan members v10
user@jdm# set virtual-network-functions vnf2 interfaces eth2 mapping vlan members v20
user@jdm# set virtual-network-functions vnf2 interfaces eth2 mapping vlan members v30
user@jdm# set virtual-network-functions vnf2 interfaces eth2 mapping vlan mode trunk
user@jdm# set virtual-network-functions vnf2 interfaces eth3 mapping vlan members v10
user@jdm# set virtual-network-functions vnf2 interfaces eth3 mapping vlan members v20
user@jdm# set virtual-network-functions vnf2 interfaces eth3 mapping vlan members v30
user@jdm# set virtual-network-functions vnf2 interfaces eth3 mapping vlan mode trunk
user@jdm# set virtual-network-functions vnf4 interfaces eth4 mapping vlan members v30
user@jdm# set virtual-network-functions vnf5 interfaces eth5 mapping vlan members v50
user@jdm# set virtual-network-functions vnf5 interfaces eth6
user@jdm# set virtual-network-functions vnf6 interfaces eth7

```

3. Configure cross-connect:

- Configure VLAN-based cross-connect:

```

user@jdm# set cross-connect vlan-cross-connect-name virtual-network-functions vnf-name interfaces
interface-name vlan-id vlan-id
user@jdm# set cross-connect c1 virtual-network-functions vnf1 interfaces eth2 vlan-id 10
user@jdm# set cross-connect c1 virtual-network-functions vnf4 interfaces eth2 vlan-id 10

```

- Configure unconditional cross-connect

```

user@jdm# set cross-connect unconditional-cross-connect-name virtual-network-functions vnf-name
interfaces interface-name
user@jdm# set cross-connect c2 virtual-network-functions vnf1 interfaces eth1
user@jdm# set cross-connect c2 virtual-network-functions vnf5 interfaces eth5

```

- Configure cross-connect with VLAN SWAP operation enabled:

```

user@jdm# set cross-connect swap-cross-connect-name virtual-network-functions vnf-name interfaces
interface-name vlan-id vlan-id

```

```
user@jdm# set cross-connect c1 virtual-network-functions vnf1 interfaces eth1 vlan-id 10
user@jdm# set cross-connect c1 virtual-network-functions vnf4 interfaces eth4 vlan-id 30
```

- Configure cross-connect with VLAN PUSH or POP operation enabled:

```
user@jdm# set cross-connect push-pop-cross-connect-name virtual-network-functions vnf-name
interfaces interface-name vlan-id vlan-id
user@jdm# set cross-connect push-pop-cross-connect-name virtual-network-functions vnf-name
interfaces interface-name
user@jdm# set cross-connect c3 virtual-network-functions vnf1 interfaces eth1 vlan-id 10
user@jdm# set cross-connect c3 virtual-network-functions vnf6 interfaces eth7
```

- Configure native VLAN traffic on cross-connect

```
user@jdm# set cross-connect push-pop cross-connect-name virtual-network-functions vnf-name
interfaces interface-name vlan-id vlan-id
user@jdm# set cross-connect c2 virtual-network-functions vnf1 interfaces eth1 vlan-id none
user@jdm# set cross-connect c2 virtual-network-functions vnf5 interfaces eth5 vlan-id none
```

Configuring Analyzer VNF and Port-mirroring

The **Port-mirroring** feature allows you to monitor network traffic. If the feature is enabled on a VNF interface, the OVS system bridge sends a copy of all network packets of that VNF interface to the analyzer VNF for analysis. You can use the port-mirroring or analyzer JDM commands for analyzing the network traffic.

NOTE:

- Port-mirroring is supported only on VNF interfaces that are connected to an OVS system bridge.
- VNF interfaces must be configured before configuring port-mirroring options.
- If the analyzer VNF is active after you configure, you must restart the VNF for changes to take effect.

- You can configure up to four input ports and only one output port for an analyzer rule.
- Output ports must be unique in all analyzer rules.
- After changing the configuration of the input VNF interfaces, you must de-activate and activate the analyzer rules referencing to it along with the analyzer VNF restart.

To configure the analyzer VNF and enable port-mirroring:

1. Configure the analyzer VNF:

```
[edit]
user@jdm#set virtual-network-functions analyzer-vnf-name image file-path
user@jdm#set virtual-network-functions analyzer-vnf-name memory features hugepages page-size page-size
user@jdm#set virtual-network-functions analyzer-vnf-name interfaces interface-name analyzer
```

2. Enable port-mirroring of the network traffic in the input and output ports of the VNF interface and analyzer VNF:

```
user@jdm# set host-os forwarding-options analyzer analyzer-instance-name input [ingress | egress]
virtual-network-function vnf-name interface interface-name
user@jdm# set host-os forwarding-options analyzer analyzer-rule-name output virtual-network-function
analyzer-vnf-name interface interface-name
```

Release History Table

Release	Description
17.3R1	Starting in Junos OS Release 17.3R1, you can implement the virtual route reflector (vRR) feature on the NFX250 Network Services platform.

RELATED DOCUMENTATION

[NFX250 Overview](#) | 2

[JDM Architecture Overview](#) | 6

[virtual-network-functions](#) | 184

[show virtual-network-functions](#) | 195

Configuring Service Chaining Using JDM

IN THIS SECTION

- Understanding Service Chaining on Disaggregated Junos OS Platforms | 109
- Configuring Service Chaining Using VLANs | 110
- Configuring Service Chaining Using DHCP Services on VLANs | 110
- Example: Configuring Service Chaining Using VLANs on NFX250 Network Services Platform | 111
- Example: Configuring Service Chaining Using SR-IOV on NFX250 Network Services Platform | 118

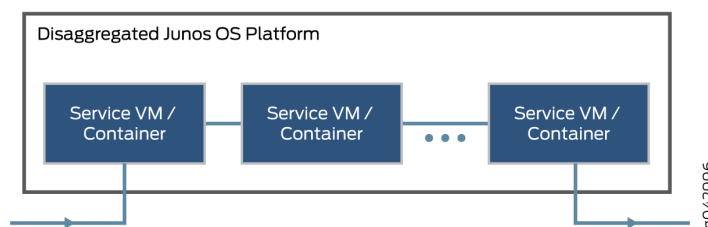
Understanding Service Chaining on Disaggregated Junos OS Platforms

In many network environments, it is common for traffic to flow through several *network services* on the way to its destination. These services—firewalls, Network Address Translators (NAT), load balancers, and so on—are generally spread across multiple network elements. Each device is a separate piece of hardware, providing a different service, and requiring separate operation and management. This method of linking together multiple network functions could be thought of as *physical service chaining*.

A more efficient model for service chaining is to virtualize and consolidate network functions onto a single device.

Platforms running the disaggregated Junos OS software support virtualized service chaining. These devices enable virtual network functions (VNFs) by supporting the installation and instantiation of VNFs. VNFs can be linked together to provide network services for traffic flowing through the device, as shown in [Figure 15 on page 109](#).

Figure 15: Virtual Network Functions on a Disaggregated Junos OS Platform



Configuring Service Chaining Using VLANs

You can achieve service chaining using VLANs.

- Ensure that connectivity to the host is not lost during the configuration process.

To configure service chaining:

1. Create a VLAN. Use one of the following commands:

- Create a VLAN without a VLAN ID. You can add only access ports to this VLAN:

```
set host-os vlans vlan-name vlan-id none
```

- Create a VLAN with a VLAN ID:

```
set host-os vlans vlan-name vlan-id vlan-id
```

- Create a VLAN using a list of VLAN IDs:

```
set host-os vlans vlan-name vlan-id-list vlan-id range | comma-separated list
```

2. Attach an interface on the VNF to the VLAN:

```
set virtual-network-functions vnf-name interfaces ethx mapping vlan mode [access|trunk]
set virtual-network-functions vnf-name interfaces ethx mapping vlan members list
```

3. Attach a native VLAN ID to the VNF interface:

```
set virtual-network-functions vnf-name interfaces ethx mapping vlan native-vlan-id vlan-id
```

Configuring Service Chaining Using DHCP Services on VLANs

Using DHCP services, you need not manually configure the IP addresses on the VNF interfaces to achieve service-chaining. Enable DHCP clients on the glue bridge interfaces within the VNF for an IP address to be assigned from the DHCP pool. Based on the IP subnet, the IRB interface on the VLAN is automatically mapped to the corresponding subnet dhcp pool.

To configure service chaining:

1. Create a VLAN with a VLAN ID `none`.

```
set host-os vlans vlan-name vlan-id none
```

NOTE: To use the DHCP pooling feature, the VLAN ID must be set to `none`.

2. Create IRB interfaces on the hypervisor

```
set host-os interfaces irb unit logical-unit-number family inet address inet-address
set host-os vlans vlan-name l3-interface irb. logical-interface-number
```

3. Specify the IP address pool to be used:

```
set access address-assignment pool p4 family inet network network-address
set access address-assignment pool p4 family inet range r4 low start-IP-address
set access address-assignment pool p4 family inet range r4 high end-IP-address
```

4. Attach an interface on the VNF to the VLAN to complete the service chain:

```
set virtual-network-functions vnf-name interfaces ethx mapping vlan mode [access|trunk]
set virtual-network-functions vnf-name interfaces ethx mapping vlan members list
```

5. Enable the DHCP client on the VNF.

To check the assigned IP address, use the [show system visibility vnf](#) command.

Example: Configuring Service Chaining Using VLANs on NFX250 Network Services Platform

IN THIS SECTION

- [Requirements | 112](#)
- [Overview | 112](#)

This example shows how to configure service chaining using VLANs on the host bridge.

Requirements

This example uses the following hardware and software components:

- NFX250 running Junos OS Release 15.1X53-D45

Before you configure service chaining, be sure you have:

- Installed and launched the relevant VNFs, assigned the corresponding interfaces, and configured the resources.

Overview

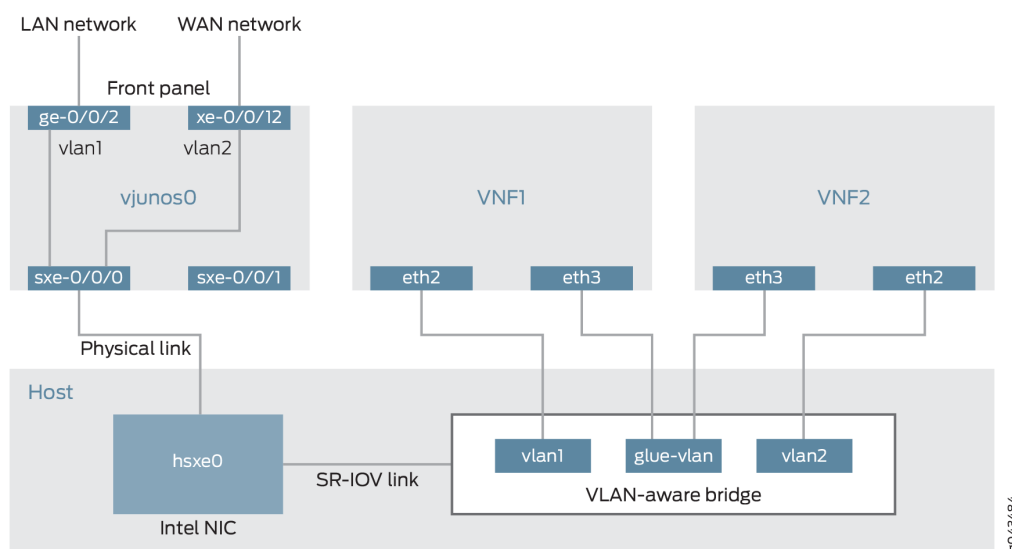
IN THIS SECTION

Service chaining on a device running the disaggregated Junos OS allows multiple services, or virtual network functions (VNFs), to be applied to traffic as it flows through the device. This example explains how to configure the various layers of the device to enable traffic to enter the device, flow through two service VNFs, and exit the device.

Topology

This example uses a single device running the disaggregated Junos OS, as shown in [Figure 16 on page 113](#).

Figure 16: Service Chaining Using VLANs



This example is configured using the Juniper Device Manager (JDM) and Junos Control Plane (JCP). The key configuration elements include:

- The Packet Forwarding Engine's front panel ports.
- The Packet Forwarding Engine's internal-facing ports.
- A routing instance named *host-os*. The *host-os* routing instance is the CLI construct that provides the ability to configure host OS elements from the JDM.
- NIC ports. As these interfaces are not directly configurable, they are abstracted in the host OS. Using the JDM CLI, NIC interfaces (sxe ports) are configured in the *host-os* routing instance as "hsxe" interfaces.
- The VM interfaces. In the JDM, VNF interfaces must use the format eth#, where # is from 2 through to 9.
- VLANs, to provide bridging between the sxe and VM interfaces.

Configuration

IN THIS SECTION

- [Configuring the Packet Forwarding Engine Interfaces | 114](#)
- [Configuring the VNF Interfaces and Creating the Service Chain | 117](#)

Configuring the Packet Forwarding Engine Interfaces

CLI Quick Configuration

To quickly configure the Packet Forwarding Engine interfaces, enter the following configuration statements from the JCP:

```
[edit]
user@jcp#

set vlans vlan1 vlan-id 77
set interfaces ge-0/0/2.0 family ethernet-switching vlan members vlan1
set interfaces sxe-0/0/0.0 family ethernet-switching interface-mode trunk
set interfaces sxe-0/0/0.0 family ethernet-switching vlan members vlan1
set vlans vlan2 vlan-id 1177
set interfaces xe-0/0/12.0 family ethernet-switching interface-mode trunk
set interfaces xe-0/0/12.0 family ethernet-switching vlan members vlan2
set interfaces sxe-0/0/0.0 family ethernet-switching vlan members vlan2
```

Step-by-Step Procedure

To configure the Packet Forwarding Engine interfaces:

1. Connect to the JCP.

```
user@jdm> ssh vjunos0
user@jcp> configure
[edit]
```

```
user@jcp#
```

2. Configure a VLAN for the LAN-side interfaces.

```
user@jcp# set vlans vlan1 vlan-id 77
```

3. Configure the Packet Forwarding Engine's LAN-side front panel port and add it to the LAN-side VLAN.

The LAN-side port is typically an access port, but could be a trunk port if appropriate.

```
user@jcp# set interfaces ge-0/0/2.0 family ethernet-switching vlan members vlan1
```

4. Configure the Packet Forwarding Engine's LAN-side internal-facing interface as a trunk port and add it to the LAN-side VLAN.

The internal-facing interfaces are typically trunk ports, as they must support traffic from multiple front panel ports and VLANs.

```
user@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching interface-mode trunk
user@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching vlan members vlan1
```

5. Configure a VLAN for the WAN-side interfaces.

```
user@jcp# set vlans vlan2 vlan-id 1177
```

6. Configure the Packet Forwarding Engine's WAN-side front panel port as a trunk port and add it to the WAN-side VLAN.

The WAN-side front panel port is typically a trunk port, as it might be required to support multiple VLANs.

```
user@jcp# set interfaces xe-0/0/12.0 family ethernet-switching interface-mode trunk
user@jcp# set interfaces xe-0/0/12.0 family ethernet-switching vlan members vlan2
```

7. Configure the Packet Forwarding Engine's WAN-side internal-facing interface as a trunk port and add it to the WAN-side VLAN.

The internal-facing interfaces are typically trunk ports, as they must support traffic from multiple front panel ports and VLANs.

```
user@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching vlan members vlan2
```

8. Commit the configuration and return to the JDM.

```
user@jcp# commit and-quit
user@jcp> exit
user@jdm>
```

Results

From configuration mode, check the results of your configuration by entering the following **show** commands:

```
[edit]
user@jcp# show interfaces ge-0/0/2
unit 0 {
    family ethernet-switching {
        vlan {
            members vlan1;
        }
    }
}

[edit]
```

```
user@jcp# show interfaces xe-0/0/12
```

```
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members vlan2;
        }
    }
}
```

```
[edit]
```

```
user@jcp# show interfaces sxe-0/0/0
```

```
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members [vlan1 vlan2];
        }
    }
}
```

```
[edit]
```

```
user@jcp# show vlans
```

```
vlan2 {
    vlan-id 1177;
}
vlan1 {
    vlan-id 77;
}
```

Configuring the VNF Interfaces and Creating the Service Chain

Step-by-Step Procedure

Once you have completed the configuration on JCP, you need to:

1. Configure the host-os instance with either with LAN, WAN, or glue-vlan to be used for service chaining

```
user@jdm# set host-os vlans vlan1 vlan-id 77
user@jdm# set host-os vlans vlan2 vlan-id 1177
user@jdm# set host-os vlans glue-vlan vlan-id 123
```

2. Bring up the VM1 with one virtio interface mapped to VLAN, and another interface mapped to glue-vlan.

```
user@jdm# set virtual-network-functions VM1 interfaces eth2 mapping vlan members vlan1
user@jdm# set virtual-network-functions VM1 interfaces eth3 mapping vlan members glue-vlan
```

3. Similarly bring up VM2 with one interface with one interface mapped to VLAN2, and the second interface mapped to the same glue-vlan.

```
user@jdm# set virtual-network-functions VM2 interfaces eth2 mapping vlan members vlan2
user@jdm# set virtual-network-functions VM2 interfaces eth3 mapping vlan members glue-vlan
```

4. Finally, configure the IP addresses and static routes for each interface of the VMs as shown in [Figure 16 on page 113](#).

Example: Configuring Service Chaining Using SR-IOV on NFX250 Network Services Platform

IN THIS SECTION

- Requirements | [119](#)
- Overview | [119](#)
- Configuration | [121](#)

This example shows how to configure service chaining using SR-IOV on platforms running the disaggregated Junos OS software.

Requirements

This example uses the following hardware and software components:

- NFX250 running Junos OS Release 15.1X53-D45

Before you configure service chaining, be sure you have:

- Installed and launched the relevant VNFs

Overview

IN THIS SECTION

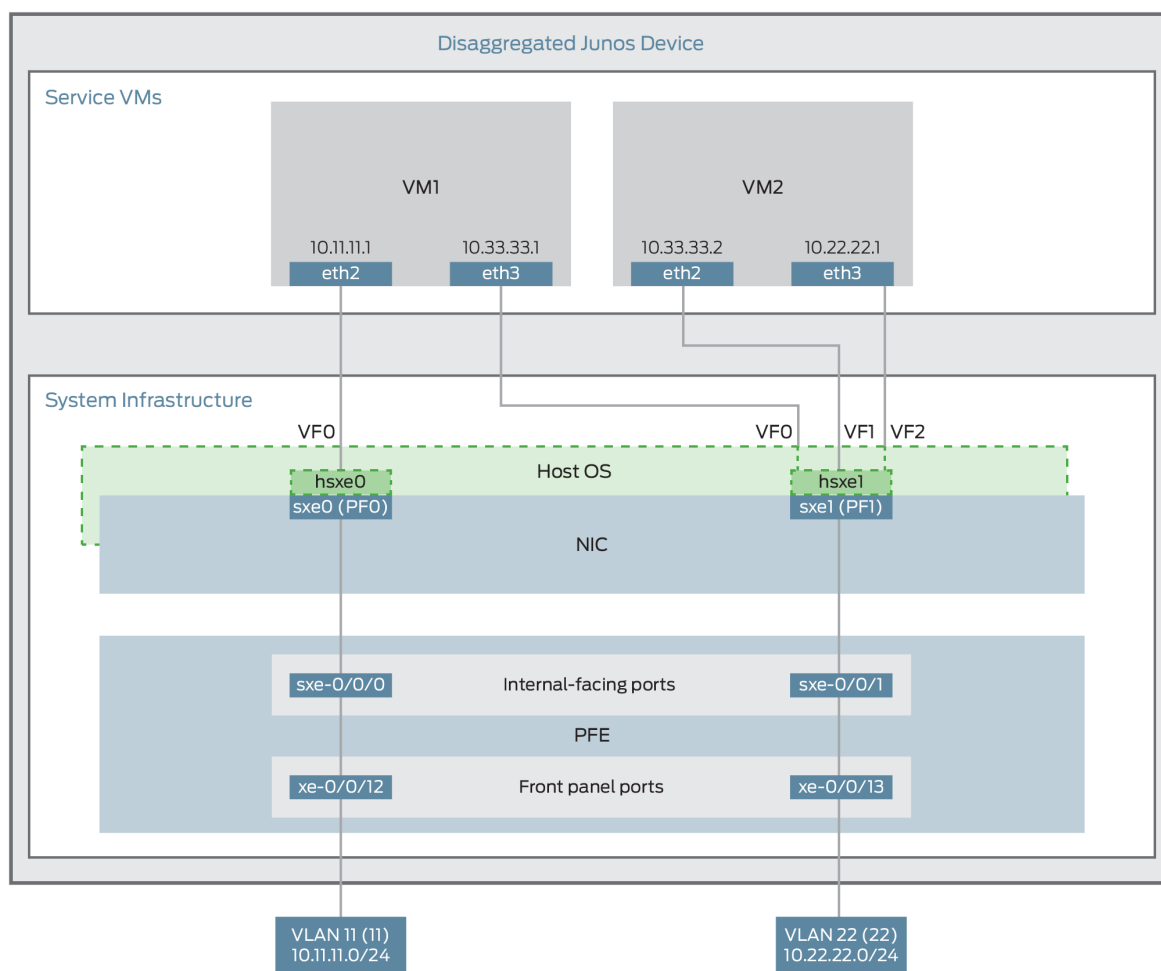
- [Topology](#) | [120](#)

Service chaining on a device running the disaggregated Junos OS allows multiple services, or virtual network functions (VNFs), to be applied to traffic as it flows through the device. This example explains how to configure the various layers of the device to enable traffic to enter the device, flow through two service VNFs, and exit the device.

Topology

This example uses a single device running the disaggregated Junos OS, as shown in [Figure 17 on page 120](#).

Figure 17: Service Chaining Using SR-IOV—Device Infrastructure



This example uses the Packet Forwarding Engine's front panel ports xe-0/0/12 and xe-0/0/13, and its internal-facing ports, sxe-0/0/0 and sxe-0/0/1. The internal NIC's two ports (sxe0 and sxe1) are not configured directly; instead, they are abstracted at the host OS layer and configured as interfaces hsxe0 and hsxe1. The VMs use two interfaces each (eth2 and eth3).

These elements are generally separated into two parts: a *LAN side* and a *WAN side*.

As this example uses SR-IOV, the NIC ports' virtual functions (VFs) are used to bypass the host OS and provide direct NIC-to-VM connectivity. Given this setup, it might seem unusual to see host OS

interfaces (hsxe0 and hsxe1) included in this scenario. However, as there is no direct configuration method for the NIC ports, it is necessary to use their abstracted versions, hsxe0 and hsxe1.

This example is configured using the Juniper Device Manager (JDM) and Junos Control Plane (JCP). The key configuration elements include:

- The Packet Forwarding Engine's front panel ports.
- The Packet Forwarding Engine's internal-facing ports.
- NIC ports. Because NIC interfaces (sxe ports) cannot be configured directly, the host OS construct for these interfaces (hsxe) must be used.
- The VNF interfaces. In the JDM, VNF interfaces must use the format eth#, where # is from 2 through to 9.
- The virtual function setting, to indicate SR-IOV is being used to provide direct access between hsxe and VNF interfaces.

Configuration

IN THIS SECTION

- [Configuring the Packet Forwarding Engine Interfaces | 121](#)
- [Creating the Service Chain | 125](#)

This example describes:

Configuring the Packet Forwarding Engine Interfaces

CLI Quick Configuration

To quickly configure the Packet Forwarding Engine interfaces, enter the following configuration statements from the JCP:

```
[edit]
user@jcp#

set vlans Vlan11 vlan-id 11
set interfaces xe-0/0/12.0 family ethernet-switching vlan member Vlan11
set interfaces sxe-0/0/0.0 family ethernet-switching interface-mode trunk
```

```

set interfaces sxe-0/0/0.0 family ethernet-switching vlan member Vlan11
set vlans Vlan22 vlan-id 22
set interfaces xe-0/0/13.0 family ethernet-switching interface-mode trunk
set interfaces xe-0/0/13.0 family ethernet-switching vlan member Vlan22
set interfaces sxe-0/0/1.0 family ethernet-switching interface-mode trunk
set interfaces sxe-0/0/1.0 family ethernet-switching vlan member Vlan22

```

Step-by-Step Procedure

To configure the Packet Forwarding Engine interfaces:

1. Connect to the JCP.

```

user@jdm> ssh vjunos0
user@jcp> configure
[edit]
user@jcp#

```

2. Configure a VLAN for the LAN-side interfaces.

```

user@jcp# set vlans Vlan11 vlan-id 11

```

3. Configure the Packet Forwarding Engine's LAN-side front panel port, and add it to the LAN-side VLAN.

The LAN-side port is typically an access port, but could be a trunk port if appropriate.

```

user@jcp# set interfaces xe-0/0/12.0 family ethernet-switching vlan members Vlan11

```

4. Configure the Packet Forwarding Engine's LAN-side internal-facing interface as a trunk port, and add it to the LAN-side VLAN.

The internal-facing interfaces are typically trunk ports, as they must support traffic from multiple front panel ports and VLANs.

```
user@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching interface-mode trunk
user@jcp# set interfaces sxe-0/0/0.0 family ethernet-switching vlan member Vlan11
```

5. Configure a VLAN for the WAN-side interfaces.

```
user@jcp# set vlans Vlan22 vlan-id 22
```

6. Configure the Packet Forwarding Engine's WAN-side front panel port as a trunk port, and add it to the WAN-side VLAN.

The WAN-side front panel port is typically a trunk port, as it might be required to support multiple VLANs.

```
user@jcp# user@jcp# set interfaces xe-0/0/13.0 family ethernet-switching interface-mode trunk
user@jcp# user@jcp# set interfaces xe-0/0/13.0 family ethernet-switching vlan members Vlan22
```

7. Configure the Packet Forwarding Engine's WAN-side internal-facing interface as a trunk port, and add it to the WAN-side VLAN.

The internal-facing interfaces are typically trunk ports, as they must support traffic from multiple front panel ports and VLANs.

```
user@jcp# set interfaces sxe-0/0/1.0 family ethernet-switching interface-mode trunk
user@jcp# set interfaces sxe-0/0/1.0 family ethernet-switching vlan members Vlan22
```

8. Commit the configuration and return to the JDM.

```
user@jcp# commit and-quit
user@jcp> exit
user@jdm>
```

Results

From configuration mode, check the results of your configuration by entering the following **show** commands:

```
[edit]
user@jcp# show interfaces xe-0/0/12
unit 0 {
    family ethernet-switching {
        vlan {
            members Vlan11;
        }
    }
}
```

```
[edit]
user@jcp# show interfaces xe-0/0/13
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members Vlan22;
        }
    }
}
```

```
[edit]
user@jcp# show interfaces sxe-0/0/0
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
        vlan {
            members Vlan11;
        }
    }
}
```

```
[edit]
user@jcp# show interfaces sxe-0/0/1
unit 0 {
    family ethernet-switching {
        interface-mode trunk;
```

```

        vlan {
            members Vlan22;
        }
    }
}

[edit]
user@jcp# show vlans
Vlan11 {
    vlan-id 11;
}
Vlan22 {
    vlan-id 22;
}

```

Creating the Service Chain

Step-by-Step Procedure

To configure the VNF interfaces and create the service chain:

1. Configure VM1's LAN-side interface as a Layer 3 interface, and map it to the LAN-side NIC interface. Include the virtual function (VF) setting to specify direct NIC-to-VM connectivity. VNF must use the interfaces from eth2 through to eth9.

The hsxe interface is the configurable representation of the related NIC (sxe) interface.

```

user@jdm> configure
[edit]
user@jdm# set virtual-network-functions vnf1 interfaces eth2 mapping hsxe0 virtual-function

```

2. Configure VM1's WAN-side interface from sxe1 NIC as shown in [Figure 17 on page 120](#).

```

user@jdm# set virtual-network-functions vnf1 interfaces eth3 mapping hsxe1 virtual-function

```

3. Similarly bring up VM2 with both interfaces eth2 and eth3 on sxe1 NIC.

```

user@jdm# set virtual-network-functions vnf2 interfaces eth2 mapping hsxe1 virtual-function
user@jdm# set virtual-network-functions vnf2 interfaces eth3 mapping hsxe1 virtual-function

```

4. Finally, configure the IP addresses and static routes for each interface of the VNFs, and add routes to achieve the complete bidirectional path for the service chain.

RELATED DOCUMENTATION

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

ADSL2 and ADSL2+ Interfaces on NFX250 Devices

IN THIS SECTION

- [ADSL Interface Overview | 126](#)
- [Example: Configuring ADSL SFP Interface on NFX250 Devices | 127](#)

ADSL Interface Overview

IN THIS SECTION

- [ADSL2 and ADSL2+ | 127](#)

Asymmetric digital subscriber line (ADSL) technology is part of the xDSL family of modem technologies that use existing twisted-pair telephone lines to transport high-bandwidth data. ADSL lines connect service provider networks and customer sites over the "last mile" of the network—the loop between the service provider and the customer site.

ADSL transmission is asymmetric because the downstream bandwidth is typically greater than the upstream bandwidth. The typical bandwidths of ADSL2 and ADSL2+ circuits are defined in [Table 8 on page 127](#).

Table 8: Standard Bandwidths of DSL Operating Modes

Operating Modes	Upstream	Downstream
ADSL2	1–1.5 Mbps	12–14 Mbps
ADSL2+	1–1.5 Mbps	24–25 Mbps

ADSL2 and ADSL2+ support the following standards:

- LLC SNAP bridged 802.1q
- VC MUX bridged

Supported security devices with xDSL SFP can use PPP over Ethernet (PPPoE) to connect through ADSL lines only.

ADSL2 and ADSL2+

The ADSL2 and ADSL2+ standards were adopted by the ITU in July 2002. ADSL2 improves the data rate and reach performance, diagnostics, standby mode, and interoperability of ADSL modems.

ADSL2+ doubles the possible downstream data bandwidth, enabling rates of 20 Mbps on telephone lines shorter than 5000 feet (1.5 km).

ADSL2 uses seamless rate adaptation (SRA) to change the data rate of a connection during operation with no interruptions or bit errors. The ADSL2 transceiver detects changes in channel conditions—for example, the failure of another transceiver in a multicarrier link—and sends a message to the transmitter to initiate a data rate change. The message includes data transmission parameters such as the number of bits modulated and the power on each channel. When the transmitter receives the information, it transitions to the new transmission rate.

Example: Configuring ADSL SFP Interface on NFX250 Devices

IN THIS SECTION

- [Requirements | 128](#)
- [Overview | 128](#)

- [Configuration | 128](#)
- [Results | 130](#)

Requirements

This example uses the following hardware and software components:

- NFX250 device running Junos OS Release 15.1X53-D495.

Overview

In this example, you are configuring ADSL SFP interface on an NFX250 device with the following configurations:

- Physical interface - **ge-0/0/11**
- Virtual network function (VNF) - **nfx250-a-vsrx1**
- Memory size - **4194304**
- ADSL SFP options - **vpi3, vci34, and encaps llcsnap-bridged-802dot1q**

To configure ADSL SFP interface on NFX250 devices, you must configure JDM, vSRX, and vJunos0.

NOTE: Ensure that connectivity to the host is not lost during the configuration process.

Configuration

IN THIS SECTION

- [Procedure | 128](#)

Procedure

Step-by-Step Procedure

To configure ADSL SFP interfaces on NFX250 devices:

1. Connect to the host.

```
user@host> configure
[edit]
user@host#
```

2. Create VLANs using VLAN IDs:

```
user@host# set host-os vlans xdsl-test vlan-id 50
user@host# set host-os vlans vlan130 vlan-id 130
user@host# set host-os vlans vlan131 vlan-id 131
user@host# set host-os vlans vlan132 vlan-id 132
```

3. Enable enhanced orchestration to manage VNFs and service chains without requiring the VNF XML descriptor files:

```
user@host# set system services enhanced-orchestration
```

4. Allocate resources for a VNF:

```
user@host# set virtual-network-functions nfx250-a-vsrx1 image /var/third-party/images/media-vsrx-
vmdisk-15.1-2018-04-24.0_DEV_X_151_X49.qcow2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu 0 physical-cpu 2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu 1 physical-cpu 6
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu count 2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu features hardware-virtualization
user@host# set virtual-network-functions nfx250-a-vsrx1 no-default-interfaces
user@host# set virtual-network-functions nfx250-a-vsrx1 memory size 4194304
user@host# set virtual-network-functions nfx250-a-vsrx1 memory features hugepages
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth0 management out-of-band
```

5. Map physical interfaces to virtual interfaces:

```
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth1 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth1 mapping vlan members vlan130
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan130
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan131
```

```

user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan132
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members xdsl-test
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan native-vlan-id 50
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth3 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth3 mapping vlan members xdsl-test
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth3 mapping vlan native-vlan-id 50

```

6. Configure the Junos Control Plane (JCP) virtual machine (VM):

```

user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching interface-mode trunk
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members xdsl-test
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan130
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan131
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan132
user@host# set interfaces ge-0/0/11 native-vlan-id 50
user@host# set interfaces ge-0/0/11 dsl-sfp-options adsl-options vpi 3
user@host# set interfaces ge-0/0/11 dsl-sfp-options adsl-options vci 34
user@host# set interfaces ge-0/0/11 dsl-sfp-options adsl-options encap llcsnap-bridged-802dot1q
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching interface-mode trunk
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members xdsl-test
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan130
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan131
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan132
user@host# set vlans vlan130 vlan-id 130
user@host# set vlans vlan131 vlan-id 131
user@host# set vlans vlan132 vlan-id 132
user@host# set vlans xdsl-test vlan-id 50

```

7. Commit the configuration.

```

user@host# commit and-quit
user@host> exit

```

Results

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

VDSL2 Interfaces on NFX250 Devices

IN THIS SECTION

- [VDSL Interface Overview | 131](#)
- [VDSL2 Network Deployment Topology | 132](#)
- [VDSL2 Interface Support on NFX Series Devices | 134](#)
- [Example: Configuring VDSL SFP Interface on NFX250 Devices | 136](#)

VDSL Interface Overview

IN THIS SECTION

- [VDSL2 Vectoring Overview | 132](#)

Very-high-bit-rate digital subscriber line (VDSL) technology is part of the xDSL family of modem technologies that provide faster data transmission over a single flat untwisted or twisted pair of copper wires. The VDSL lines connect service provider networks and customer sites to provide high bandwidth applications (triple-play services) such as high-speed Internet access, telephone services like VoIP, high-definition TV (HDTV), and interactive gaming services over a single connection.

VDSL2 is an enhancement to G.993.1 (VDSL) and permits the transmission of asymmetric (half-duplex) and symmetric (full-duplex) aggregate data rates up to 100 Mbps on short copper loops using a bandwidth up to 17 MHz. The VDSL2 technology is based on the ITU-T G.993.2 (VDSL2) standard, which is the International Telecommunication Union standard describing a data transmission method for VDSL2 transceivers.

The VDSL2 uses discrete multitone (DMT) modulation. DMT is a method of separating a digital subscriber line signal so that the usable frequency range is separated into 256 frequency bands (or

channels) of 4.3125 KHz each. The DMT uses the Fast Fourier Transform (FFT) algorithm for demodulation or modulation for increased speed.

VDSL2 interface supports Packet Transfer Mode (PTM). The PTM mode transports packets (IP, PPP, Ethernet, MPLS, and so on) over DSL links as an alternative to using Asynchronous Transfer Mode (ATM). PTM is based on the Ethernet in the First Mile (EFM) IEEE802.3ah standard.

VDSL2 provides backward compatibility with ADSL2 and ADSL2+ because this technology is based on both the VDSL1-DMT and ADSL2/ADSL2+ recommendations.

VDSL2 Vectoring Overview

Vectoring is a transmission method that employs the coordination of line signals that reduce crosstalk levels and improve performance. It is based on the concept of noise cancellation, like noise-cancelling headphones. The ITU-T G.993.5 standard, "Self-FEXT Cancellation (Vectoring) for Use with VDSL2 Transceivers," also known as G.vector, describes vectoring for VDSL2.

The scope of Recommendation ITU-T G.993.5 is specifically limited to the self-FEXT (far-end crosstalk) cancellation in the downstream and upstream directions. The FEXT generated by a group of near-end transceivers and interfering with the far-end transceivers of that same group is canceled. This cancellation takes place between VDSL2 transceivers, not necessarily of the same profile.

VDSL2 Network Deployment Topology

In standard telephone cables of copper wires, voice signals use only a fraction of the available bandwidth. Like any other DSL technology, the VDSL2 technology utilizes the remaining capacity to carry the data and multimedia on the wire without interrupting the line's ability to carry voice signals.

This example depicts the typical VDSL2 network topology deployed using NFX device.

A VDSL2 link between network devices is set up as follows:

1. Connect an end-user device such as a LAN, hub, or PC through an Ethernet interface to the customer premises equipment (CPE) (for example, an NFX device).
2. Connect the CPE to a DSLAM.
3. The VDSL2 interface uses either Gigabit Ethernet or fiber as second mile to connect to the Broadband Remote Access Server (B-RAS) as shown in [Figure 18 on page 133](#).
4. The ADSL interface uses either Gigabit Ethernet (in case of IP DSLAM) as the "second mile" to connect to the B-RAS or OC3/DS3 ATM as the second mile to connect the B-RAS as shown in [Figure 19 on page 133](#).

NOTE: The VDSL2 technology is backward compatible with ADSL2 and ADSL2+. VDSL2 provides an ADSL2 and ADSL2+ interface in an ATM DSLAM topology and provides a VDSL2 interface in an IP or VDSL DSLAM topology.

The DSLAM accepts connections from many customers and aggregates them to a single, high-capacity connection to the Internet.

Figure 18 on page 133 shows a typical VDSL2 network topology.

Figure 18: Typical VDSL2 End-to-End Connectivity and Topology Diagram

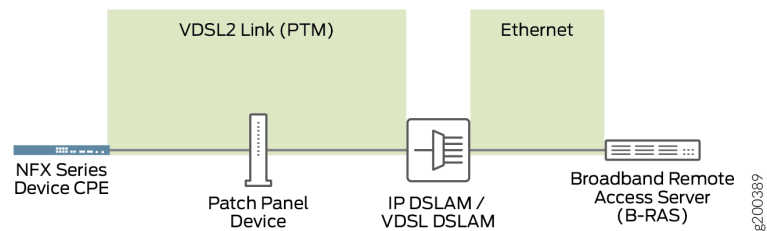
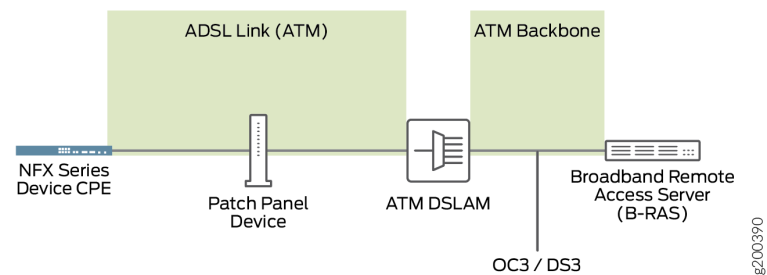


Figure 19 on page 133 shows a backward-compatible ADSL topology using ATM DSLAM.

Figure 19: Backward-Compatible ADSL Topology (ATM DSLAM)



VDSL2 Interface Support on NFX Series Devices

IN THIS SECTION

- [VDSL2 Interface Compatibility with ADSL Interfaces | 135](#)
- [VDSL2 Interfaces Supported Profiles | 135](#)

The VDSL2 interface is supported on the NFX Series devices listed in [Table 9 on page 134](#). (Platform support depends on the Junos OS release in your installation.)

Table 9: VDSL2 Annex A and Annex B Features

Features	POTS
Devices	CPE-SFP-VDSL2
Supported annex operating modes	Annex A and Annex B*
Supported Bandplans	Annex A 998 Annex B 997 and 998
Supported standards	ITU-T G.993.2 and ITU-T G.993.5 (VDSL2)
Used in	North American network implementations
ADSL backward compatibility	G 992.3 (ADSL2) G 992.5 (ADSL2+)

NOTE: Only one CPE-SFP-VDSL2 device is supported at a time.

VDSL2 Interface Compatibility with ADSL Interfaces

VDSL2 interfaces on NFX Series devices are backward compatible with most ADSL2 and ADSL2+ interface standards. The VDSL2 interface uses Ethernet in the First Mile (EFM) mode or Packet Transfer Mode (PTM) and uses the named interface ge-0/0/10 and ge-0/0/11.

NOTE:

- The VDSL2 interface has backward compatibility with ADSL2 and ADSL2+.
- It requires around 60 seconds to switch from VDSL2 to ADSL2 and ADSL2+ or from ADSL2 and ADSL2+ to VDSL2 operating modes.

VDSL2 Interfaces Supported Profiles

A profile is a table that contains a list of pre-configured VDSL2 settings. [Table 10 on page 135](#) lists the different profiles supported on the VDSL2 interfaces and their properties.

Table 10: Supported Profiles on the VDSL2 Interfaces

Profiles	Data Rate
8a	50
8b	50
8c	50
8d	50
12a	68
12b	68
17a	100

Table 10: Supported Profiles on the VDSL2 Interfaces *(Continued)*

Profiles	Data Rate
Auto	Negotiated (based on operating mode)

Example: Configuring VDSL SFP Interface on NFX250 Devices

IN THIS SECTION

- [Requirements | 136](#)
- [Overview | 136](#)
- [Configuration | 137](#)
- [Results | 139](#)

Requirements

This example uses the following hardware and software components:

- NFX250 device running Junos OS Release 15.1X53-D495.

Overview

In this example, you are configuring VDSL SFP interface on an NFX250 device with the following configurations:

- Physical interface - **ge-0/0/11**
- Virtual network function (VNF) - **nfx250-a-vsrx1**
- Memory size - **4194304**
- VDSL SFP options - **profile auto and carrier auto**

To configure VDSL SFP interface on NFX250 devices, you must configure JDM, vSRX, and vJunos0.

NOTE: Ensure that connectivity to the host is not lost during the configuration process.

Configuration

IN THIS SECTION

- [Procedure | 137](#)

Procedure

Step-by-Step Procedure

To configure VDSL SFP interfaces on NFX250 devices:

1. Connect to the host.

```
user@host> configure
[edit]
user@host#
```

2. Allocate hugepages:

```
user@host# run show system visibility memory
user@host# set system memory hugepages size 1024 count 5
```

Reboot the device.

3. Create VLANs using VLAN IDs:

```
user@host# set host-os vlans xdsl-test vlan-id 50
user@host# set host-os vlans vlan130 vlan-id 130
user@host# set host-os vlans vlan131 vlan-id 131
user@host# set host-os vlans vlan132 vlan-id 132
```

4. Allocate resources for a VNF:

```

user@host# set virtual-network-functions nfx250-a-vsrx1 image /var/public/media-vsrx-
vmdisk-15.1-2018-04-24.0_DEV_X_151_X49.qcow2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu 0 physical-cpu 2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu 1 physical-cpu 6
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu count 2
user@host# set virtual-network-functions nfx250-a-vsrx1 virtual-cpu features hardware-virtualization
user@host# set virtual-network-functions nfx250-a-vsrx1 no-default-interfaces
user@host# set virtual-network-functions nfx250-a-vsrx1 memory size 4194304
user@host# set virtual-network-functions nfx250-a-vsrx1 memory features hugepages
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth0 management out-of-band

```

5. Map VNF interfaces to NFV backplane:

```

user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth1 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth1 mapping vlan members vlan130
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan130
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan131
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members vlan132
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan members xdsl-test
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth2 mapping vlan native-vlan-id 50
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth3 mapping vlan mode trunk
user@host# set virtual-network-functions nfx250-a-vsrx1 interfaces eth3 mapping vlan members xdsl-test

```

6. Configure the Junos Control Plane (JCP):

```

user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching interface-mode trunk
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members xdsl-test
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan130
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan131
user@host# set interfaces sxe-0/0/0 unit 0 family ethernet-switching vlan members vlan132
user@host# set interfaces ge-0/0/11 native-vlan-id 50
user@host# set interfaces ge-0/0/11 dsl-sfp-options vdsl-options profile auto
user@host# set interfaces ge-0/0/11 dsl-sfp-options vdsl-options carrier auto
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching interface-mode trunk
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members xdsl-test
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan130
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan131

```

```
user@host# set interfaces ge-0/0/11 unit 0 family ethernet-switching vlan members vlan132
user@host# set vlans vlan130 vlan-id 130
user@host# set vlans vlan131 vlan-id 131
user@host# set vlans vlan132 vlan-id 132
user@host# set vlans xdsl-test vlan-id 50
```

7. Commit the configuration.

```
user@host# commit and-quit
user@host> exit
```

Results

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

3

CHAPTER

Monitoring and Troubleshooting JDM

Configuring SNMP on JDM | 141

Managing the Log Files and Core Files | 144

Recovering the Root Password for NFX250 | 147

Configuring SNMP on JDM

IN THIS SECTION

- [Configuring SNMP Community | 141](#)
- [Configuring SNMP System Parameters | 141](#)
- [Configuring SNMP v3 | 142](#)
- [Configuring SNMP Traps | 143](#)
- [Querying SNMP MIBs | 143](#)
- [Managing Traps | 144](#)

There are several SNMP-enabled components in NFX (JDM, hypervisor, and so on). This topic discusses the SNMP implementation of JDM and hypervisor. For JCP, see the Junos documentation. On the NFX250 platform, JDM plays the role of the SNMP agent and at the same time it acts as an SNMP proxy for the hypervisor (host OS). When SNMP is configured in JDM, hypervisor also takes the same SNMP configuration. By default, SNMP is disabled on disaggregated Junos OS platforms. To enable SNMP, you must include the SNMP configuration statements at the `[edit snmp]` hierarchy level. This section describes:

Configuring SNMP Community

To configure SNMP community:

Specify a name for the SNMP community:

```
user@jdm# set snmp community community
```

Configuring SNMP System Parameters

To configure SNMP system parameters:

1. Set the system name:

```
user@jdm# set snmp name name
```

2. Enter a description for the system being managed:

```
user@jdm# set snmp description description
```

3. Specify the location of the system:

```
user@jdm# set snmp location location
```

4. Specify the name of the contact person:

```
user@jdm# set snmp contact contact
```

Configuring SNMP v3

In contrast to SNMP version 1 (SNMPv1) and SNMP version 2 (SNMPv2), SNMP version 3 (SNMPv3) supports authentication and encryption. SNMPv3 uses the user-based security model (USM) for message security and the view-based access control model (VACM) for access control. USM specifies authentication and encryption. VACM specifies access-control rules. To configure local engine information for the user-based security model (USM) with Secure Hash Algorithm (SHA) as the authentication type for the SNMPv3 user, enter the command:

```
user@jdm# set snmp v3 usm local-engine user username authentication-sha authentication-password  
authentication-password
```

To configure local engine information for the USM with MD5 as the authentication type for the SNMPv3 user, enter the command:

```
user@jdm# set snmp v3 usm local-engine user username authentication-md5 authentication-password  
authentication-password
```

Configuring SNMP Traps

To configure SNMP traps, create a named group of hosts to receive the specified trap notifications. At least one trap group must be configured for SNMP traps to be sent:

```
user@jdm# set snmp trap-group group-name targets address
```

Querying SNMP MIBs

The NFX 250 platform supports querying SNMP MIBs on both, the JDM and the hypervisor. NFX MIBs are read-only, which means that the values can be read from the MIB but cannot be configured using SNMP.

The commands below are the queries on SNMP v1, SNMP v2 and SNMP v3. :

```
user@SNMP_server# snmpwalk -v 1 -c community-name jdm_ip-address oid
```

```
user@jdm# snmpwalk -v 2 -c community-name ip-address oid
```

```
user@jdm# snmpwalk -v3 -u username -l authNoPriv -a SHA -A password ip-address oid
```

To query the hypervisor, you need to provide an additional context name, which is the user name appended by -host:

```
user@jdm# snmpwalk -v 1 -c community-name-host ip-address oid
```

```
user@jdm# snmpwalk -v 2 -c community-name-host ip-address oid
```

```
user@jdm# snmpwalk -v3 -u username-host -l authNoPriv -a SHA -A password ip-address oid
```


You can query libvirt MIBs only as a host:

```
user@jdm# snmpwalk -v 2c -c community-name-host ip-address oid
```

Managing Traps

The agent sends traps to notify the manager of significant events that occur on the device. To configure traps:

```
user@jdm# set snmp trap-group group-name targets ip-address
```

JDM traps are assigned the context **jdm**, and hypervisor traps are assigned the context **host**.

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

Managing the Log Files and Core Files

IN THIS SECTION

- [Viewing and Managing Centralized Log Files | 145](#)
- [Managing Core Files | 146](#)

Viewing and Managing Centralized Log Files

IN THIS SECTION

- [Enabling Centralized Logging | 145](#)
- [Viewing Log Messages | 146](#)

On a disaggregated Junos OS platform, a centralized logging server collects all system logs for all computing entities in the disaggregated Junos OS. Centralized logging simplifies device management by allowing users to view all log files for all disaggregation Junos entities in a single place.

NOTE: Syslog feature configuration is not supported on JDM.

This topic describes:

Enabling Centralized Logging

You can enable centralized logging on vSRX to view the log files on JDM.

1. Log in to vSRX:

```
root@jdm> ssh vsrx-hostname
```

2. Ensure that the fxp0 interface is assigned the management IP address:

```
root>show interfaces fxp0 terse
```

3. Assign a hostname to the vSRX instance:

```
root# set system host-name hostname
```

4. To specify JDM as the host for the central log file:

```
root# set system syslog host 192.168.1.254 any any
```

192.168.1.254 is the internal management IP address of JDM

You can specify filters for the messages that you want displayed. If you specify any any all the messages will be logged.

Viewing Log Messages

To view system log messages logged by VNFs and the system generated logs on JDM, use the command:

```
user@jdm> show log syslog
```

To view other log messages, use the command:

```
user@jdm> show log <filename>
```

NOTE: You cannot view JCP log files on JDM.

A log rotation process runs every 5 minutes. If the log file size is greater than 5 MB, then the log files get rotated. The log files are saved in the **/var/log** directory.

Managing Core Files

IN THIS SECTION

- [Viewing Core Files | 147](#)

This topic discusses how to view core files for a disaggregated Junos OS platform.

It contains the following sections:

Viewing Core Files

To view all core files from the JDM CLI, enter the **show system core-dumps** command:

```
/var/tmp/*core* :
drwxr-xr-x 2 root root 4096 Jan 17 19:01 corefiles
/var/crash/*core* :
-rw-r--r-- 1 root root 382853 Jan 1 00:54 jdm.jdmd.4899.1420073655.core.tgz
-rw-r--r-- 1 root root 372095 Jan 1 01:11 jdm.jdmd.5697.1420074677.core.tgz
-rw-r--r-- 1 root root 3141405 Jan 17 19:01 jdm.jdmd.6875.1421521308.core.tgz
-rw-r--r-- 1 root root 18440 Jan 17 14:24 jdm.test.19278.1421504682.core.tgz
```

To view all core files displayed in JDM, open the core files using Unix commands. The core files are stored in the **/var/tmp/corefiles/** directory on Hypervisor.

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

Recovering the Root Password for NFX250

The root password helps to prevent unauthorized users from making changes to your network. If you forget the root password, you can use the password recovery procedure to reset the root password.

NOTE: You can recover the root password only if the password recovery option is enabled. To enable password recovery, use the following commands:

```
user@jdm# set system password-recovery
user@jdm# commit
```

You need console access to the device to recover the root password.

To recover the root password:

1. Power off the device by switching off the AC power outlet of the device. Or, if necessary, pull the power cords out of the device's power supplies.
2. Plug one end of the Ethernet rollover cable supplied with the device into the RJ-45-to-DB-9 serial port adapter supplied with the device.
3. Plug the RJ-45-to-DB-9 serial port adapter into the serial port on the management device.
4. Connect the other end of the Ethernet rollover cable to the console port on the device.
5. On the management device, start any asynchronous terminal emulation application (such as Microsoft Windows HyperTerminal), and select the appropriate COM port to use (for example, COM1).
6. Configure the port settings as follows:
 - Bits per second—9600
 - Data bits—8
 - Parity—None
 - Stop bits—1
 - Flow control—None
7. Power on the NFX250 by switching on the AC power outlet the device is plugged into. Or, if necessary, power it on by plugging the power cords into the device's power supply.
The terminal emulation screen on your management device displays the device's boot sequence.
8. When the system prompts you to press the Esc key, start pressing the Down Arrow key.

```

Checking Primary BIOS code integrity...Passed!
Press Esc for boot options
ME is in normal operational state
Booting HDD00.1 (TS512GMTS600)...

Secure boot is enforced
Welcome to GRUB!

Secure Grub2 Diskboot

```

9. Continue to press the Down Arrow key until you see the following output. Select **Juniper Linux - Debug**.

```

GNU GRUB version 2.02-juniper/re1_v4-
-----
Juniper Linux

```

```
Juniper Linux - Debug
Juniper Linux - Recovery
```

10. Press the Enter key until you see the following message on your screen. Select **Reset JDM root password**.

```
Hit [Enter] immediately to interrupt booting...
```

```
Select one of the following options:
```

1. Reset JDM root password.
2. JDM shell for maintenance.
3. Reboot the system.

```
Selection [1-3]:
```

11. Enter the new password and reboot the device.

```
Selection [1-3]:1
```

```
Enter new UNIX password:
```

```
Retype new UNIX password:
```

```
Authentication tokens updated successfully
```

```
Reboot (y/n)? Y
```

After the device reboots, you can login with the new password.

RELATED DOCUMENTATION

[NFX250 Overview | 2](#)

[JDM Architecture Overview | 6](#)

[JDM CLI Overview | 18](#)

4

CHAPTER

Virtual Network Functions Configuration Statements and Operational Commands

[cross-connect](#) | 152

[features](#) | 155

[host-os forwarding-options analyzer](#) | 156

[hugepages](#) | 158

[image](#) | 160

[init-descriptor](#) | 162

[interfaces](#) | 163

[mac-address](#) | 166

[mapping](#) | 167

[memory](#) | 169

[mtu](#) | 171

[no-autostart](#) | 173

[offloads](#) | 174

[pci-address](#) | 176

[size](#) | 177

[storage](#) | 178

[type](#) | 181

[virtual-cpu](#) | 182

[virtual-network-functions](#) | 184

[vjunos0](#) | 189

[vnf-name](#) | 191

[show virtual-network-functions](#) | 195

[show vlans](#) | 201

cross-connect

IN THIS SECTION

- [Syntax | 152](#)
- [Hierarchy Level | 153](#)
- [Description | 154](#)
- [Options | 154](#)
- [Required Privilege Level | 154](#)
- [Release Information | 154](#)

Syntax

```
cross-connect {
    cross-connect-name {
        physical-interface {
            hsxe0 {
                vlan-id vlan-id;
            }
            hsxe1 {
                vlan-id vlan-id;
            }
        }
        virtual-network-function vnf-name {
            interface interface-name;
            vlan-id vlan-id;
        }
    }
    push-pop-cross-connect-name {
        virtual-network-function vnf-name {
            interface interface-name;
            vlan-id vlan-id;
        }
        virtual-network-function vnf-name {
            interface interface-name;

```

```

    }
}
swap-cross-connect-name {
    virtual-network-function vnf-name {
        interface interface-name;
        vlan-id vlan-id;
    }
    virtual-network-function vnf-name {
        interface interface-name;
        vlan-id vlan-id;
    }
}
unconditional-cross-connect-name {
    virtual-network-function vnf-name {
        interface interface-name;
    }
    virtual-network-function vnf-name {
        interface interface-name;
    }
}
vlan-based-cross-connect-name {
    virtual-network-function vnf-name {
        interface interface-name;
        vlan-id vlan-id;
    }
    virtual-network-function vnf-name {
        interface interface-name;
        vlan-id vlan-id;
    }
}
}

```

Hierarchy Level

[edit]

Description

Connects any two VNF interfaces, VLANs on physical interfaces such as hsxe0 and hsxe1, and also provides features such as unconditional cross-connect, VLAN-based cross-connect, cross-connect of native VLAN traffic, and cross-connect with operations such as PUSH, POP, and SWAP.

Options

<code>virtual-network-functions <i>vnf-name</i></code>	Name of the VNF instance.
<code><i>cross-connect-name</i></code>	Name of the cross-connect.
<code>vlan-id</code>	Virtual LAN identifier.

Required Privilege Level

set—To view this statement in the configuration.
 routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D47.

RELATED DOCUMENTATION

[Configuring Cross-connect](#) | 105

features

IN THIS SECTION

- [Syntax | 155](#)
- [Hierarchy Level | 155](#)
- [Description | 155](#)
- [Options | 156](#)
- [Required Privilege Level | 156](#)
- [Release Information | 156](#)

Syntax

```
features {  
    hugepages;  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Displays the supported features of a VNF.

Options

features	Features of a VNF.
hugepages	Option to support memory pages with a size of 2 MB and 1 GB.

Required Privilege Level

routing—To view this statement in the configuration.
 routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

host-os forwarding-options analyzer

IN THIS SECTION

- [Syntax | 157](#)
- [Hierarchy Level | 157](#)
- [Description | 158](#)
- [Options | 158](#)

- [Required Privilege Level | 158](#)
- [Release Information | 158](#)

Syntax

```
forwarding-options {  
  analyzer analyzer-instance-name {  
    input {  
      ingress {  
        virtual-network-function vnf-name {  
          interface interface-name;  
        }  
      }  
      egress {  
        virtual-network-function vnf-name {  
          interface interface-name;  
        }  
      }  
    }  
    output {  
      virtual-network-function analyzer-vnf-name;  
      interface interface-name;  
    }  
  }  
}
```

Hierarchy Level

[edit]

Description

Configures an analyzer for port mirroring and configures port mirroring for either ingress or egress traffic of a VNF interface to an analyzer VNF.

Options

virtual-network-functions <i>vnf-name</i>	Name of the VNF instance.
<i>analyzer-instance-name</i>	Name of the analyzer.
ingress	Traffic that enters the port.
egress	Traffic that leaves the port.

Required Privilege Level

set host-os—To view this statement in the configuration.
routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

hugepages

IN THIS SECTION

- [Syntax | 159](#)
- [Hierarchy Level | 159](#)

- [Description | 159](#)
- [Required Privilege Level | 159](#)
- [Release Information | 159](#)

Syntax

```
hugepages;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

An option to support of 2 MB and 1 GB size memory pages.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

image

IN THIS SECTION

- [Syntax | 160](#)
- [Hierarchy Level | 160](#)
- [Description | 161](#)
- [Options | 161](#)
- [Required Privilege Level | 161](#)
- [Release Information | 161](#)

Syntax

```
image {  
    file-path;  
    bus-type [ide | virtio];  
    image-type [qcow2 | raw];  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Specify the VNF image source file. VNF image is virtual hard disk, which contains the bootable file-system for the VNF.

Options

<i>file-path</i>	Path of the image source file.
<i>image-type</i>	Format of the image. Default value is qcow2.
<i>bus-type</i>	Type of the system bus. Default value is virtio.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

init-descriptor

IN THIS SECTION

- [Syntax | 162](#)
- [Hierarchy Level | 162](#)
- [Description | 162](#)
- [Options | 163](#)
- [Required Privilege Level | 163](#)
- [Release Information | 163](#)

Syntax

```
init-descriptor file-path;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Create an XML descriptor file to launch a VNF. You can launch a VNF by configuring the VNF name, and specifying either the path to the XML descriptor file or to an image.

Options

file-path Path of the init-descriptor XML file.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

interfaces

IN THIS SECTION

- [Syntax | 164](#)
- [Hierarchy Level | 164](#)
- [Description | 164](#)
- [Options | 165](#)
- [Required Privilege Level | 165](#)
- [Release Information | 165](#)

Syntax

```

interfaces interface-name {
  pci-address pci-address;
  mac-address mac-address;
  mtu size;
  mapping {
    hsxe0 {
      virtual-function {
        vlan-id vlan-id;
      }
    }
    hsxe1 {
      virtual-function {
        vlan-id vlan-id;
      }
    }
    vlan {
      members vlan-name;
      mode [access | trunk];
      native-vlan-id vlan-id;
    }
  }
}

```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Configure Virtual Network Functions (VNF) interfaces on platforms running disaggregated Junos OS.

Options

<i>interface-name</i>	Name of the VNF interfaces.
<i>mac-address</i>	MAC address of the VNF interfaces.
<i>mtu</i>	Maximum transfer unit (MTU) size of packets in bytes.
<i>pci-address</i>	Target PCI address of the VNF interfaces.
<i>vlan-id</i>	SR-IOV virtual function to use to attach a VNF to a physical interface.
<i>vlan members</i>	Membership for this interface.
<i>native-vlan-id</i>	Virtual LAN identifier for untagged frames. For example, 1...4095
<i>vlan-name</i>	Name of the VLAN members.

Required Privilege Level

interface—To view this statement in the configuration.

interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D50.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

mac-address

IN THIS SECTION

- [Syntax | 166](#)
- [Hierarchy Level | 166](#)
- [Description | 166](#)
- [Required Privilege Level | 166](#)
- [Release Information | 167](#)

Syntax

```
mac-address mac-address;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

MAC address for the VNF interfaces.

Required Privilege Level

interface—To view this statement in the configuration.

interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D47.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

mapping

IN THIS SECTION

- [Syntax | 167](#)
- [Hierarchy Level | 168](#)
- [Description | 168](#)
- [Options | 168](#)
- [Required Privilege Level | 169](#)
- [Release Information | 169](#)

Syntax

```
mapping {  
  hsxe0 {  
    virtual-function {  
      vlan-id vlan-id;  
    }  
  }  
  hsxe1 {  
    virtual-function {
```



```

        vlan-id vlan-id;
    }
}
vlan {
    members vlan-name;
    mode [access | trunk];
    native-vlan-id vlan-id;
}
peer-interfaces peer-interface-name;
}

```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Mapping Virtual Network Functions (VNF) interfaces on platforms running disaggregated Junos OS.

Options

<i>vlan-id</i>	SR-IOV virtual function to use to attach a VNF to a physical interface.
<i>vlan members</i>	Membership for this interface.
<i>native-vlan-id</i>	Virtual LAN identifier for untagged frames. For example, 1...4095
<i>vlan-name</i>	Name of the VLAN members.
<i>peer-interface-name</i>	Name of the virtual peer interfaces that is mapped to a physical interface.

Required Privilege Level

interface—To view this statement in the configuration.

interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D50.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

memory

IN THIS SECTION

- [Syntax | 170](#)
- [Hierarchy Level | 170](#)
- [Description | 170](#)
- [Options | 170](#)
- [Required Privilege Level | 170](#)
- [Release Information | 171](#)

Syntax

```
memory {  
    size size;  
    features {  
        hugepages;  
    }  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Configure memory parameters for VNFs on a platform that is running a disaggregated Junos OS.

Options

memory *size* Amount of memory allocated to a VNF in kilobytes. The default size is 1 GB.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

mtu

IN THIS SECTION

- [Syntax | 171](#)
- [Hierarchy Level | 172](#)
- [Description | 172](#)
- [Options | 172](#)
- [Required Privilege Level | 172](#)
- [Release Information | 172](#)

Syntax

```
mtu size;
```

Hierarchy Level

```
[edit interfaces interface-name]
```

Description

Specify the maximum transmission unit (MTU) size for the media in bytes. MTU size can be either 1500 bytes or 2048 bytes.

Options

<i>size</i>	Size of the MTU
-------------	-----------------

Required Privilege Level

interface—To view this statement in the configuration.
interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D47.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

no-autostart

IN THIS SECTION

- [Syntax | 173](#)
- [Hierarchy Level | 173](#)
- [Description | 173](#)
- [Required Privilege Level | 173](#)
- [Release Information | 174](#)

Syntax

```
no-autostart;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Disable auto-start of VNF on the VNF configuration commit.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

offloads

IN THIS SECTION

- [Syntax | 174](#)
- [Hierarchy Level | 175](#)
- [Description | 175](#)
- [Options | 175](#)
- [Required Privilege Level | 175](#)
- [Release Information | 175](#)

Syntax

```
offloads {  
    disable;  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Offloads configuration for the VNF interface.

NOTE: This feature is available only in enhanced-orchestration mode.

Options

disable Disable the **offloads** option that offloads the configuration for the VNF interface.

Required Privilege Level

interface—To view this statement in the configuration.

interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D471.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions](#) | 65

[Managing the VNF Life Cycle](#) | 68

[show virtual-network-functions](#) | 195

pci-address

IN THIS SECTION

- [Syntax | 176](#)
- [Hierarchy Level | 176](#)
- [Description | 176](#)
- [Required Privilege Level | 176](#)
- [Release Information | 177](#)

Syntax

```
pci-address pci-address;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

PCI address for the VNF interfaces.

Required Privilege Level

interface—To view this statement in the configuration.

interface-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D50.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

size

IN THIS SECTION

- [Syntax | 177](#)
- [Hierarchy Level | 177](#)
- [Description | 178](#)
- [Required Privilege Level | 178](#)
- [Release Information | 178](#)

Syntax

```
size size;
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Size of the memory in kilobytes.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

storage

IN THIS SECTION

- [Syntax | 179](#)
- [Hierarchy Level | 179](#)
- [Description | 179](#)
- [Options | 180](#)
- [Required Privilege Level | 180](#)
- [Release Information | 180](#)

Syntax

```
storage device-name {  
  type {  
    cdrom {  
      source {  
        file filename;  
      }  
    }  
    disk {  
      bus-type [ide | virtio];  
      file-type [qcow2 | raw];  
      source {  
        file filename;  
      }  
    }  
    usb {  
      source {  
        file filename;  
      }  
    }  
  }  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Configure storage parameters on VNFs.

Options

storage *device-name* Name of the storage device. For example, hda, hdb, sdb, or vdb.

type *disk-type* Type of disk. For example, cdrom, usb, or disk.

source *file-name* Path of the source file of the storage device.

NOTE: The source file to be attached to the VNF must be located in the **/var/third-party/** directory and must have read and write permissions for all.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

type

IN THIS SECTION

- [Syntax | 181](#)
- [Hierarchy Level | 181](#)
- [Description | 181](#)
- [Options | 182](#)
- [Required Privilege Level | 182](#)
- [Release Information | 182](#)

Syntax

```
type {  
    linux-container | virtual-machine;  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Type of the VNF.

Options

linux-container	The VNF type is Linux container.
virtual-machine	The VNF type is virtual machine.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

virtual-cpu

IN THIS SECTION

- [Syntax | 183](#)
- [Hierarchy Level | 183](#)
- [Description | 183](#)
- [Options | 183](#)

- [Required Privilege Level | 184](#)
- [Release Information | 184](#)

Syntax

```
virtual-cpu {
    virtual-cpu-number {
        physical-cpu number | range;
    }
    count number;
    features {
        hardware-virtualization;
    }
}
```

Hierarchy Level

[edit virtual-network-functions]

Description

Specify the number of virtual CPUs the VNF can use. By default, a VNF is assigned one virtual CPU, which is independent of any specific physical CPU.

Options

- | | |
|--|--|
| virtual-cpu count <i>number</i> | Number of virtual CPUs. For example, 2. |
| virtual-cpu features | Features of the virtual cpu. For example, hardware-virtualization. |

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D40.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

virtual-network-functions

IN THIS SECTION

- [Syntax | 185](#)
- [Hierarchy Level | 185](#)
- [Description | 188](#)
- [Options | 188](#)
- [Required Privilege Level | 189](#)
- [Release Information | 189](#)

Syntax

```
virtual-network-functions vnf-name;
```

Hierarchy Level

```
virtual-network-functions {  
    ipsec-nm {  
        interfaces {  
            heth1 | heth2 {  
                mapping {  
                    vlan {  
                        members vlan-name;  
                    }  
                }  
            }  
        }  
    }  
    vjunos0 {  
        interfaces {  
            em1 {  
                mapping {  
                    vlan {  
                        members vlan-name;  
                    }  
                }  
            }  
        }  
    }  
    vmx {  
        image filepath;  
        virtual-cpu {  
            count virtual-cpu-count;  
            features;  
        }  
        memory size size;  
        storage {  
            vdc {
```

```

        type type;
        source file-path;
    }
    vdb {
        type type;
        source file-path;
    }
}
vsrx {
    vnf-name {
        type {
            linux-container | virtual-machine;
        }
        image {
            file-path;
            bus-type [ide | virtio];
            image-type [qcow2 | raw];
        }
        init-descriptor file-path;
        memory {
            size size;
            features {
                hugepages ;
            }
        }
    }
    no-autostart;
    storage device-name {
        type {
            cdrom {
                source {
                    file filename;
                }
            }
            disk {
                bus-type [ide | virtio];
                file-type [qcow2 | raw];
                source {
                    file filename;
                }
            }
        }
        usb {
            source {
                file filename;
            }
        }
    }
}

```

```

        }
    }
}
virtual-cpu {
    virtual-cpu-number {
        physical-cpu number | range;
    }
    count number;
    features {
        hardware-virtualization;
    }
}
interfaces interface-name {
    pci-address pci-address;
    mapping {
        hsxe0 {
            virtual-function {
                vlan-id vlan-id;
            }
        }
        hsxe1 {
            virtual-function {
                vlan-id vlan-id;
            }
        }
        vlan {
            members vlan-name;
            mode [access | trunk];
            native-vlan-id vlan-id;
        }
        peer-interfaces peer-interface-name;
    }
    management {
        internal;
        out-of-band;
    }
}
}

```

Description

Create an instance of a virtual network function (VNF) on platforms that run disaggregated Junos OS software.

NOTE:

- Creating a VNF instance fails if the resources required by the VNF are not available in the system.
- If you use init-descriptor to define a VNF by specifying and setting different values for the virtual CPU count or memory and later if you delete the virtual cpu count, the system restores the value to a default value of 1 for vCPU and 1GB for memory.
- You can enable the VNF options such virtual-cpu-hardware-virtualization (vmx), hugepages, image-type, and image-bus-type only when you define the VNF initially. You cannot enable or disable the VNF options after committing the VNF configuration. To enable or disable the VNF options, you must delete the VNF configuration and re-configure with the VNF options.

Options

virtual-network-functions <i>vnf-name</i>	Name of the VNF instance. It is mandatory to provide one of the options: init-descriptor or image.
<i>file-path</i>	Path of the source file.
<i>number / range</i>	Number or a range of physical CPUs. For example, 2 or 2...5.
<i>interface-name</i>	Name of the VNF interface, which can range from eth0 to eth9. You can configure eth0 and eth1 interfaces and can assign VLAN IDs. To configure eth0 and eth1 interfaces, you must configure no-default-interfaces option.
<i>physical-interface-name</i>	Name of the physical interface to which the VNF interface is attached.
<i>vlan-id</i>	SR-IOV virtual function to use to attach a VNF to a physical interface.
<i>native-vlan-id</i>	Virtual LAN identifier for untagged frames. For example, 1...4095
<i>vlan-name</i>	Name of the VLAN members.

<i>size</i>	Amount of memory allocated to a VNF in kilobytes. The default size is 1 GB.
<i>device-name</i>	Name of the storage device.
<i>file-name</i>	Name of the source file of the storage device.
<i>peer-interface-name</i>	Name of the virtual peer interfaces that is mapped to a physical interface.
<i>management</i>	VNF interface management configuration.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

vjunos0

IN THIS SECTION

● [Syntax | 190](#)

● [Hierarchy Level | 190](#)

- [Description | 190](#)
- [Options | 191](#)
- [Required Privilege Level | 191](#)
- [Release Information | 191](#)

Syntax

```
vjunos0 {  
  interfaces {  
    em1 {  
      mapping {  
        vlan {  
          members vlan-name;  
        }  
      }  
    }  
  }  
}
```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

An option that enables or disables the vjunos virtual network function if the vjunos0 option is configured in the system.

Options

interfaces	Name of the interface. For example, em1.
vlan members <i>vlan-name</i>	Name of the VLAN members.

Required Privilege Level

routing—To view this statement in the configuration.
 routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions | 65](#)

[Managing the VNF Life Cycle | 68](#)

[show virtual-network-functions | 195](#)

vnf-name

IN THIS SECTION

- [Syntax | 192](#)
- [Hierarchy Level | 193](#)
- [Description | 194](#)
- [Options | 194](#)

- Required Privilege Level | 194
- Release Information | 194

Syntax

```

vnf-name {
  type {
    linux-container | virtual-machine;
  }
  image {
    file-path;
    bus-type [ide | virtio];
    image-type [qcow2 | raw];
  }
  init-descriptor file-path;
  memory {
    size size;
    features {
      hugepages;
    }
  }
  no-autostart;
  storage device-name {
    type {
      cdrom {
        source {
          file filename;
        }
      }
      disk {
        bus-type [ide | virtio];
        file-type [qcow2 | raw];
        source {
          file filename;
        }
      }
      usb {
        source {

```

```

        file filename;
    }
}
}
}
virtual-cpu {
    virtual-cpu-number {
        physical-cpu number | range;
    }
    count number;
    features {
        hardware-virtualization;
    }
}
interfaces interface-name {
    pci-address pci-address;
    mapping {
        hsxe0 {
            virtual-function {
                vlan-id vlan-id;
            }
        }
        hsxe1 {
            virtual-function {
                vlan-id vlan-id;
            }
        }
        vlan {
            members vlan-name;
            mode [access | trunk];
            native-vlan-id vlan-id;
        }
    }
}
}
}

```

Hierarchy Level

```
[edit virtual-network-functions]
```

Description

Name of the virtual network function.

Options

- interfaces** Name of the interface. For example, em1.
- no-autostart** An option to disable auto-start of VNF on the VNF configuration commit.
- pci-address** An option to specify PCI address for the VNF interfaces.
- mapping** An option to map VNF interfaces on platforms running disaggregated Junos OS.
- virtual-cpu** An option to specify the number of virtual CPUs the VNF can use. By default, a VNF is assigned one virtual CPU, which is independent of any specific physical CPU.

Required Privilege Level

routing—To view this statement in the configuration.

routing-control—To add this statement to the configuration.

Release Information

Statement introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions](#) | 65

[Managing the VNF Life Cycle](#) | 68

[show virtual-network-functions](#) | 195

show virtual-network-functions

IN THIS SECTION

- [Syntax | 195](#)
- [Description | 195](#)
- [Options | 196](#)
- [Required Privilege Level | 196](#)
- [Output Fields | 196](#)
- [Sample Output | 197](#)
- [Sample Output | 198](#)
- [Sample Output | 199](#)
- [Sample Output | 199](#)
- [Release Information | 200](#)

Syntax

```
show virtual-network-functions
show virtual-network-functions vnf-name
show virtual-network-functions ipsec-nm
show virtual-network-functions vjunos0
```

Description

Display Virtual Network Function (VNF) information.

Options

- vnf-name*** (Optional) Display information for a specific VNF.
- ipsec-nm** (Optional) Display information of the system VNF IPsec.
- vjunos0** (Optional) Display information of the system VNF vjunos0.
- brief** (Optional) Display brief output.

NOTE: This is the default option.

- detail** (Optional) Display detailed output.

Required Privilege Level

view

Output Fields

Table 11 on page 196 describes the output fields for the `show virtual-network-functions` command. Output fields are listed in the approximate order in which they appear.

Table 11: show virtual-network functions Output Fields

Field Name	Field Description
ID	ID of the VNF
Name	Name of the VNF
State	Status of the VNF. Possible values are Running, Shutdown, or Undefined.

Table 11: show virtual-network-functions Output Fields (Continued)

Field Name	Field Description
Liveliness	Indicates whether or not the IP address of the VNF is reachable.
IP Address	IP address of the VNF
VCPUs	Number of virtual CPUs
Maximum Memory	Maximum amount of memory available to the VNF
Used Memory	Amount of memory used by the VNF
Used 1G Hugepages	The number of used 1G hugepages.
Used 2M Hugepages	The number of used 2M hugepages

Sample Output

show virtual-network-functions vjunos0

```
user@host> show virtual-network-functions vjunos0
```

ID	Name	State	Liveliness
2	vjunos0	Running	alive

show virtual-network-functions vjunos0 detail

```
user@host> show virtual-network-functions vjunos0 detail
```

Virtual Network Function Information

```
Id:          3
Name:        vjunos0
State:       Running
Liveliness:  Up
IP Address:  192.0.2.2
VCPUs:       1
Maximum Memory: 1000448 KiB
Used Memory:  1000448 KiB
Used 1G Hugepages: 0
Used 2M Hugepages: 0

Block Devices
-----
Target | Source
-----
vda    | /junos/images/0/vjunos.img
vdb    | /junos/images/0/vjunos-data.img
hdc    | /junos/images/shared/swap-disk.img
vdd    | /junos/images/0/vjunos-platform.img
vde    | /junos/images/0/metadata-usb-re.img
```

Sample Output

show virtual-network-functions vsrx

```
user@host> show virtual-network-functions vsrx

ID      Name                               State    Liveliness
-----
67      vsrx                                Running  alive
```

Sample Output

show virtual-network-functions vmx

```
user@host> show virtual-network-functions vmx
```

ID	Name	State	Liveliness

3	vjunos0	Running	alive
10	vm x		Running alive
11341	jdm		Running alive

show virtual-network-functions

```
user@host> show virtual-network-functions
```

ID	Name	State	Liveliness

3	vjunos0	Running	alive
1	LTE-VM	Running	down

Sample Output

show virtual-network-functions detail

```
user@host> show virtual-network-functions detail
```

Virtual Network Function Information	

Id:	3
Name:	vjunos0
State:	Running
Liveliness:	Up
IP Address:	192.0.2.2
VCPUs:	1
Maximum Memory:	1000448 KiB
Used Memory:	1000448 KiB
Used 1G Hugepages:	0

Used 2M Hugepages: 0

Block Devices

Target	Source
vda	/junos/images/0/vjunos.img
vdb	/junos/images/0/vjunos-data.img
hdc	/junos/images/shared/swap-disk.img
vdd	/junos/images/0/vjunos-platform.img
vde	/junos/images/0/metadata-usb-re.img

Virtual Network Function Information

Id: 1

Name: LTE-VM

State: Running

Liveliness: Down

IP Address: -

VCPUs: 2

Maximum Memory: 122880 KiB

Used Memory: 122880 KiB

Used 1G Hugepages: 0

Used 2M Hugepages: 0

Block Devices

Target	Source
vdb	/usr/share/juniper/LTE/lte_vm.latest

Release Information

Command introduced in Junos OS Release 15.1X53-D45.

RELATED DOCUMENTATION

[Understanding Virtual Network Functions](#) | 65

show vlans

IN THIS SECTION

- Syntax | 201
- Description | 201
- Options | 201
- Required Privilege Level | 202
- Output Fields | 202
- Sample Output | 203
- Release Information | 203

Syntax

```
show vlans vlan-name
```

Description

Display the details about the VLANs.

Options

<i>vlan-name</i>	Display information for a specific VLAN.
brief detail extensive terse	(Optional) Display the specified level of output.

Required Privilege Level

view

Output Fields

Table 12 on page 202 describes the output fields for the `show virtual-network-functions` command. Output fields are listed in the approximate order in which they appear.

Table 12: show virtual-network functions Output Fields

Field Name	Field Description
vlan-name	Display information for a specified VLAN
brief	Display brief output
detail	Display detailed output
extensive	Display extensive output
instance	Display information for a specified instance
interface	Name of interface for which to display table
logical-system	Name of logical systems

Sample Output

show vlans

```
root@jdm> show vlans
```

Routing instance	VLAN name	Tag	Interfaces
host-os	vlan100	100	vsrx1_eth6.0 vsrx2_eth6.0
host-os	vlan200-202-vlan-0200	200	vsrx1_eth7.0 vsrx2_eth7.0
host-os	vlan200-202-vlan-0202	202	vsrx1_eth7.0 vsrx2_eth7.0

Release Information

Command introduced in Junos OS Release 15.1X53-D40.