



DANOS-Vyatta edition

Disaggregated Network Operating System Version 2009a

Virtualization Support User Guide
October 2020

Contents

Chapter 1. Copyright Statement.....	1
Chapter 2. Preface.....	2
Document conventions.....	2
Chapter 3. About This Guide.....	4
Chapter 4. VNF platform Overview.....	5
VNF platform architecture.....	5
Integrated hypervisor overview.....	5
Guest life cycle overview.....	5
Chapter 5. Provisioning Guests Manually.....	7
Prerequisites.....	7
cpus and cpuset parameters.....	7
Defining CPUs and CPU sets for low-power systems.....	8
Provisioning a guest manually.....	8
Provisioning a guest with multiple virtual disks.....	10
Chapter 6. Provisioning Guests Automatically.....	12
Prerequisites.....	12
Provisioning a guest automatically.....	12
Using the force and save options.....	13
Supported libvirt domain XML elements and attributes.....	15
Valid memory element identifier and multiplier values.....	18
Sample XML snippet for the libvirt domain XML file.....	18
VRF overview.....	19
Chapter 7. Provisioning Guests Commands.....	20
add virtualization xml image.....	20
generate virtualization { random-mac uuid }.....	21
virtualization guest.....	21
virtualization guest cpus.....	22
virtualization guest cpuset.....	23
virtualization guest devices disk.....	24
virtualization guest devices disk boot-order.....	24

virtualization guest devices disk bus-type.....	25
virtualization guest devices disk device-type.....	26
virtualization guest devices disk source file.....	27
virtualization guest devices display address.....	28
virtualization guest devices display connection-type.....	29
virtualization guest devices display password.....	30
virtualization guest devices display port.....	31
virtualization guest memory.....	31
virtualization guest devices network.....	32
virtualization guest devices network mac-address.....	33
virtualization guest devices video memory.....	34
virtualization guest devices video model.....	34
virtualization guest uuid.....	35
Chapter 8. Connecting Guests to a VNF Platform Data Plane.....	37
Overview.....	37
Configuring a vhost interface.....	37
Configuring sFlow for vhost interface.....	38
Configuring QoS on a vhost interface.....	39
Chapter 9. Connecting Guests Commands.....	41
interfaces vhost sflow.....	41
interfaces vhost flow-monitoring aggregator.....	41
interfaces vhost flow-monitoring exporter.....	42
interfaces vhost flow-monitoring selector.....	43
interfaces vhost policy qos.....	43
Chapter 10. Upgrading Guests.....	45
Overview.....	45
Upgrading a guest by using a libvirt domain XML file.....	46
Upgrading multiple guest images for a guest.....	48
Upgrading a guest and keeping the old files.....	49
Chapter 11. Upgrading Guests Commands.....	52
upgrade virtualization guest images.....	52
upgrade virtualization xml.....	53
Chapter 12. Guest Lifecycle Commands.....	54

add virtualization guest snapshot description.....	54
copy virtualization image to.....	54
delete virtualization guest snapshot.....	55
delete virtualization image.....	56
download virtualization image from.....	56
move virtualization image to.....	57
restart virtualization guest.....	58
reset virtualization guest to snapshot.....	59
virtualization guest autostart.....	59
poweroff virtualization guest.....	60
start virtualization guest.....	60
show virtualization images.....	61
show virtualization status.....	61
show virtualization guest snapshots.....	62
upload virtualization image to.....	62
Chapter 13. Monitoring Guests.....	65
Monitoring guests overview.....	65
Probe type.....	65
Probe action overview.....	66
Logging overview.....	66
Configuring Probes.....	67
Configuring a guest ping probe.....	67
Configuring a guest HTTP probe.....	68
Configuring a guest HTTPS probe.....	69
Configuring a guest SNMP probe.....	70
Chapter 14. Monitoring Guests Commands.....	72
service probes debug.....	72
service probes probe.....	72
service probes probe type network target.....	73
service probes probe type network routing-instance.....	73
service probes probe interval.....	74
service probes probe retries.....	75
service probes probe type network ping.....	76

service probes probe type network http.....	76
service probes probe type network http path.....	77
service probes probe type network http port.....	77
service probes probe type network http status-code.....	78
service probes probe type network https.....	79
service probes probe type network https path.....	79
service probes probe type network https port.....	80
service probes probe type network https status-code.....	81
service probes probe type network https certificate-validation.....	82
service probes probe type network snmp.....	82
service probes probe type network snmp oid.....	83
service probes probe type network snmp value.....	84
service probes probe type network snmp v1.....	84
service probes probe type network snmp v2.....	85
service probes probe type network snmp v1 community.....	86
service probes probe type network snmp v2 community.....	86
service probes probe action virtualization guest action.....	87
service probes probe action virtualization guest startup-wait.....	88
show probes status.....	89
Chapter 15. Connecting Guests in a Service Chain.....	91
Service chaining overview.....	91
Configuring service chaining by using PBR.....	91
Configuring service chaining by using bridging.....	92
Configuring service chaining by using cross-connect.....	93
Configuring service chaining by using a hybrid model.....	94
Chapter 16. Service Chaining Commands.....	96
policy route pbr rule.....	96
protocols static table routenext-hop.....	96
interfaces bridge address.....	97
interfaces vhost address.....	98
interfaces dataplane address.....	99
interfaces vhost bridge-group bridge.....	99
interfaces dataplane bridge-group bridge.....	100

interfaces vhost vif vlan.....	101
interfaces vhost vif inner-vlan.....	102
interfaces dataplane xconnect dataplane.....	103
interfaces dataplane cross-connect vhost.....	103
interfaces vhost xconnect vhost.....	105
interfaces vhost xconnect dataplane.....	105
show policy.....	106
show policy route.....	107
show policy route table.....	108
Chapter 17. Troubleshooting.....	109
Inability to reprovision a configured guest automatically.....	109
Disk source image error when reprovisioning a guest automatically.....	109
Duplicate IP address error message when provisioning a guest automatically.....	110
Invalid XML file error when provisioning a guest automatically.....	110
Specific XML parsing errors when provisioning a guest automatically.....	110
Paused probe state in the show probe status command.....	112
Fatal probe state in the show probe status command.....	113
Show command failures.....	113

Chapter 1. Copyright Statement

© 2020 IP Infusion Inc. All Rights Reserved.

This documentation is subject to change without notice. The software described in this document and this documentation are furnished under a license agreement or nondisclosure agreement. The software and documentation may be used or copied only in accordance with the terms of the applicable agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's internal use without the written permission of IP Infusion Inc.

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900

<http://www.ipinfusion.com/>.

For support, questions, or comments via E-mail, contact:

support@ipinfusion.com.

Trademarks:

IP Infusion is a trademark of IP Infusion. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Use of certain software included in this equipment is subject to the IP Infusion, Inc. End User License Agreement at <http://www.ipinfusion.com/license>. By using the equipment, you accept the terms of the End User License Agreement.


Chapter 2. Preface


Document conventions


The document conventions describe text formatting conventions, command syntax conventions, and important notice formats used in this document.


Notes, cautions, and warnings

Notes, cautions, and warning statements may be used in this document. They are listed in the order of increasing severity of potential hazards.

 **Note:** A Note provides a tip, guidance, or advice, emphasizes important information, or provides a reference to related information.

 **Attention:** An Attention statement indicates a stronger note, for example, to alert you when traffic might be interrupted or the device might reboot.

 **CAUTION:** A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

 **DANGER:** A Danger statement indicates conditions or situations that can be potentially lethal or extremely hazardous to you. Safety labels are also attached directly to products to warn of these conditions or situations.

Text formatting conventions

Text formatting conventions such as boldface, italic, or Courier font are used to highlight specific words or phrases.

Format	Description
bold text	Identifies command names. Identifies keywords and operands.
<i>italic text</i>	Identifies emphasis. Identifies variables. Identifies document titles.
Courier font	Identifies CLI output. Identifies command syntax examples.

Command syntax conventions

Bold and italic text identify command syntax components. Delimiters and operators define groupings of parameters and their logical relationships.

Convention	Description
bold text	Identifies command names, keywords, and command options.
<i>italic text</i>	Identifies a variable.
[]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ x y z }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options.
x y	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, for example, passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, <i>member</i> [<i>member</i> ...].
\	Indicates a “soft” line break in command examples. If a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

Chapter 3. About This Guide

This guide describes how to provision, connect, upgrade, and monitor guests. This guide also describes service chaining and troubleshooting guests.

Chapter 4. VNF platform Overview

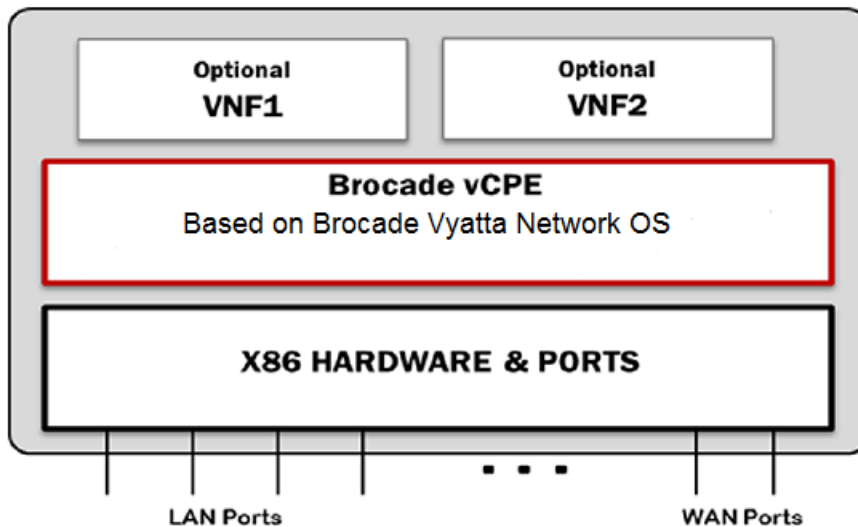
VNF platform architecture

VNF platform allows the virtualization of the hardware that is required to run your business and provides a set of value-added services, including network connectivity.

You can use VNF platform to run various virtualized network function (VNF) devices.

The following illustration presents the VNF platform architecture.

Figure 1. VNF platform architecture



The *DANOS-Vyatta edition* runs on top of VNF platform. You install the hypervisor image, and then create guests for various VNF roles.

Integrated hypervisor overview

The hypervisor is integrated with VNF platform and is deployed directly onto bare-metal hardware.

The hypervisor is based on Quick Emulator (QEMU) and libvirt. The hypervisor provides a limited set of features that are required for deploying VNFs or guests on top of VNF platform. You can control the hypervisor by using any of the standard interfaces, such as the CLI, NETCONF, REST API, and the configured API.

Guest life cycle overview

After you receive VNF platform, connect it to the network to follow and complete the ZeroTouch procedure. The ZeroTouch procedure enables VNF platform to install a boot

image and complete an initial configuration. ZeroTouch enables VNF platform to connect to the network management system (NMS).

Alternately, you can install the integrated hypervisor image on the computer system by using the `install image` command. You can then install a guest on VNF platform by using either automatic or manual methods. For more information on provisioning guests, refer to [Provisioning Guests Automatically](#) and [Provisioning Guests Manually](#).

After you have provisioned a guest on the hypervisor, you can use the CLI to run guest life cycle commands.

Some life cycle features are discussed in this guide. These features include the following:

- Guest snapshots—Enables you to take a snapshot of a guest and restore a guest to a previous state. For more information, refer to [Guest Lifecycle Commands](#).
- Guest upgrades— Enables you to upgrade a guest, including its image and resources. For more information, refer to [Upgrading Guests](#).
- Guest monitoring—Enables you to monitor a guest by using an SNMP, an HTTP, an HTTPS, or a ping probe. For more information, refer to [Monitoring Guests](#).
- Service chaining—Enables you to link a logical group of service functions to a packet or flow to realize a service. For more information, refer to [Connecting Guests in a Service Chain](#).

Chapter 5. Provisioning Guests Manually

Prerequisites

Installing a guest manually consists of installing the guest and then configuring the vhost interface for the guest.

Before proceeding with installing the guest manually, ensure that the following prerequisites are met:

- Copy the disk image of the guest in QCOW2 format to the `/var/lib/libvirt/images` folder.
- Run the `generate virtualization uuid` operational command to obtain a universally unique identifier (UUID) for the guest. For example:

```
tmpuser@node:~$ generate virtualization uuid
a885d67d-aa2f-4478-92f0-d50ae2018033
tmpuser@node:~$
```

- Run the `generate virtualization random-mac` operational mode command to obtain a MAC address for the guest. For example:

```
tmpuser@node generate virtualization random-mac
52:54:00:7a:ed:43
tmpuser@node:~$
```

cpus and cpuset parameters

A guest has two important parameters: `cpus` and `cpuset`.

cpus

The parameter specifies how many CPUs are used by a guest. For a guest installed with the router, the number of CPUs is usually four for optimum performance. For most other guests, one CPU is sufficient. For more information on assigning CPUs to a guest, refer to [virtualization guest cpus](#).

cpuset

The parameter specifies which CPUs are used. For a guest, the value of this parameter must point to those CPUs that are not used by the data plane and other guests. For more information on assigning CPU sets to a guest, refer to [virtualization guest cpuset](#).

When VNF platform is installed on low-power devices, guests may receive but become unable to transmit data. To avoid the inability to transmit data, provision guests so that they run on unique CPUs. After you install VNF platform, you must ensure that each guest is

assigned its own unique CPU and the VNF platform data plane includes CPU 0 among its CPU set.

Defining CPUs and CPU sets for low-power systems

Assume a eight-core system installed with VNF platform. Furthermore, the system is to be provisioned with two guests, one installed with the router. You must first provision the data plane with CPU 0 and then provision the guests with unique CPUs.

Perform the following steps to define values for `cpus` and `cpuset` for the data plane and guests of the system.

1. Determine the number of CPUs of the system by entering this command at the terminal prompt: `cat /proc/cpuinfo`
The result displays the details for the CPUs.

```
processor       : 7
vendor_id      : GenuineIntel
cpu family     : 6
model          : 77
model name     : Intel(R) Atom(TM) CPU C2758 @ 2.40GHz
```

2. To allow the data plane to use four CPUs, enter this command at the terminal prompt:
`set system default dataplane cpu-affinity 0-3`
3. Restart the system.
4. Display the CPU affinity set on the data plane by entering this command in configuration mode: `show system default dataplane cpu-affinity`
A result similar to the following is displayed:

```
cpu-affinity 0-3
```

The system used in this example has eight CPUs. The first four CPUs are used by the data plane, which enables the guests to use the next four CPUs. If you provision a guest that requires two CPUs, you can define the guest CPU set as 4-5. If you now provision a second guest, you must define the guest CPU set as 6 to make the CPU set unique from the data plane and the first guest.

Provisioning a guest manually

You can provision a guest manually by using a series of `set virtualization guest` commands.

The following example shows how to install and configure a guest with a remote display console connected to one network device and with the following parameters:

- Unique name of the guest: `a-guest`.

The name for a guest can be alphanumeric, consisting of uppercase letters, lowercase letters, numbers, the dash, and underscore . However, you cannot start a guest name with either the dash or underscore.

- Number of CPUs: `2`.

The maximum number of CPUs that can be configured is `8`.

- Amount of RAM: `2048 MB`.

The maximum memory that can be configured is `6144 MB`.

To install and configure a guest, perform the following steps in configuration mode.

Table 1. Provisioning a guest manually

Step	Command
Create a guest called <code>a-guest</code> .	<pre>tmpuser@node# set virtualization guest a-guest</pre>
Assign the UUID to the guest.	<pre>tmpuser@node# set virtualization guest a-guest uuid afbca013-f984-44bf-a294-2190771ab53f</pre>
Set the number of CPUs as <code>2</code> .	<pre>tmpuser@node# set virtualization guest a-guest cpus 2</pre>
Set the RAM as <code>2048 MB</code> .	<pre>tmpuser@node# set virtualization guest a-guest memory 2048</pre>
Set a virtual disk for the guest.	<pre>tmpuser@node# set virtualization guest a-guest devices disk 0 device-type disk</pre>
Set the path of the disk image as <code>/var/lib/libvirt/images/an-image.qcow2</code> .	<pre>tmpuser@node# set virtualization guest a-guest devices disk 0 source file /var/lib/libvirt/images/an-image.qcow2</pre>
Set the boot order of the disk as <code>1</code> .	<pre>tmpuser@node# set virtualization guest a-guest devices disk 0 boot-order 1</pre>
Set the display password as <code>lotr</code> .	<pre>tmpuser@node# set virtualization guest a-guest devices display password lotr</pre>
Set the display port as <code>5910</code> .	<pre>tmpuser@node# set virtualization guest a-guest devices display port 5910</pre>
Set the remote display access type as <code>vnc</code> .	<pre>tmpuser@node# set virtualization guest a-guest devices display connection-type vnc</pre>
Commit the changes.	<pre>tmpuser@node# commit</pre>
Display the result.	<pre>tmpuser@node# show virtualization virtualization { guest a-guest { cpus 2 devices { disk 0 { boot-order 1 device-type disk source { file /var/lib/libvirt/images/an-image.qcow2 } } display { connection-type vnc password ***** port 5910 } } } memory 2048 uuid afbca013-f984-44bf-a294-2190771ab53f }</pre>

Step	Command
	} }

Provisioning a guest with multiple virtual disks

You can provision a guest with multiple virtual disks specifying different images and different boot orders.

The following example shows how to install and configure a guest with two different virtual disks, `disk 1` and `disk 2`. The guest parameters are:

- Unique name of the guest: `a-guest`.
- Number of CPUs: `2`.
- Amount of RAM: `2048 MB`.
- Location of `image1` for `disk 1`: `/var/lib/libvirt/images/image1.qcow2`
- Location of `image2` for `disk 2`: `/var/lib/libvirt/images/image2.qcow2`

To install and configure a guest, perform the following steps in configuration mode.

Table 2. Provisioning a guest manually

Step	Command
Create a guest called <code>a-guest</code> .	<code>tmpuser@node# set virtualization guest a-guest</code>
Assign the UUID to the guest.	<code>tmpuser@node# set virtualization guest a-guest uuid b7bbf124-7314-21e2-819f-fe8119cd991</code>
Set the number of CPUs as <code>2</code> .	<code>tmpuser@node# set virtualization guest a-guest cpus 2</code>
Set the RAM as <code>2048 MB</code> .	<code>tmpuser@node# set virtualization guest a-guest memory 2048</code>
Set a virtual disk for the guest as <code>disk 1</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 1 device-type disk</code>
Set the path of the image for <code>disk 1</code> as <code>/var/lib/libvirt/images/image1.qcow2</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 1 source file /var/lib/libvirt/images/image1.qcow2</code>
Set the boot order for <code>disk 1</code> as <code>1</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 1 boot-order 1</code>
Set another virtual disk for the guest as <code>disk 2</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 2 device-type disk</code>
Set the path of the image for <code>disk 2</code> as <code>/var/lib/libvirt/images/image2.qcow2</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 2 source file /var/lib/libvirt/images/image2.qcow2</code>
Set the boot order for <code>disk 2</code> as <code>3</code> .	<code>tmpuser@node# set virtualization guest a-guest devices disk 2 boot-order 3</code>
Set the display password as <code>vyatta123</code> .	<code>tmpuser@node# set virtualization guest a-guest devices display password vyatta123</code>
Set the display port as <code>5911</code> .	<code>tmpuser@node# set virtualization guest a-guest devices display port 5911</code>

Step	Command
Set the remote connection display access as vnc.	<pre>tmpuser@node# set virtualization guest a-guest devices display connection-type vnc</pre>
Commit the changes.	<pre>tmpuser@node# commit</pre>
Display the result.	<pre>tmpuser@node# show virtualization virtualization { guest a-guest { cpus 2 devices { disk 1 { boot-order 1 device-type disk source { file /var/lib/libvirt/images/image1.qcow2 } } disk 2 { boot-order 3 device-type disk source { file /var/lib/libvirt/images/image2.qcow2 } } display { connection-type vnc password "*****" port 5911 } } memory 2048 uuid b7bbf124-7314-21e2-819f-fef8119cd991 } }</pre>

Chapter 6. Provisioning Guests Automatically

Prerequisites

Provisioning a guest automatically on the VNF platform hypervisor requires using a libvirt domain XML file, a disk image, and an optional ISO file.

Before provisioning a guest automatically, ensure that the required files are either on the system or in a remotely accessible location. A mixture of local and remote sources is supported within a single command. The supported protocols for retrieving remote files are HTTP, HTTPS, and FTP. The following list presents the required files.

- The libvirt domain XML file defines the guest parameters, such as the number of CPUs, amount of RAM, and details about the bridge and vhost interfaces.

For the example in this chapter, the `another-guest.xml` file is used.

- The disk image must be in the QCOW2 format and specifies the guest operating system.

For the example in this chapter, the `disk.qcow2` file is used.

- The ISO configuration file specifies the guest configuration.

For the example in this chapter, the `config.iso` file is used.

Provisioning a guest automatically

You can provision a guest automatically by using the `add virtualization xml` command. For automatic deployment of a guest, a maximum of one hard disk and one cdrom is configured for a guest.

The following example shows how to install and configure a guest with the following parameters:

- Unique name of the guest: `another-guest`.
- Number of CPUs: 4.

The maximum number of CPUs that can be configured is 8.

- Amount of RAM: 4096 MB.

The maximum memory that can be configured is 6144 MB.

To install the guest automatically, perform the following steps in operational mode.

Table 3. Installing a guest automatically

Step	Command
Specify the libvirt domain XML file, disk image, and optional ISO file for the guest	<pre>tmpuser@node:~\$ add virtualization xml /home/vyatta/another-guest.xml image /home/vyatta/disk.qcow2 iso /home/vyatta/config.iso [INFO] - Configured bridge interface "br20" [INFO] - Configured vhost interface "dp0vhost0"</pre>

Step	Command
installation. Ensure that you use fully qualified file names for all three files.	<pre>[INFO] - Configured guest "another-guest" [INFO] - Deployment of /home/vyatta/another-guest.xml complete</pre>
Display the result in configuration mode.	<pre>tmpuser@node:~\$ config tmpuser@node# show virtualization virtualization { guest another-guest { cpus 4 devices { disk 0 { boot-order 1 bus-type virtio device-type disk source { file /var/lib/libvirt/images/another-guest/0-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-config.iso } } network dp0vhost0 { mac-address 52:54:00:16:56:fd } } memory 4096 uuid f7333ab9-8e3d-4a87-a27a-6d5d7d6e99ba } } tmpuser@node# show interfaces vhost vhost dp0vhost0 { bridge-group { bridge br20 } } tmpuser@node# show interfaces bridge bridge br20</pre>

Using the force and save options

The `add virtualization xml` operational mode command has two options: `force` and `save`.

force

The `force` option overwrites any of the following existing items:

- Guest configuration with the same name.
- Disk images at the specified paths that conflict with the new deployment.
- vhost interfaces configuration.

The interface assignments to other guests are not affected.

save

The `save` option enables the following conditions:

- The running configuration is saved as a boot-up configuration if the guest deployment is successful.

The `save` option causes the entire running configuration to be saved as a boot-up configuration and not just the changes made during the deployment of the guest. If there was a configuration that was previously committed and not saved, the configuration is saved as the boot-up configuration.

- If the deployment makes no changes to the configuration, then the running configuration is not saved, even if the `save` option is specified.
- Guests provisioned automatically by using the `save` option are not removed if you restart the system.

When provisioning a guest automatically, you can use the `force` and `save` options, either singly or together.

The following configuration example shows how to change the configuration of a guest by using the `force` and `save` options. In this example, the command specifies another set of the QCOW2 file and the ISO file. When you run the command, the guest details are overwritten and saved.

Table 4. Changing the guest configuration, QCOW2 file, and ISO file

Step	Command
<p>Specify the libvirt XML file, disk image, and the optional ISO file for the guest installation. Ensure that you use fully qualified file names for all the three files.</p>	<pre>tmpuser@node:~\$ add virtualization xml /home/vyatta/another-guest.xml image /home/vyatta/disk.qcow2 iso /home/vyatta/config.iso force save [INFO] - Overwriting existing "another-guest" guest config [INFO] - Using existing bridge interface "br20" [INFO] - Overwriting "dp0vhost0" interface configuration [INFO] - Configured vhost interface "dp0vhost0" [INFO] - Configured guest "another-guest" [INFO] - Powering off existing guest "another-guest" [INFO] - Overwriting "/var/lib/libvirt/images/another-guest/0-disk.qcow2" [INFO] - Overwriting "/var/lib/libvirt/images/another-guest/1-config.iso" [INFO] - Starting guest "another-guest" [INFO] - Configuration saved [INFO] - Deployment of /home/vyatta/another-guest.xml complete</pre>
<p>Display the result in configuration mode.</p>	<pre>tmpuser@node:~\$ config tmpuser@node# show virtualization virtualization { guest another-guest { cpus 2 devices { disk 0 { boot-order 1 bus-type virtio device-type disk source { file /var/lib/libvirt/images/another-guest/0-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-config.iso } } } network dp0vhost0 { mac-address 52:54:00:16:56:fd } } memory 2048 uuid f7333ab9-8e3d-4a87-a27a-6d5d7d6e99ba } tmpuser@node# show interfaces vhost vhost dp0vhost0 { bridge-group { bridge br20 } }</pre>

Step	Command
	<pre> } tmpuser@node# show interfaces bridge bridge br20 </pre>

Supported libvirt domain XML elements and attributes

You can generate examples of libvirt domain XML tags by running the `virsh dumpxml <domain>` command. Only specific libvirt domain XML elements and attributes are supported.

The format for libvirt domain XML is described by <https://libvirt.org/formatdomain.html>. The XML file specifies the guest and guest hardware details. The XML file is parsed, processed, and translated into a number of VNF platform virtualization commands.

The following table lists the supported libvirt domain XML elements and attributes. The element and attribute values used in the following tables are examples only. For XML code snippets that include the supported elements and attributes, refer to [Sample XML snippet for the libvirt domain XML file](#).

Legend for the table:

- The XML element names are shown with leading and trailing angle brackets, such as `<name>`.
- The XML attribute values are enclosed within single quotation marks, such as 'device'.
- Variables are denoted by the *italicized* font.
- The commands and XML snippets are in `monospace` font.

Table 5. Supported XML elements and attributes

XML element or attribute	VNF platform command	Description
<code><domain type='kvm':></domain></code>		The rest of the guest XML specifications must be enclosed by the <code><domain></code> element with a 'type' attribute of 'kvm'.
<code><name>aguest</name></code>	<code>set virtualization guest <guest-name></code>	The XML must contain a nonblank <code><name></code> element. The value of the <code><name></code> element becomes the guest name in the subsequent VNF platform commands.
<code><vcpu>2</vcpu></code>	<code>set virtualization guest <guest-name></code> <code>cpus 2</code>	The <code><vcpu></code> element defines the number of virtual CPUs assigned to the guest.
<code><vcpu cpuset='2-3'>2</vcpu></code>	<code>set virtualization guest <guest-name></code> <code>cpuset 2-3</code>	The <code><cpuset></code> element defines the virtual CPU affinity.
<code><memory>1048576</memory></code>	<code>set virtualization guest <guest-name></code> <code>memory 1024</code>	The <code><memory></code> element defines the amount of memory allocated to the guest. The default memory unit is kilobytes (blocks of 1024 bytes), but you can change it by using the <code><memory unit='.'></code> attribute. For more information, refer to xx .
<code><uuid>d4d15d82-f21e-4b0a-bb91-3224cfbc8514</uuid></code>	<code>set virtualization guest <guest-name></code> <code>uuid d4d15d82-f21e-4b0a-bb91-3224cfbc8514</code>	Defines the UUID assigned to the guest. The UUID is generated automatically if the element is not present.
<code><devices></devices></code>		All hardware devices must be contained within the <code><devices></code> element. The <code><devices></code> subelements that

XML element or attribute	VNF platform command	Description
		are supported are: <disk>, <interface>, <graphics>, <video>, and <watchdog>.
<pre><disk device='cdrom'>...</disk> <disk device='disk'>...</disk></pre>	<pre>set virtualization guest <guest-name> devices disk <disk-id> device-type {cdrom disk}</pre>	<p>The <disk> element must always have a 'device' attribute with a valid value. Two types of disk devices are supported: 'cdrom' and 'disk'.</p> <p>The <disk-id> is automatically allocated on a per-guest basis, starting with the lowest unused disk-id (from 0 to 7).</p> <p>The <disk> element may contain a <target> or a <boot> subelement.</p>
<pre><target bus='ide' /></pre>	<pre>set virtualization guest <guest-name> devices disk <disk-id> bus-type ide</pre>	The attribute 'ide' is normally used for cdrom drives.
<pre><target bus='sata' /></pre>	<pre>set virtualization guest <guest-name> devices disk <disk-id> bus-type sata</pre>	
<pre><target bus='virtio' /></pre>	<pre>set virtualization guest <guest-name> devices disk <disk-id> bus-type virtio</pre>	The attribute 'virtio' is normally used for disk drives.
<pre><boot order='3' /></pre>	<pre>set virtualization guest <guest-name> devices disk <disk-id> boot-order 3</pre>	The <boot> element may contain an 'order' attribute whose value determines the boot order of the disk devices. If the <boot> element is missing, or if it does not have an 'order' attribute, the disk is given a default boot-order number equal to its <disk-id> plus 1. So disk 0 has a default boot-order of 1, disk 1 has a default boot-order of 2, and so on.
<pre><interface type='...'> </ interface></pre>		The <interface> element must always have a 'type' attribute with a valid value. Two types of interface devices are supported: 'bridge' and 'ethernet'. Both interface types can contain a <target> element which can have a 'dev' attribute.
<pre><target dev='...' /></pre>		
<pre><mac-address='52:54:00:4b:d6:fd' /></pre>	<pre>set interfaces vhost dp0vhost<n> mac- address 52:54:00:4b:d6:fd</pre>	The <mac-address> element sets the MAC address of the interface. The value is generated automatically if the element is not present.
<pre><target dev='dp0vhost8' /></pre>	<pre>set interfaces vhost dp0vhost8</pre>	When the dev attribute is of the form dp0vhost<n> the deployment attempts to use or create the vhost interface with that particular name.
<pre><target dev='network-23' /></pre>	<pre>set interfaces vhost dp0vhost<n> description 'network-23'</pre>	If the 'dev' attribute value does not start with the 'dp0vhost' prefix, the next free vhost interface is automatically created with the set interfaces vhost command. The 'dev' value is inserted into the description field.
<pre><interface type='bridge'> <source bridge='br4' /> </interface></pre>	<pre>set interfaces vhost dp0vhost<n> bridge-group bridge br4</pre>	<p>The 'bridge' type interface can contain a <source> element with a 'bridge' attribute whose value defines the name of the required bridge. The following three outcomes are possible on running the XML snippet.</p> <ul style="list-style-type: none"> • If the specified bridge interface does not already exist in the running configuration of the host, it is automatically configured with the set interfaces bridge command. • If the bridge interface already exists, the existing configuration is used. • If there is a <target> element that specifies a vhost interface that already exists, the deployment fails if force is not specified. If force is specified, the existing vhost interface configuration is overwritten.

XML element or attribute	VNF platform command	Description
<pre><interface type='ethernet'> <ip address='192.168.1.1' prefix='24' /> </interface></pre>	<pre>set interfaces vhost dp0vhost<n> address 192.168.1.1/24</pre>	The 'ethernet' type interface can contain an <ip> subelement that has 'address' and 'prefix' attributes whose values define the IP address of the interface. If there is a <target> element that specifies a vhost interface that already exists, the deployment fails if force is not specified. If force is specified, the existing vhost interface configuration is overwritten.
<pre><graphics type='...' /></pre>		Only the first <graphics> element in the XML file is processed. The <graphics> element must have a 'type' attribute with one of the following two values: 'spice' or 'vnc'. The <graphics> element must also have either a 'port' or an 'autoport' attribute.
<pre><graphics type='spice'port='6543' /></pre>	<pre>set virtualization guest <guest-name> devices display connection-type spice set virtualization guest <guest-name> devices display port 6543</pre>	If the 'port' attribute value is anything other than '-1', its value is reflected in the devices display port command. A 'port' attribute value of '-1' is treated as the equivalent of an 'autoport' attribute value of 'yes'.
<pre><graphics type='vnc'autoport='yes' /></pre>	<pre>set virtualization guest <guest-name> devices display connection-type vnc set virtualization guest <guest-name> devices display port 5900</pre>	When 'autoport' is 'yes', the first unused port number greater than 5899 is used.
<pre><graphics passwd='top-secret' /></pre>	<pre>set virtualization guest <guest-name> devices display password top-secret</pre>	The password passed and stored in the configuration is in clear text, and, therefore, is not secure.
<pre><graphics> <listen type='address' address='127.0.0.1' /> </graphics></pre>	<pre>set virtualization guest <guest-name> devices display address 127.0.0.1</pre>	The listen element (or attribute) causes the display to only listen for connections on the configured address. The configured address must be assigned to an interface of the VNF platform host.
<pre><graphics tlsPort='1234' /></pre>		Currently, the 'tlsPort' attribute is ignored.
<pre><video>:</video></pre>	<pre>set virtualization guest <guest-name> devices video ...</pre>	Only the first <video> element in the XML file is processed. The <video> element must have a <model> subelement with a 'type' attribute with one of the following three values: 'cirrus', 'qxl' or 'vga'.
<pre><model type='cirrus' /></pre>	<pre>set virtualization guest <guest-name> devices video model cirrus</pre>	
<pre><model type='qxl' /></pre>	<pre>set virtualization guest <guest-name> devices video model qxl</pre>	
<pre><model type='vga' /></pre>	<pre>set virtualization guest <guest-name> devices video model vga</pre>	
<pre><model type='qxl' vram='8196' /></pre>	<pre>set virtualization guest <guest-name> devices video memory 8196</pre>	The <model> element may also have the optional 'vram' attribute.
<pre><watchdog model='i6300esb' /></pre>	<pre>set virtualization guest <guest-name> devices watchdog ...</pre>	Only the first <watchdog> element in the XML file is processed. The <watchdog> element must have the 'model' attribute, and its value must be 'i6300esb'. The <watchdog> element may also contain an optional 'action' attribute that can have one of the following values: 'none', 'poweroff' or 'reset'.
<pre><watchdog model='i6300esb'action='none' /></pre>	<pre>set virtualization guest <guest-name> devices watchdog action none</pre>	
<pre><watchdog model='i6300esb'action='poweroff' /></pre>	<pre>set virtualization guest <guest-name> devices watchdog action poweroff</pre>	
<pre><watchdog model='i6300esb'action='reset' /></pre>	<pre>set virtualization guest <guest-name> devices watchdog action reset</pre>	

Valid memory element identifier and multiplier values

The 'unit' attribute for the <memory> element in the libvirt domain XML file for provisioning guests automatically accepts specific identifiers and multiplier values.

If you do not allocate accepted values, such as those shown in the following table, the system displays an error message about an unrecognized allocation unit.

For example, the <memory unit='G'>4</memory> XML snippet attempts to allocate 4,294,967,296 bytes of memory ($4,294,967,296 = 4 * 1,024 * 1,024 * 1,024$) for the guest. The default memory allocation unit is 1,024 bytes.

Table 6. Valid memory element identifier and multiplier values

Identifier	Value
b	1 byte
bytes	1 byte
KB	1000 bytes
k	1024 bytes
KiB	1024 bytes
MB	1000 * 1000 bytes
M	1024 * 1024 bytes
MiB	1024 * 1024 bytes
GB	1000 * 1000 * 1000 bytes
G	1024 * 1024 * 1024 bytes
GiB	1024 * 1024 * 1024 bytes
TB	1000 * 1000 * 1000 * 1000 bytes
T	1024 * 1024 * 1024 * 1024 bytes
TiB	1024 * 1024 * 1024 * 1024 bytes

Sample XML snippet for the libvirt domain XML file

The libvirt domain XML file that is used in provisioning a guest automatically defines the guest parameters, such as the number of CPUs, amount of RAM, and details about the bridge and vhost interfaces.

When you are building your XML file, you can refer to the following sample XML snippet and the details of the elements in [Supported libvirt domain XML elements and attributes](#). The following XML file configures a guest with the following details:

- Name of the guest: `aguest`
- Number of vCPUs: 4
- Amount of RAM: `512 MB`
- IP address of the vhost interface: `192.168.1.1/24`


```
<domain type='kvm'>
  <name>aguest</name>
  <vcpu>4</vcpu>
  <memory unit='M'>512</memory>
  <devices>
    <interface type='ethernet'>
      <ip address='192.168.1.1' prefix='24' />
    </interface>
  </devices>
</domain>
```

VRF overview

A variety of technologies exist to allow multiple scopes, or routing instances, within a single router. Virtual Routing and Forwarding (VRF) is a technology that controls information flow within a network by partitioning the network and separating Layer 3 traffic into different logical VRF domains. For each VRF domain, the router maintains a separate routing table and Layer 3 forwarding tables and can run separate instances of routing protocols.

VRF allows you to logically split the router into multiple L3 routers that allows for overlapping address spaces without contention. The vhost interfaces are supported as part of VRF. For information on VRF, refer to *Basic Routing Configuration Guide*.

Chapter 7. Provisioning Guests Commands

add virtualization xml image

Provisions a guest automatically by using a libvirt domain XML file to specify the guest parameters and an image file in the QCOW2 format to specify the guest operating system.

```
add virtualization xml xml image disk-image [ iso iso ] [ force ] [ save ] [ routing-instance name ]
```

xml

A libvirt domain XML file that defines the guest parameters, such as the number of CPUs, amount of RAM, and details about the bridge and vhost interfaces.

disk-image

An image for the guest operating system in the QCOW2 format.

iso

An ISO configuration file that defines the guest configuration.

force

Enables the overwriting of the existing guest configuration, disk image, and vhost interfaces while installing a guest.

save

Enables the running configuration to be saved as a boot-up configuration if the guest installation is successful.

name

Specifies the name of the routing instance to reach the remote host or hosts serving the XML, disk image or ISO files. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Operational mode.

Use this command to provision a guest automatically by using a libvirt domain XML file to specify the guest parameters and an image file in the QCOW2 format to specify the guest operating system.

When provisioning a guest automatically, you can use the `force` and `save` options, either singly or together. For more information on the `force` and `save` options, refer to [Using the force and save options](#).

Before provisioning a guest automatically, ensure that the required files are either on the system or in a remotely accessible location. A mixture of local and remote sources is supported within a single command. When a routing instance is specified, it is used to reach all remote hosts that are specified. For more information about VRF configuration, refer to the VRF Support chapter in the *Basic Routing Configuration Guide*.

During a deployment or upgrade, the disk images are moved from their current, specified location, and not copied, if they are local.

Some examples of locally hosted files are:

- /home/anotherguest.xml
- /home/disk.qcow2
- /home/config.iso

Some examples of remotely hosted files are:

- http://10.10.1.2/guest_app.xml
- http://10.10.1.2/guest_app.qcow2
- http://10.10.1.2/guest_app.iso

The following example shows how to provision a guest automatically.

```
tmpuser@node:~$ add virtualization xml /home/vyatta/anotherguest.xml
image /home/vyatta/disk.qcow2 iso /home/vyatta/config.iso
```

generate virtualization { random-mac | uuid }

Generates a random UUID or MAC address for assignment to a guest.

```
generate virtualization { random-mac | uuid }
```

random-mac

Generates a MAC address.

uuid

Generates a UUID.

Operational mode.

Use this command to generate a random UUID or MAC address for assignment to a guest.

The following example shows how to generate a MAC address.

```
tmpuser@node:~$ generate virtualization random-mac
52:54:00:7a:ed:43
tmpuser@node:~$
```

virtualization guest

Creates a guest with a specific guest name.

```
set virtualization guest guest-name
delete virtualization guest guest-name
show virtualization
```

None.

guest-name

A unique guest name.

Configuration mode.

```
virtualization {
    guest guest-name
}
```

Use this command to create a guest with a specific name.

The name for a guest can be alphanumeric, consisting of uppercase letters, lowercase letters, numbers, the dash, and underscore . However, you cannot start a guest name with either the dash or underscore.

Use the `set` form of the command to create a guest with a specific name. Use a unique guest name.

Use the `delete` form of the command to delete a guest with a specific name. The `delete virtualization guest` command deletes the guest configuration only. The command does not delete any disk images that are used by the hypervisor in the `/var/lib/libvirt/images` folder.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest cpus

Assigns a number of virtual CPUs to a guest.

```
set virtualization guest guest-name cpus cpu-number
delete virtualization guest guest-name cpus cpu-number
show virtualization
```

If a number is not specified, the guest is assigned one virtual CPU.

guest-name

A guest name.

cpu-number

A number of virtual CPUs. The number ranges from 1 through 8.

Configuration mode.

```
virtualization {
  guest guest-name {
    cpus cpu-number
  }
}
```

Use this command to assign a number of virtual CPUs to a guest.

Use the `set` form of the command to assign a number of virtual CPUs to a guest.

Use the `delete` form of the command to delete the number of virtual CPUs from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest cpuset

Assigns a number of CPUs to a cpuset for a guest.

```
set virtualization guest guest-name cpuset cpu-number
```

```
delete virtualization guest guest-name cpuset cpu-number
```

```
show virtualization
```

None.

guest-name

A guest name.

cpu-number

A number of CPUs assigned to a cpuset.

Configuration mode.

```
virtualization {
  guest guest-name {
    cpuset cpu-number
  }
}
```

Use this command to assign a cpuset to a guest. When a cpuset is established to a guest, the guest runs only on the CPUs that are attached to the cpuset.

Use the `set` form of the command to assign a cpuset to a guest.

Use the `delete` form of the command to delete the cpuset from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices disk

Specifies a virtual disk for a guest.

```
set virtualization guest guest-name devices disk disk-id
```

```
delete virtualization guest guest-name devices disk disk-id
```

```
show virtualization
```

None.

guest-name

A unique guest name.

disk-id

A number for a virtual disk. The number ranges from 0 through 7.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      disk disk-id
    }
  }
}
```

Use this command to specify a virtual disk for a guest.

Use the `set` form of the command to specify a virtual disk for a guest.

Use the `delete` form of the command to delete the virtual disk from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices disk boot-order

Specifies a boot order for a virtual disk of a guest.

```
set virtualization guest guest-name devices disk disk-id boot-order boot-order-number
```

```
delete virtualization guest guest-name devices disk disk-id boot-order boot-order-number
```

```
show virtualization
```

None.

guest-name

A guest name.

disk-id

The number of a virtual disk. The number ranges from 0 through 7.

boot-order-number

A boot order number for the virtual disk. The number ranges from 1 through 8.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      disk disk-id {
        boot-order boot-order-number
      }
    }
  }
}
```

Use this command to specify a boot order number for a virtual disk of a guest. The disk with the lowest boot order number boots first.

Use the `set` form of the command to specify a boot order for a virtual disk of a guest.

Use the `delete` form of the command to delete the boot order from a virtual disk of a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices disk bus-type

Specifies the bus type for a virtual disk of a guest.

```
set virtualization guest guest-name devices disk disk-id bus-type { ide | sata | virtio }
```

```
delete virtualization guest guest-name devices disk disk-id bus-type { ide | sata | virtio }
```

```
show virtualization
```

If a bus type is not specified, the guest is assigned a bus type of virtio.

guest-name

A guest name.

disk-id

The number of a virtual disk. The number ranges from 0 through 7.

ide

Specifies the bus type as IDE.

sata

Specifies the bus type as SATA.

virtio

Specifies the bus type as virtio.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      disk disk-id {
        bus-type { ide | sata | virtio }
      }
    }
  }
}
```

Use this command to specify the bus type for a virtual disk of a guest.

If you select `device-type` as `cdrom`, you cannot select `virtio` as `bus-type`. Virtio is the fastest of the available bus types. Most Linux and BSD-based images support Virtio. If the guest cannot find the virtual disk, try changing `bus-type` to `ide` or `sata`.

Use the `set` form of the command to specify a bus type for a virtual disk of a guest.

Use the `delete` form of the command to delete the bus type from a virtual disk of a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices disk device-type

Specifies the disk type for a virtual disk of a guest.

```
set virtualization guest guest-name devices disk disk-id device-type { cdrom | disk }
```

```
delete virtualization guest guest-name devices disk disk-id device-type { cdrom | disk }
```

```
show virtualization
```

None.

guest-name

A guest name.

disk-id

The number of a virtual disk. The number ranges from 0 through 7.

Configuration command.


```

virtualization {
  guest guest-name {
    devices {
      disk disk-id {
        device-type { cdrom | disk }
      }
    }
  }
}

```

Use this command to specify the disk type for a virtual disk of a guest.

If `device-type` is `disk` then the guest image must be in the QCOW2 format. If `device-type` is `cdrom`, then the guest image must be in the ISO-9660 format. If you select `cdrom` as `device-type`, you cannot select `virtio` as `bus-type`.

Use the `set` form of the command to specify the disk type for a virtual disk of a guest.

Use the `delete` form of the command to delete the disk type from a virtual disk of a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices disk source file

Specifies an image in QCOW2 or ISO-9660 format for a virtual disk.

```
set virtualization guest guest-name devices disk disk-id source file file-name
```

```
delete virtualization guest guest-name devices disk disk-id source file file-name
```

```
show virtualization guest
```

None.

guest-name

A guest name.

disk-id

The number of a virtual disk. The number ranges from 0 through 7.

file-name

An image in QCOW2 or ISO-9660 format.

Configuration mode.

```

virtualization {
  guest guest-name {
    devices {
      disk disk-id {

```

```

        source {
            file file-name
        }
    }
}

```

Use this command to specify an image in QCOW2 or ISO-9660 format for a virtual disk. Use fully qualified file names. The image must be located at `/var/lib/libvirt/images/`.

If `device-type` is `disk`, the QCOW2 format is supported. If `device-type` is `cdrom`, the ISO-9660 format is supported.

Use the `set` form of the command to specify an image for a virtual disk.

Use the `delete` form of the command to delete an image from a virtual disk.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices display address

Enables a remote display to only listen for connections on the specified IPv4 or IPv6 address.

```
set virtualization guest guest-name devices display address ip-address
```

```
delete virtualization guest guest-name devices display address ip-address
```

```
show virtualization
```

None.

guest-name

A guest name.

ip-address

An IPv4 or IPv6 address.

Configuration mode.

```

virtualization {
  guest guest-name {
    devices {
      display {
        address ip-address
      }
    }
  }
}

```

Use this command to enable a remote display to only listen for connections on the specified IPv4 or IPv6 address. The address must be present on an interface of the host.

By configuring a loopback address for the display and the SSH service, you can ensure that the display connections are tunneled by using SSH.

Use the `set` form of the command to enable a remote display to only listen for connections on the specified IPv4 or IPv6 address.

Use the `delete` form of the command to remove a remote display from only listening to connections on the specified IPv4 or IPv6 address.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices display connection-type

Sets the type of remote display for a guest.

```
set virtualization guest guest-name devices display connection-type { vnc |
spice }
```

```
delete virtualization guest guest-name devices display connection-type { vnc |
spice }
```

```
show virtualization
```

If the connection type for a remote display is not specified, the remote display is assigned VNC as the connection type.

guest-name

A guest name.

vnc

Specifies a VNC-based remote display.

spice

Specifies a SPICE-based remote display.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      display {
        connection-type { vnc | spice }
      }
    }
  }
}
```

Use this command to set the type of remote display for a guest. Of the two remote display types, SPICE is the faster and more modern protocol, but VNC supports guests on more platforms.

Use the `set` form of the command to set the type of remote display for a guest.

Use the `delete` form of the command to delete the remote display from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices display password

Specifies a text-based password for the connection to the remote display of a guest.

```
set virtualization guest guest-name devices display password password-text
```

```
delete virtualization guest guest-name devices display password password-text
```

```
show virtualization
```

None.

guest-name

A guest name.

password-text

A password.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      display {
        password password-text
      }
    }
  }
}
```

Use this command to specify a text-based password for the connection to the remote display of a guest.

Use the `set` form of the command to set a text-based password for the connection to the remote display of a guest.

Use the `delete` form of the command to delete a text-based password from the connection to the remote display of a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices display port

Specifies a port for a connection to the remote display of a guest.

```
set virtualization guest guest-name devices display port port-number
```

```
delete virtualization guest guest-name devices display port port-number
```

```
show virtualization
```

None.

guest-name

A guest name.

port-number

The port number for a connection to the remote display of a guest. The number ranges from 5,900 to 65,535.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      display {
        port port-number
      }
    }
  }
}
```

Use this command to specify a port for a connection to the remote display of a guest

Use the `set` form of the command to specify a port for a connection to the remote display of a guest

Use the `delete` form of the command to delete a port from a connection to the remote display of a guest

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest memory

Assigns an amount of memory to a guest.

```
set virtualization guest guest-name memory memory-value
```

```
delete virtualization guest guest-name memory memory-value
```

```
show virtualization
```

If an amount of memory is not specified, the guest is assigned 1,024 MB of memory.

guest-name

A guest name.

memory-value

An amount of memory in megabytes. The amount ranges from 1 through 6,144 MB.

Configuration mode.

```
virtualization {
  guest guest-name {
    memory memory-value
  }
}
```

Use this command to assign an amount of memory to a guest.

Use the `set` form of the command to assign an amount of memory to a guest. The maximum memory that can be assigned is 6,144 MB.

Use the `delete` form of the command to delete an amount of memory from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices network

Specifies a vhost interface for a guest.

```
set virtualization guest guest-name devices network dp0vhostN
```

```
delete virtualization guest guest-name devices network dp0vhostN
```

```
show virtualization
```

None.

guest-name

A guest name.

dp0vhostN

A vhost interface.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      network dp0vhostN
    }
  }
}
```

```

    }
  }
}

```

Use this command to specify a vhost interface for a guest.

Use the `set` form of the command to specify a vhost interface for a guest.

Use the `delete` form of the command to delete a vhost interface from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices network mac-address

Specifies a unique MAC address for a vhost interface.

```
set virtualization guest guest-name devices network dp0vhostN mac-address mac-address-value
```

```
delete virtualization guest guest-name devices network dp0vhostN mac-address mac-address-value
```

```
show virtualization
```

None.

guest-name

A guest name.

dp0vhostN

A vhost interface.

mac-address-value

A unique MAC address.

Configuration mode.

```

virtualization {
  guest guest-name {
    devices {
  network dp0vhostN {
    mac-address mac-address-value
  }
}
}
}

```

Use this command to specify a unique MAC address for a vhost interface.

Use the `set` form of the command to specify a unique MAC address for a vhost interface.

Use the `delete` form of the command to delete the MAC address from a vhost interface.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices video memory

Specifies video memory for a guest.

```
set virtualization guest guest-name devices video memory memory-value
```

```
set virtualization guest guest-name devices video memory memory-value
```

```
show virtualization
```

If video memory is not specified, the guest is assigned 64 MB of video memory.

guest-name

A guest name.

memory-value

Video memory for a guest in megabytes. The amount ranges from 1 through 65,536 in MB.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      video {
        memory memory-value
      }
    }
  }
}
```

Use this command to specify video memory for a guest.

Use the `set` form of the command to specify video memory for a guest.

Use the `delete` form of the command to delete the video memory from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest devices video model

Specifies the type of video card for a guest.

```
set virtualization guest guest-name devices video model { cirrus | qxl | vga }
```

```
delete virtualization guest guest-name devices video model { cirrus | qxl | vga }
}
```



```
show virtualization
```

If the type of video card is not specified, the guest is assigned a QXL video card.

guest-name

A guest name.

cirrus

Specifies an unaccelerated Cirrus compatible video card.

qxl

Specifies a QXL accelerated video card.

vga

Specifies an unaccelerated VGA compatible video card.

Configuration mode.

```
virtualization {
  guest guest-name {
    devices {
      video {
        model { cirrus | qxl | vga }
      }
    }
  }
}
```

Use this command to specify the type of video card for a guest.

Use the `set` form of the command to specify the type of video card for a guest.

Use the `delete` form of the command to delete the type of video card from a guest.

Use the `show` form of the command to display the configuration details for a guest.

virtualization guest uuid

Assigns a universally unique identifier (UUID) to a guest.

```
set virtualization guest guest-name uuid uuid-number
```

```
delete virtualization guest guest-name uuid uuid-number
```

```
show virtualization
```

None.

guest-name

A guest name.

uuid *uuid-number*

Assigns a UUID to a guest.

Configuration mode.

```
virtualization {
    guest guest-name {
        uuid uuid-number
    }
}
```

Use this command to assign a UUID to a guest. Each guest uses the UUID, in addition to a guest name, to uniquely identify itself to the hypervisor. The UUID is mandatory for configuring a guest.

Use the `set` form of the command to assign a UUID to a guest.

Use the `delete` form of the command to delete the UUID from a guest.

Use the `show` form of the command to display the configuration details for a guest.

Chapter 8. Connecting Guests to a VNF Platform Data Plane

Overview

The vhost interface handles communication between the hypervisor and a guest running on top of the hypervisor.

Each guest requires a unique vhost interface. If you want the guests to communicate with each other, you can add all the vhost interfaces of the different guests to the same bridge group. Alternatively, to keep each guest isolated, add the vhost interface of each guest to a unique bridge group. You can also assign the IP address of the vhost interface to the same LAN as the guest to enable the guest to communicate with the hypervisor. vhost interfaces can be assigned to a routing-instance.

For more information about routing instance, refer to the *Basic Routing Configuration Guide*.

Configuring a vhost interface

The following example shows how to create and configure a vhost interface for a guest with the following parameters:

- Name of the guest: `a-guest`
- Name of the bridge: `br0`
- Name of the vhost interface: `dp0vhost0`

To create and configure a vhost interface, perform the following steps in configuration mode.

Table 7. Configuring a vhost interface


Step	Command
Create a bridge called <code>br0</code> .	<pre>tmpuser@node# set interfaces bridge br0</pre>
Create a vhost interface called <code>dp0vhost0</code> .	<pre>tmpuser@node# set interfaces vhost dp0vhost0</pre>
Add the vhost interface to the bridge group.	<pre>tmpuser@node# set interfaces vhost dp0vhost0 bridge-group bridge br0</pre>
Set the <code>a-guest</code> guest to use the vhost interface you just created.	<pre>tmpuser@node# set virtualization guest a-guest devices network dp0vhost0</pre>
Assign a unique MAC address for the guest.	<pre>tmpuser@node# set virtualization guest a-guest devices network dp0vhost0 mac-address 52:54:00:16:56:fd</pre>
Commit the changes.	<pre>tmpuser@node# commit</pre>
Display the result.	<pre>tmpuser@node# show interfaces bridge bridge br0 tmpuser@node# show interfaces vhost</pre>

Step	Command
	<pre>vhost dp0vhost0 { bridge-group { bridge br0 } }</pre>
	<pre>tmpuser@node# show virtualization guest a-guest devices network dp0vhost0 network dp0vhost0 { mac-address 52:54:00:16:56:fd }</pre>

Configuring sFlow for vhost interface

[Table 8: Configuring sFlow](#) shows how to configure sFlow for the sample configuration. For the configuration topology, refer to **sFlow Overview** topic in *Services Configuration Guide*.

Table 8. Configuring sFlow

Step	Command
Specify the address of the agent. By default, this collector listens for sFlow data on the 6343 UDP port.  Note: A router supports up to four collectors.	<pre>tmpuser@node# set service sflow agent-address 1.1.1.1</pre>
Set the polling interval to three seconds. Every three seconds, the sFlow agent collects port counter data.	<pre>tmpuser@node# set service sflow polling-interval 3</pre>
Configure the sFlow agent to send the UDP datagrams that contain the collected sFlow information to the default port (6343) of the sFlow collector server at 198.51.100.2.	<pre>tmpuser@node# set service sflow server-address 198.51.100.2 server-port 6343</pre>
Specify 512 as the number of packets from which a sample is taken by the sFlow agent. In other words, for every 512 packets that flow through the interface, the sFlow agent selects one packet for analysis.	<pre>tmpuser@node# set service sflow sampling-rate 512</pre>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>
Display the sFlow configuration. The output shows that sFlow was configured, but no statistics were collected. This is because sFlow has not yet been enabled on an interface.	<pre>tmpuser@node# run show sflow { "sFlow information": { "sFlow version": 5, "sFlow services": "enabled", "sFlow agent IP address": "1.1.1.1", "Collector destinations configured": 1, "Collectors": [{ "IP address": "198.51.100.2", "UDP port number": 6343 }], "Polling interval": 3, "Configured default sampling rate": 512, "Actual default sampling rate": 512, "sFlow max-header size": 128, "UDP packets sent": 0, "Flow samples collected": 0, "sFlow interfaces": [], "Total sFlow interfaces": 0 }</pre>
Enable sFlow on the dp0vhost0 interface (The topology diagram indicates it as DP1).	<pre>tmpuser@node# set interfaces vhost dp0vhost0 sflow</pre>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>

Step	Command
The sFlow agent can now start collecting packet samples and port-counter statistics.	
<p>Display the sFlow configuration.</p> <p>The output shows that sFlow is enabled for the dp0p192p1 interface and shows that one sample packet was collected.</p>	<pre>tmpuser@node# run show sflow { "sFlow information": { "sFlow version": 5, "sFlow services": "enabled", "sFlow agent IP address": "1.1.1.1", "Collector destinations configured": 1, "Collectors": [{ "IP address": "198.51.100.2", "UDP port number": 6343 }], "Polling interval": 3, "Configured default sampling rate": 512, "Actual default sampling rate": 512, "sFlow max-header size": 128, "UDP packets sent": 0, "Flow samples collected": 0, "sFlow interfaces": [{ "name": "dp0p192p1" }], "Total sFlow interfaces": 1 }</pre>

Configuring QoS on a vhost interface

Quality of Service (QoS) is supported on the vhost devices that provide communication between the guests running within a VNF platform hypervisor. All existing QoS commands are supported in the direction from the vhost to the guest. The commands can be used to classify, limit, remark, and prioritize traffic entering the guest. QoS is not supported for control of traffic from the guest to the vhost.

For more information on QoS commands, refer to *QoS Configuration Guide*.

The following example shows how to create a vhost interface and assign a QoS policy with the following parameters:

- Name of the vhost interface: dp0vhost0
- Name of the QoS policy: guest1

To assign a QoS policy to a vhost interface, perform the following steps in configuration mode.

Table 9. Configuring QoS on a vhost interface

Step	Command
Create a vhost interface called dp0vhost0.	<pre>tmpuser@node# set interfaces vhost dp0vhost0</pre>
Assign the guest1 QoS policy to the interface.	<pre>tmpuser@node# set interfaces vhost dp0vhost0 policy qos guest1</pre>
Commit the changes.	<pre>tmpuser@node# commit</pre>
Verify the QoS policy.	<pre>tmpuser@node# show interfaces vhost dp0vhost0 policy policy { qos guest1 }</pre>

Step	Command
	}
Show the QoS policy.	<pre>tmpuser@node# show queuing dp0vhost0 class Interface Prio Packets Bytes Match ----- dp0vhost0 1 4 1368 proto 17 to any port 67 tag 1 2 5 570 proto 17 to any port 547 tag 1</pre>

Chapter 9. Connecting Guests Commands

interfaces vhost sflow

Specifies an interface for which to record inbound sFlow packet statistics and port counters.

```
set interfaces vhost dp-port sflow
```

```
delete interfaces vhost dp-port sflow
```

dp-port

The name of a vhost interface.

Configuration mode

```
interfaces {  
    vhost dp-port {  
        sflow  
    }  
}
```

You can enable multiple interfaces by issuing this command multiple times, once for each interface.

Use the `set` form of this command to enable sFlow on an interface.

Use the `delete` form of this command to disable sFlow on an interface.

interfaces vhost flow-monitoring aggregator

Defines flow-monitoring aggregator on a vhost interface.

```
set interfaces vhost interface flow-monitoring aggregator aggregator-name
```

```
delete interfaces vhost interface flow-monitoring aggregator aggregator-name
```

```
show interfaces vhost interface flow-monitoring aggregator aggregator-name
```

None

interface

Vhost interface.

aggregator-name

Name of a flow-aggregator.

Configuration mode

```

interfaces {
  vhost interface {
    flow-monitoring {
      aggregator aggregator-name
    }
  }
}

```

Use the `set` form of this command to define a flow-monitoring aggregator on a vhost interface.

Use the `delete` form of this command to remove a flow-monitoring aggregator from the vhost interface.

Use the `show` form of this command to display the name of flow-monitoring aggregator for a vhost interface.

interfaces vhost flow-monitoring exporter

Defines flow-monitoring exporter on a data plane interface.

```
set interfaces vhost interface flow-monitoring exporter exporter-name
```

```
delete interfaces vhost interface flow-monitoring exporter exporter-name
```

```
show interfaces vhost interface flow-monitoring exporter exporter-name
```

None

interface

A vhost interface.

exporter-name

Name of a flow exporter.

Configuration mode

```

interfaces {
  vhost interface {
    flow-monitoring {
      exporter exporter-name
    }
  }
}

```

Use the `set` form of this command to define a flow-monitoring exporter on a vhost interface.

Use the `delete` form of this command to remove a configured flow-monitoring exporter from the vhost interface.

Use the `show` form of this command to display the name of a configured exporter for a vhost interface.

interfaces vhost flow-monitoring selector

Associates a packet selector with a vhost interface through which the traffic to be monitored flows.

```
set interfaces vhost interface flow-monitoring selector selector-name
```

```
delete interfaces vhost interface flow-monitoring selector selector-name
```

```
show interfaces vhost interface flow-monitoring
```

interface

A vhost interface.

selector-name

The name of a packet selector.

Configuration mode

```
interfaces {
  vhost interface {
    flow-monitoring {
      selector selector-name
    }
  }
}
```

Use the `set` form of this command to associate a packet selector with a vhost interface through which the traffic to be monitored flows.

Use the `delete` form of this command to disassociate a packet selector from a vhost interface.

Use the `show` form of this command to display the name of the configured selector for a vhost interface.

interfaces vhost policy qos

Assigns a QoS policy for traffic from a vhost interface to a guest.

```
set interfaces vhost dp0vhostN policy qos policy-name
```

```
delete interfaces vhost dp0vhostN policy qos policy-name
```

```
show interfaces vhost dp0vhostN
```

None.

dp0vhostN

Specifies a vhost interface.

policy-name

Specifies the name of a QoS policy to apply to a vhost interface.

Configuration mode.

```
interfaces {  
  vhost dp0vhostN {  
    policy {  
      qos policy-name  
    }  
  }  
}
```

Use the `set` form of this command to assign a QoS policy for traffic from a vhost to the guest.

Use the `delete` form of this command to remove a QoS policy for traffic from a vhost to the guest.

Use the `show` form of this command to display the current vhost interface for traffic from a vhost to the guest.

Chapter 10. Upgrading Guests

Overview

You can use the `upgrade virtualization guest` and `upgrade virtualization xml` commands to upgrade a guest.

The guest, which is created automatically, is specified by three files that can be modified by the `upgrade virtualization xml` command:

- The libvirt domain XML file defines the guest parameters, such as the number of CPUs, amount of RAM, and the details for the bridge and vhost interfaces.
- The guest image must be in the QCOW2 format and specifies the guest operating system.
- The ISO configuration file specifies the guest configuration.

The upgrade commands are described in the following paragraphs:

`upgrade virtualization guest`

The `upgrade virtualization guest` command replaces the disk image of a virtual disk for a guest with a different image. The command also powers the guest off and on. The image may be local or remote. You can upgrade multiple disks for a guest in one command. Any existing images that conflict with the new images are overwritten during an upgrade. The configuration is saved after an upgrade. Any images which were previously used by the disks being upgraded are deleted after an upgrade, if they are not overwritten during the upgrade. The `keep-old-images` option prevents the deletion of old images.

`upgrade virtualization xml`

After you provision a guest automatically, you can use the `upgrade virtualization xml` command to upgrade a guest.

The `upgrade virtualization xml` command uses a libvirt domain XML file and, optionally, a new disk image, configuration ISO file, or both.

The command reconfigures an existing guest according to the parameters defined within the specified XML file. The XML file used must contain the same guest name as the guest being upgraded. If a disk image is not specified in the command, then the current guest image is retained for the new configuration. If an ISO file is not specified in the command, the current ISO image of the guest is retained for the new configuration. The files may be local or remote. The existing images are overwritten if there is a conflict with the location where the new images are moved during the upgrade. If the existing guest configuration has more than one disk or one cdrom drive then the extra disks are lost during the upgrade. The `keep-old-images` option prevents the deletion of old images.

Note: During a deployment or upgrade, the disk images are moved from their current, specified location, and not copied, if they are local.

Note: Location of the remotely located files can be influenced by the keyword `routing-instance`.

For a configuration example about upgrading a guest XML file, refer to [Upgrading a guest by using a libvirt domain XML file](#).

For a configuration example about upgrading multiple guest image files, refer to [Upgrading multiple guest images for a guest](#).

For a configuration example about using the `keep-old-images` option, refer to [Upgrading a guest and keeping the old files](#).

Upgrading a guest by using a libvirt domain XML file

You can upgrade a guest configuration by specifying another libvirt domain XML file with the same guest name.

The example that follows uses a guest called `another-guest` with the following key parameters:

- Number of CPUs: 4
- Number of virtual disks: 2
- Location of QCOW2 file for disk 0: `/var/lib/libvirt/images/another-guest/0-disk.qcow2`
- Location of ISO file for disk 1: `/var/lib/libvirt/images/another-guest/1-config.iso`
- Type of disk defined for disk 0: `disk`
- Type of disk defined for disk 1: `cdrom`
- Memory: 4096 MB
- vhost interface: `dp0vhost0`

We can reconfigure the guest by using another XML file with different values for established parameters and a set of additional parameters, for example, a remote display. In reconfiguring the guest, the old parameters are deleted and the guest restarted.

Table 10. Upgrading a guest by using a libvirt domain XML file

Step	Command
Switch to configuration mode.	<pre>tmpuser@node:~\$ config</pre>
Display the details of the guest that is going to be modified.	<pre>tmpuser@node# show virtualization virtualization { guest another-guest { cpus 4 devices { disk 0 { boot-order 1 bus-type virtio device-type disk source { file /var/lib/libvirt/images/another-guest/0-disk.qcow2</pre>

Step	Command
	<pre> } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-config.iso } } network dp0vhost0 { mac-address 52:54:00:16:56:fd } } memory 4096 uuid f7333ab9-8e3d-4a87-a27a-6d5d7d6e99ba } </pre>
Display the bridge details for the guest.	<pre> tmpuser@node# show interfaces bridge bridge br20 </pre>
Display the vhost interfaces for the guest.	<pre> tmpuser@node# show interfaces vhost vhost dp0vhost0 { bridge-group { bridge br20 } description vhost0 } </pre>
Exit the configuration mode.	<pre> tmpuser@node# exit logout </pre>
Apply the new libvirt domain XML file for the guest in operational mode.	<pre> tmpuser@node:~\$ upgrade virtualization xml /home/vyatta/new-another-guest.xml [INFO] - Deleted existing guest "another-guest" config [INFO] - Deleted vhost interface "dp0vhost0" [INFO] - Configured vhost interface "dp0vhost0" [INFO] - Configured vhost interface "dp0vhost1" [INFO] - Configured guest "another-guest" [INFO] - Powering off existing guest "another-guest" [INFO] - Starting guest "another-guest" [INFO] - Configuration saved [INFO] - Upgrade using /home/vyatta/new-another-guest.xml complete </pre>
Switch to the configuration mode.	<pre> tmpuser@node:~\$ config </pre>
Display the new parameters of the guest.	<pre> tmpuser@node# show virtualization virtualization { guest another-guest { cpus 2 devices { disk 0 { boot-order 1 device-type disk source { file /var/lib/libvirt/images/another-guest/0-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-config.iso } } } display { connection-type vnc password "*****" port 5900 } network dp0vhost0 { mac-address 52:54:00:4c:ba:05 } network dp0vhost1 { mac-address 52:54:00:07:f7:24 } } memory 2048 uuid 18949bf1-5d1b-4d7b-b768-f4029d24bdcb } </pre>

Step	Command
Display the vhost interfaces for the guest.	<pre>tmpuser@node# show interfaces vhost vhost dp0vhost0 { address 10.10.10.1/24 } vhost dp0vhost1 { address 10.10.15.1/24 }</pre>

Upgrading multiple guest images for a guest

You can upgrade multiple disks for a guest with one command. Any existing images that conflict with the new images are overwritten during an upgrade. Any images that were previously used by the disks being upgraded are deleted after an upgrade is successful, if they are not overwritten during the upgrade.

The example that follows uses a guest called `another-guest` with the following key parameters for the XML file:

- Number of CPUs: 2
- Number of virtual disks: 2
- Type of disk defined for `disk 0`: disk
- Type of disk defined for `disk 1`: cdrom
- Memory: 2048 MB
- vhost interface: `dp0vhost0` and `dp0vhost1`

We can upgrade the images for both `disk 0` and `disk 1` with a single command. The old images are overwritten.

Table 11. Upgrading multiple guest images for a guest

Step	Command
Switch to configuration mode.	<pre>tmpuser@node:~\$ config</pre>
Display the details of the guest parameters.	<pre>tmpuser@node# show virtualization virtualization { guest another-guest { cpus 2 devices { disk 0 { boot-order 1 device-type disk source { file /var/lib/libvirt/images/another-guest/0-new-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-config.iso } } } display { connection-type vnc password "*****" port 5900 } network dp0vhost0 { mac-address 52:54:00:3f:9a:c4 } network dp0vhost1 {</pre>

- Number of CPUs: 2
- Number of virtual disks: 2
- Type of disk defined for disk 0: disk
- Type of disk defined for disk 1: cdrom
- Memory: 2048 MB
- vhost interface: dp0vhost0 and dp0vhost1

We can upgrade the images for both `disk 0` and `disk 1` with a single command. The old images are not overwritten.

Table 12. Upgrading a guest and keeping the old files

Step	Command
Switch to configuration mode.	<pre>tmpuser@node:~\$ config</pre>
Display the details of the guest parameters.	<pre>tmpuser@node# show virtualization guest guest another-guest { cpus 2 devices { disk 0 { boot-order 1 bus-type virtio device-type disk source { file /var/lib/libvirt/images/another-guest/0-newer-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-newer-config.iso } } } display { connection-type vnc password "*****" port 5900 } network dp0vhost0 { mac-address 52:54:00:3f:9a:c4 } network dp0vhost1 { mac-address 52:54:00:08:0d:ad } } memory 2048 uuid 20b03fea-93b6-454c-b9fd-678dd27aeafa</pre>
Display the vhost interfaces of the guest called <code>another-guest</code> .	<pre>tmpuser@node# show interfaces vhost vhost dp0vhost0 { address 10.10.10.1/24 } vhost dp0vhost1 { address 10.10.15.1/24 }</pre>
Exit the configuration mode.	<pre>tmpuser@node# exit logout</pre>
Apply the new XML file, new guest image, and new ISO file for the guest.	<pre>tmpuser@node:~\$ upgrade virtualization xml /home/vyatta/latest-another-guest.xml image /home/vyatta/latest-disk.qcow2 iso /home/vyatta/latest-config.iso keep-old-images [INFO] - Deleted existing guest "another-guest" config [INFO] - Deleted vhost interface "dp0vhost0" [INFO] - Deleted vhost interface "dp0vhost1" [INFO] - Configured vhost interface "dp0vhost0" [INFO] - Configured guest "another-guest" [INFO] - Powering off existing guest "another-guest" [INFO] - Starting guest "another-guest" [INFO] - Configuration saved [INFO] - Upgrade using /home/vyatta/latest-another-guest.xml complete</pre>

Step	Command
Switch to the configuration mode.	<pre>tmpuser@node:~\$ config</pre>
Display the details of the upgraded guest.	<pre>tmpuser@node# show virtualization guest guest another-guest { cpus 2 devices { disk 0 { boot-order 1 device-type disk source { file /var/lib/libvirt/images/another-guest/0-latest-disk.qcow2 } } disk 1 { boot-order 2 bus-type ide device-type cdrom source { file /var/lib/libvirt/images/another-guest/1-latest-config.iso } } network dp0vhost0 { mac-address 52:54:00:28:5a:ae } } memory 2048 uuid 91903030-2ed7-40f3-a4d9-a7a440de1711 }</pre>
Display the vhost interfaces of the upgraded guest.	<pre>tmpuser@node# show interfaces vhost vhost dp0vhost0 { address 10.10.10.1/24 }</pre>

Chapter 11. Upgrading Guests Commands

upgrade virtualization guest images

Upgrades the guest images for multiple disks of a guest and powers the guest off and on.

```
upgrade virtualization guest guest-name images { device disk-id source location  
image-uri } [ keep-old-images ] [ routing-instance name ]
```

None.

guest-name

The name of a guest.

disk-id

The number for a virtual disk.

image-uri

An image in QCOW2 or ISO-9660 format for the corresponding virtual disk.

keep-old-images

Specifies that the old images are not deleted.

name

Specifies the name of the routing instance to reach the remote host or hosts serving the disk image files. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Operational mode.

Use this command to upgrade the guest images for multiple disks of a guest and power the guest off and on.

If `device-type` is `disk`, the QCOW2 format is supported. If device type is `cdrom`, the ISO-9660 format is supported.

The command replaces the disk image of a virtual disk for a guest with another guest image. The command also powers the guest off and on. The image may be local or remote. When a routing instance is specified, it is used to reach all remote hosts that are specified. For information about VRF configuration, refer to the VRF Support chapter in the *Basic Routing Configuration Guide*. You must use fully qualified path names for the image files. You can upgrade multiple disks for a guest with one command. Any existing images that conflict with the new images are overwritten during an upgrade. The configuration is saved if changes were made to it during the upgrade. Any images which were previously used by the disks being upgraded are deleted after an upgrade, if they are not overwritten during the upgrade. The `keep-old-images` option prevents the deletion of old images.

During a deployment or upgrade, the disk images are moved from their current, specified location, and not copied, if they are local.

For more information, refer to [Upgrading Guests](#).

upgrade virtualization xml

Reconfigures a guest according to a libvirt domain XML file and optionally upgrade the guest image and ISO files.

```
upgrade virtualization xml xml-file [ image image-file] [ iso iso-file ] [ keep-old-images ] [ routing-instance name ]
```

None.

xml-file

A libvirt domain XML file.

image-file

An image file.

iso-file

An ISO file.

keep-old-images

Specifies the retention of old images.

name

Specifies the name of the routing instance to reach the remote host or hosts serving the XML, disk image or ISO files. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Operational mode.

Use this command to reconfigure a guest according to a libvirt domain XML file and optionally upgrade the guest image and ISO files.

The command reconfigures an existing guest according to the parameters that are defined within an XML file. The XML file must contain the same guest name as the guest being upgraded. If a disk image is not specified in the command, the current guest image is retained for the new configuration. If an ISO file is not specified in the command, the current ISO image of the guest is retained for the new configuration. The files may be local or remote. When a routing instance is specified, it is used to reach all remote hosts that are specified. Use fully qualified path names for the files. If an image or ISO file is specified, any existing files that conflict are overwritten during the upgrade. If the existing guest configuration has more than one disk or CDROM drive, the extra disks are lost during the upgrade. The configuration is saved if changes were made to it during the upgrade. The `keep-old-images` option prevents the deletion of old images.

During a deployment or upgrade, the disk images are moved from their current, specified location, and not copied, if they are local.

Chapter 12. Guest Lifecycle Commands

add virtualization guest snapshot description

Creates a snapshot of a guest with an optional description.

```
add virtualization guest guest-name snapshot snapshot-name [ description description ]
```

guest-name

A guest name.

snapshot-name

A snapshot name.

description

A description of the snapshot.

Operational mode.

Use this command to create a snapshot of a guest.

The following example shows how to create a snapshot.

```
tmpuser@node:~$ add virtualization guest vr5600 snapshot testing1  
description testMonday
```

copy virtualization image to

Copies an image to a new path relative to the `/var/lib/libvirt/images` folder.

```
copy virtualization image image to new-path
```

image

An image file relative to `/var/lib/libvirt/images`.

new-path

An image file location relative to `/var/lib/libvirt/images`.

Operational mode.

Use this command to copy an image to a new path. All the image paths are relative to the `/var/lib/libvirt/images` folder.

Any existing image at the new path is overwritten.

The following example shows how to copy an image to a new path.

```

tmpuser@node:~$ show virtualization images
an-image.qcow2

tmpuser@node:~$ copy virtualization image an-image.qcow2 to an-image-
copy.qcow2

tmpuser@node:~$ show virtualization images
an-image-copy.qcow2
an-image.qcow2

```

delete virtualization guest snapshot

Deletes a snapshot of a guest.

```
delete virtualization guest guest-name snapshot snapshot-name
```

guest-name

A guest name.

snapshot-name

A snapshot name.

Operational mode.

Use this command to delete a snapshot of a guest.

The following example shows how to delete a snapshot.

```

tmpuser@node:~$ show virtualization guest vr5600 snapshots

Snapshot      Date              Description
-----      -
test1         2015-11-27T07:59:57+00:00
test2         2015-12-09T14:38:59+00:00
testing1      2015-12-09T14:43:03+00:00      testMonday

tmpuser@node:~$ delete virtualization guest vr5600 snapshot testing1

tmpuser@node:~$ show virtualization guest vr5600 snapshots

Snapshot      Date              Description
-----      -
test1         2015-11-27T07:59:57+00:00
test2         2015-12-09T14:38:59+00:00

tmpuser@node:~$

```

delete virtualization image

Deletes an image from the `/var/lib/libvirt/images` folder.

```
delete virtualization image image
```

image

An image file relative to `/var/lib/libvirt/images`.

Operational mode.

Use this command to delete an image from the `/var/lib/libvirt/images` folder.

An image is not deleted if it is referenced by a guest configuration.

The following example shows how to delete an image.

```
tmpuser@node:~$ show virtualization images
an-image.qcow2
moved-image.qcow2

tmpuser@node:~$ delete virtualization image moved-image.qcow2

tmpuser@node:~$ show virtualization images
an-image.qcow2
```

download virtualization image from

Downloads a remote image to the `/var/lib/libvirt/images` folder by using HTTP, HTTPS, FTP, SFTP, or SCP.

```
download virtualization image from uri [ to image-path ] [ authentication
username username-text [ password password-text ] ] [ disable-verification ] [
routing-instance name ]
```

uri

An HTTP, HTTPS, FTP, SFTP, or SCP location for an image.

image-path

An image location relative to the `/var/lib/libvirt/images` folder.

username *username-text*

Specifies a username.

password *password-text*

Specifies a password. The system prompts for a password, if one is not specified.

disable-verification

Disables the SSL certificate verification or SSH host key verification.

name

Specifies the name of the routing instance to reach the host serving the image. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Operational mode.

Use this command to download a remote image to the `/var/lib/libvirt/images` folder by using HTTP, HTTPS, FTP, SFTP, or SCP.

For more information about VRF configuration, refer to the VRF Support chapter in the *Basic Routing Configuration Guide*.

The `download virtualization image` command displays the following information:

Output field	Description
% Total	Size of the image file in the remote location.
% Received	Size of the image file received.
% Xferd	Size of the uploaded image file. Not relevant for downloads.
Average Speed Dload	Average download speed.
Average Speed Upload	Average upload speed. Not relevant for downloads.
Time Total	Total time required for the download.
Time Spent	Time spent since the download started.
Time Left	Time left to complete the download.
Current Speed	Current speed of the download.

The command output is not displayed until the download has finished.

The following example shows how to download an image from an SCP location.

```
tmpuser@node:~$ download virtualization image from scp://10.10.1.1/home/
image-user/another-image.qcow2 authentication username image-user
```

```
Enter password for 'image-user':
```

```

% Total      % Received % Xferd   Average Speed   Time    Time       Time
Current
                                     Dload  Upload   Total   Spent    Left
Speed
100 292M  100 292M    0     0 31.9M      0  0:00:09  0:00:09  --:--:--
32.0M
100 292M  100 292M    0     0 31.9M      0  0:00:09  0:00:09  --:--:--
31.9M
```

move virtualization image to

Moves an image to a new path relative to the `/var/lib/libvirt/images` folder.

```
move virtualization image image to new-path
```

image

An image file relative to `/var/lib/libvirt/images`.

new-path

An image file location relative to `/var/lib/libvirt/images`.

Operational mode.

Use this command to move an image to a new path. All the image paths are relative to the `/var/lib/libvirt/images` folder.

An image is not moved if the original path is referenced by a guest configuration.

Any existing image at the new path is overwritten.

The following example shows how to move an image to a new path.

```
tmpuser@node:~$ show virtualization images
an-image.qcow2

tmpuser@node:~$ move virtualization image an-image.qcow2 to moved-
image.qcow2

tmpuser@node:~$ show virtualization images
moved-image.qcow2
```

restart virtualization guest

Powers down a guest and then powers it on.

```
restart virtualization guest guest-name
```

guest-name

A guest name.

Operational mode.

Use this command to restart a guest.

If the guest does not respond to the shutdown request, then the command forcibly powers off the guest after 300 seconds.

The following example shows how to restart a guest.

```
tmpuser@node:~$ restart virtualization guest aguest
```


reset virtualization guest to snapshot

Rolls back a guest to a previous snapshot state.

```
reset virtualization guest guest-name to snapshot snapshot-name
```

guest-name

A guest name.

snapshot-name

A snapshot name.

Operational mode.

Use this command to roll back a guest to a previous state.

The following example shows how to reset a guest to a previous state.

```
tmpuser@node:~$ reset virtualization guest aguest to snapshot test1
tmpuser@node:~$
```

virtualization guest autostart

Sets the autostart options of the guest.

```
set virtualization guest guest-name { true | false }
```

```
delete virtualization guest guest-name { true | false }
```

```
show virtualization guest guest-name { true | false }
```

The default autostart option is true.

probe-name

A name for a probe.

true

Enables guest autostart.

false

Disables guest autostart.

Configuration mode.

```
virtualization
guest <guest name> {
    autostart { true | false}
```

```
}
```

Use this command to set autostart of the guest.

Use the **set** form of this command to set the autostart of the guest.

Use the **delete** form of this command to set the default autostart of the guest, which is true.

Use the **show** form of this command to display the autostart of the guest.

poweroff virtualization guest

Powers down a guest.

```
poweroff virtualization guest guest-name
```

guest-name

A guest name.

Operational mode.

Use this command to power down a guest. The guest must be running when you enter the command.

If the guest does not respond to the shutdown request, then the command forcibly powers off the guest after 300 seconds.

The following example shows how to power down a guest.

```
ttmpuser@node:~$ poweroff virtualization guest test1
```

start virtualization guest

Starts a guest.

```
start virtualization guest guest-name
```

guest-name

A guest name.

Operational mode.

Use this command to start a guest. The guest must already be configured and must not be running when you enter the command.

The following example shows how to start a guest.

```
tmpuser@node:~$ poweroff virtualization guest test1
tmpuser@node:~$ start virtualization guest test1
```

show virtualization images

Displays a list of all images available in the `/var/lib/libvirt/images` folder.

```
show virtualization images
```

Operational mode.

Use this command to display a list of all images available in the `/var/lib/libvirt/images` folder.

The following example shows how to display a list of images available in the `/var/lib/libvirt/images` folder. .

```
tmpuser@node:~$ show virtualization images
an-image.qcow2
moved-image.qcow2
```

show virtualization status

Displays the status of guests.

```
show virtualization status
```

Operational mode.

Use this command to display the status of guests.

The `show virtualization status` command displays the following information:

Output field	Description
Name	Name of a guest.
State	State of a guest, such as running.

The following example shows how to display the status of guests.

```
tmpuser@node:~$ show virtualization status
Name          State
----          -
vr5600        running
```

show virtualization guest snapshots

Displays all available snapshots for a specified guest.

```
show virtualization guest guest-name snapshots
```

guest-name

A unique guest name.

Operational mode.

Use this command to display all available snapshots for a guest.

The `show virtualization guest <guest-name> snapshots` command displays the following information:

Output field	Description
Snapshot	Name of a snapshot.
Date	Date when the snapshot was created.
Description	Description of the snapshot.

The following example shows how to display the list of available snapshots for a guest.

```
tmpuser@node:~$ show virtualization guest another-guest snapshots

Snapshot      Date                Description
-----
sn0           2015-08-26T16:35:02+00:00  Test1
sn1           2015-08-27T15:01:08+00:00  Test2
```

upload virtualization image to

Uploads an image located relative to the `/var/lib/libvirt/images` folder to the remote server by using FTP, SFTP, or SCP.

```
upload virtualization image image to uri [ authentication username username-text [ password password-text ] ] [ disable-verification ] [ routing-instance name ]
```

image

An image location relative to the `/var/lib/libvirt/images` folder.

uri

An FTP, SFTP, or SCP location for an image.

username *username-text*

Specifies a username.

password *password-text*

Specifies a password. The system prompts for a password, if one is not used.

disable-verification

Disables the SSL certificate verification or SSH host key verification.

name

Specifies the name of the routing instance to reach the host on which the image must be uploaded to. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Operational mode.

Use this command to upload an image located relative to the `/var/lib/libvirt/images` folder to the remote server by using FTP, SFTP, or SCP.

For information about VRF configuration, refer to the VRF Support chapter in the *Basic Routing Configuration Guide*.

The `upload virtualization image` command displays the following information:

Output field	Description
% Total	Size of the uploaded image file.
% Received	Size of the image file received. Not relevant for uploads.
% Xferd	Size of the uploaded image.
Average Speed Dload	Average speed of the download. Not relevant for uploads.
Average Speed Upload	Average speed of the upload.
Time Total	Total time required for the upload.
Time Spent	Time spent since the upload started.
Time Left	Time left to complete the upload.
Current Speed	Current speed of the upload.

The command output is not displayed until the download has finished.

The following example shows how to upload an image to an SCP location.

```

tmpuser@node:~$ upload virtualization image another-image.qcow2 to
scp://10.10.1.1/home/image-user/another-image-used.qcow2 authentication
username image-user
Enter password for 'image-user':
  % Total      % Received % Xferd  Average Speed   Time    Time       Time
  Current
                                 Dload  Upload   Total   Spent    Left
  Speed
100 292M    0      0    0 292M      0 48.6M  --:--:--  0:00:06  --:--:--
48.6M

```


```
100 292M 0 0 0 292M 0 48.6M ---:---:-- 0:00:06 ---:---:--  
48.6M
```


Chapter 13. Monitoring Guests

Monitoring guests overview

The guest probe feature provides the ability to monitor the vitality of guests that run on top of the VNF platform hypervisor. You can monitor each guest by configuring an SNMP, an HTTP, an HTTPS or a ping probe to check for expected responses at a target IP address. The probe runs at a configurable interval and can take action on the associated guest after a configurable number of consecutive failures.

Multiple independent probes are supported, each with a single type of SNMP, HTTP, HTTPS or ping. You can separately configure the failure action, interval, retries, and guest start-up interval.

 **Note:** Only one probe for each guest is supported.

 **Note:** The probe service is routing-instance aware. It is possible for guest VMs to be configured so that they can only be network reachable by using a particular routing instance.

Probe type

The following sections provide specific information about each type of VNF platform probe.

Ping probe

A ping probe sends an ICMP echo request to the target. It then expects an ICMP echo reply. If the response is not received, the probe fails.

HTTP probe

The HTTP probe sends an HTTP GET request to the target and expects a particular status code in response. The probe fails if there is no response or a different status code is received. Following are the parameters of an HTTP probe:

- Path—Server document path, which defaults to /.
- Port—Server port, which defaults to 80.
- Status code—Expected HTTP status code, which defaults to 200.

HTTPS probe

The HTTPS probe sends an HTTPS GET request to the target and expects a particular status code in response. The probe fails if there is no response or a different status code is received. An invalid SSL certificate also causes a failure by default. Following are the parameters of an HTTPS probe:

- Path—Server document path, which defaults to /.
- Port—Server port, which defaults to 443.
- Status code—Expected HTTPS status code, which defaults to 200.
- Certificate validation—Certificate validation enforcement enabled by default.

SNMP Probe

The SNMP probe sends an SNMP GET request to the target and expects a particular value at an OID. The probe fails if there is no response, or a different value is received. Following are the parameters of an SNMP probe:

- OID—Object identifier, in numbered dot notation, to look up.
- Value—Expected value.
- SNMP version—SNMP versions, which are 1 or 2.
- Community—SNMP community string, which defaults to public.

Probe action overview

A waiting period is required for the guest to boot and guest services to be available. The defined guest action is not executed until this period has expired. Probing may begin earlier if multiple guest actions with different waits are configured on a probe. Only one probe can take action on a guest. The startup wait must be long enough to allow the guest to come up in normal operation and the waiting period ranges from 1 to 30 minutes. The default waiting period is 5 minutes.

Following are available actions for guest probes:

- **none**—Takes no action.
- **reset**—Performs a forceful reboot of the guest.
- **poweroff**—Performs a forceful shutdown of the guest.

Logging overview

Whenever the probe service takes action on a guest that has stopped responding to probes, the probe service logs an appropriate message in the `/var/log/messages` file. By default, messages are logged only when probes fail or an action is taken. With the debug option enabled, additional messages are logged when a probe is sent, a response is received, or the probe times out.

The following section provides an example of nondebug log entries. In the example, the **foo** probe fails three times and causes the `a-guest` guest to be reset.

```
2015-10-14T01:12:17.882305+00:00 localhost probed: Probe foo failed
2015-10-14T01:12:43.440805+00:00 localhost probed: message repeated 3
times: [ Probe foo failed]
2015-10-14T01:12:43.441094+00:00 localhost probed: Sequential failures for
probe foo have exceeded the threshold (3), running actions
```



```
2015-10-14T01:12:43.452570+00:00 localhost probed: a-guest reset performed
```

Configuring Probes

The following sections provide configuration examples for these probes:

- Ping
- HTTP
- HTTPS
- SNMP

Configuring a guest ping probe

The following example shows how to configure a guest ping probe with the following parameters:

- Probe type: `ping`
- Probe: `my-guest-ping-probe`
- Network target: `10.10.10.10`
- Routing instance: `blue`
- Action: `reset`

In this example, an ICMP echo request is sent to the `10.10.10.10` network target in the `blue` routing instance every 10 seconds after the `a-guest` guest has been running for at least 3 minutes. If no reply is received after three consecutive attempts, then the guest is forcibly reset.

To configure the guest probe and ping the guest, follow these steps in configuration mode.

Table 13. Configuring a guest ping probe

Step	Command
Set the probe name.	<code>tmpuser@node# set service probes probe my-guest-ping-probe</code>
Set the probe type.	<code>tmpuser@node# set service probes probe my-guest-ping-probe type network ping</code>
Set the target IP address of the probe.	<code>tmpuser@node# set service probes probe my-guest-ping-probe type network target 10.10.10.10</code>
Set the routing instance of the probe.	<code>tmpuser@node# set service probes probe my-guest-ping-probe type network routing-instance blue</code>
Set the guest action of the probe.	<code>tmpuser@node# set service probes probe my-guest-ping-probe action virtualization guest a-guest action reset</code>
Set the guest startup waiting time.	<code>tmpuser@node# set service probes probe my-guest-ping-probe action virtualization guest a-guest startup-wait 3</code>

Step	Command
Commit the changes.	<pre>tmpuser@node# commit</pre>
Save the changes.	<pre>tmpuser@node# save</pre>
Display the results.	<pre>tmpuser@node# show service probes probes { probe my-guest-ping-probe { action { virtualization { guest a-guest { action reset startup-wait 3 } } } type { network { ping target 10.10.10.10 routing-instance blue } } } }</pre>

Configuring a guest HTTP probe

The following example shows how to configure a guest HTTP probe with the following parameters:

- Probe type: `http`
- Probe: `my-guest-http-probe`
- Network target: `10.1.1.2`
- Path: `/`
- Port: `80`
- Status Code: `200`
- Action: `poweroff`

In this example, an HTTP GET request is sent to the `10.1.1.2` network target in the default routing instance and expects an HTTP status code in return. The probe fails if there is no response or a status code other than 200 is received.

To configure the guest HTTP probe, follow these steps in configuration mode.

Table 14. Configuring a guest HTTP probe

Step	Command
Set the probe name.	<pre>tmpuser@node# set service probes probe my-guest-http-probe</pre>
Setting the probe type to HTTP.	<pre>tmpuser@node# set service probes probe my-guest-http-probe type network http</pre>
Set the target IP address of the probe.	<pre>tmpuser@node# set service probes probe my-guest-http-probe type network target 10.1.1.2</pre>

Step	Command
Set the HTTP server document path.	<pre>tmpuser@node# set service probes probe my-guest-http-probe type network http path /</pre>
Set the HTTP server port.	<pre>tmpuser@node# set service probes probe my-guest-http-probe type network http port 80</pre>
Set the expected HTTP status code.	<pre>tmpuser@node# set service probes probe my-guest-http-probe type network http status-code 200</pre>
Set up the guest action	<pre>tmpuser@node# set service probes probe my-guest-http-probe action virtualization guest a-guest action poweroff</pre>
Commit the changes.	<pre>tmpuser@node# commit</pre>
Save the changes.	<pre>tmpuser@node# save</pre>
Display the results.	<pre>tmpuser@node# show service probes probe my-guest-http-probe probe my-guest-http-probe { action { virtualization { guest a-guest { action poweroff } } } type { network { http { path / port 80 status-code 200 } target 10.1.1.2 } } }</pre>

Configuring a guest HTTPS probe

The following example shows how to configure a guest HTTPS probe with the following parameters:

- Probe type: `https`
- Probe: `my-guest-https-probe`
- Network target: `10.1.1.2`
- Path: `/`
- Port: `443`
- Status code: `200`
- Certificate validation: `true`
- Action: `poweroff`

In this example, an HTTPS GET request is sent to the `10.1.1.2` network target in the default routing instance and expects an HTTPS status code in return. The probe fails if there is no response or a status code other than 200 is received or the server uses an invalid SSL certificate.

To configure the guest HTTPS probe, follow these steps in configuration mode.

Table 15. Configuring a guest HTTPS probe

Step	Command
Set the probe name.	<code>tmpuser@node# set service probes probe my-guest-https-probe</code>
Set the probe type to HTTPS.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network https</code>
Set the target IP address of the probe.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network target 10.1.1.2</code>
Set the HTTPS server document path.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network https path /</code>
Set the HTTPS server port.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network https port 443</code>
Set the expected HTTPS status code.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network https status-code 200</code>
Setup the certificate validation.	<code>tmpuser@node# set service probes probe my-guest-https-probe type network https certificate-validation true</code>
Set up the guest action	<code>tmpuser@node# set service probes probe my-guest-https-probe action virtualization guest a-guest action poweroff</code>
Commit the changes.	<code>tmpuser@node# commit</code>
Save the changes.	<code>tmpuser@node# save</code>
Display the results.	<pre>tmpuser@node# show service probes probe my-guest-https-probe probe my-guest-https-probe { action { virtualization { guest a-guest { action poweroff } } } type { network { https { certificate-validation true path / port 80 status-code 100 } target 10.1.1.2 } } }</pre>

Configuring a guest SNMP probe

The following example shows how to configure a guest SNMP probe with the following parameters:

- Probe type: `snmp`
- Probe: `my-guest-snmp-probe`
- Network target: `10.1.1.2`
- OID: `1.3.6.1.2.1.1.1.0`
- SNMP protocol version: `v2`

- **Community:** public
- **Value:** text string
- **Guest:** vr5600
- **Action:** reset

In this example, an SNMP GET request is sent to the 10.1.1.2 network target in the default routing instance and expects the OID to return a value. The probe fails if there is no response or a different value is returned.

To configure the guest SNMP probe, follow these steps in configuration mode.

Table 16. Configuring a guest SNMP probe

Step	Command
Set the probe name.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe</code>
Set the probe type to SNMP.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network snmp</code>
Set the target IP address of the probe.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network target 10.1.1.2</code>
Set the OID.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network snmp oid 1.3.6.1.2.1.1.1.0</code>
Set the SNMP protocol version.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network snmp v2</code>
Set the SNMP community.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network snmp v2 community public</code>
Set the expected value at the OID.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe type network snmp value "text string"</code>
Set up the guest action.	<code>tmpuser@node# set service probes probe my-guest-snmp-probe action virtualization guest vr5600 action reset</code>
Commit the changes.	<code>tmpuser@node# commit</code>
Save the changes.	<code>tmpuser@node# save</code>
Display the results.	<pre>tmpuser@node# show service probes probe my-guest-snmp-probe probe my-guest-snmp-probe { action { virtualization { guest vr5600 { action reset } } } type { network { snmp { oid 1.3.6.1.2.1.1.1.0 v2 { community public } value "text string" } target 10.1.1.2 } } }</pre>

Chapter 14. Monitoring Guests Commands

service probes debug

Enables verbose probe logging.

```
set service probes debug
```

```
delete service probes debug
```

```
show service probes
```

None.

debug

Enables verbose probe logging.

Configuration mode

```
service probes debug
```

Use this command to enable verbose probe logging. The probe service logs are stored in `/var/log/messages`. Run the `grep probed /var/log/messages` command to extract and display the probe log messages from the system log. To view the additional debug messages, the daemon syslog facility must be configured to at least the info level.

Use the **set** version of the command to enable verbose logging.

Use the **delete** version of the command to disable verbose logging.

Use the **show** version of the command to display the status of all probes.

service probes probe

Creates a probe with a specific probe name.

```
set service probes probe probe-name
```

```
delete service probes probe probe-name
```

```
show service probes probe probe-name
```

None

probe-name

The name of a probe, which must be alphanumeric.

Configuration mode.

Use this command to create a probe with a specific probe name.

Use the **set** version of the command to create a probe with a specific probe name.

Use the **delete** version of the command to delete a probe with a specific probe name.

Use the **show** form of the command to display a probe with a specific probe name.

service probes probe type network target

Sets the target IP address of a probe.

```
set service probes probe probe-name type network target ip-address
```

```
delete service probes probe probe-name type network target ip-address
```

```
show service probes probe probe-name type network target
```

None.

probe-name

The name of a probe.

ip-address

The IPv4 or IPv6 address of a target.

Configuration mode.

```
service probes probe probe-name {
    type {
        network {
            target ip-address
        }
    }
}
```

Use this command to set the target IP address of a probe.

Use the **set** form of this command to set the target IP address of a probe.

Use the **delete** form of this command to remove the target IP address of a probe.

Use the **show** form of this command to display the target IP address of a probe.

service probes probe type network routing-instance

Sets the routing instance of a probe.

```
set service probes probe probe-name type network routing-instance name
```

```
delete service probes probe probe-name type network routing-instance name
```

```
show service probes probe probe-name type network routing-instance
```

None.

probe-name

The name of a probe.

name

Specifies the name of the routing instance to reach the probe target. The default routing-instance is used if a value for the routing-instance keyword is not specified.

Configuration mode.

```
service probes probe probe-name {
  type {
  network {
    routing-instance name
  }
}
}
```

Use this command to set the routing instance of a probe. If a routing instance is not set, the default routing instance is used. For more information about VRF configuration, refer to the VRF Support chapter in the *Basic Routing Configuration Guide*.

Use the **set** form of this command to set the routing instance of a probe.

Use the **delete** form of this command to remove the routing instance of a probe.

Use the **show** form of this command to display the routing instance of a probe.

service probes probe interval

Sets the interval between probe attempts.

```
set service probes probe probe-name interval interval
```

```
delete service probes probe probe-name interval interval
```

```
show service probes probe probe-name interval
```

The default interval is 10 seconds.

probe-name

A name for a probe.

interval-number

An interval in seconds between probe attempts.

Configuration mode.


```
service probes probe probe-name {
    interval interval-number
}
```

Use this command to set the interval between probe attempts.

Use the **set** form of this command to set the interval between probe attempts.

Use the **delete** form of this command to set the interval to the default, which is 10 seconds.

Use the **show** form of this command to display the interval between probe attempts.

service probes probe retries

Sets the number of consecutive probe failures before action is taken.

```
set service probes probe probe-name retries retries
```

```
delete service probes probe probe-name retries retries
```

```
show service probes probe probe-name retries
```

The default number of retries is 3.

probe-name

A name for a probe.

retries

A number of consecutive probe failures before acting.

Configuration mode.

```
service probes probe probe-name {
    retries retries-number
}
```

Use this command to set the number of consecutive probe failures before action is taken.

Use the **set** form of this command to set the number of consecutive probe failures before action is taken.

Use the **delete** form of this command to set the default number of consecutive probe failures, which is 3, before action is taken.

Use the **show** form of this command to display the number of consecutive probe failures before action is taken.

service probes probe type network ping

Enables the ping probe.

```
set service probes probe probe-name type network ping
delete service probes probe probe-name type network ping
show service probes probe probe-name type network
```

probe-name

A name for a probe.

Configuration mode

```
service probes probe probe-name type
network {
    ping
}
```

Use this command to enable the ping probe.

Use the **set** form of this command to enable the probe ping.

Use the **delete** form of this command to remove the ping probe.

Use the **show** form of this command to display the ping probe.

service probes probe type network http

Enables the HTTP probe.

```
set service probes probe probe-name type network http
delete service probes probe probe-name type network http
show service probes probe probe-name type network
```

probe-name

A name for a probe.

Configuration mode.

```
service probes probe probe-name type
network {
    http
}
```

Use this command to enable the HTTP probe.

Use the **set** form of this command to enable the HTTP probe.

Use the **delete** form of this command to remove the HTTP probe.

Use the **show** form of this command to display the HTTP probe.

service probes probe type network http path

Sets the document path on the HTTP server.

```
set service probes probe probe-name    type network http path path-text
delete service probes probe probe-name  type network http path path-text
show service probes probe probe-name    type network http path
```

The default value is /.

probe-name

A name for a probe.

path-text

The document path to the HTTP server.

Configuration mode.

```
service probes probe probe-name type
network {
    http {
        path path-text
    }
}
```

Use this command to set the document path on the HTTP server.

Use the **set** form of this command to set the document path on the HTTP server.

Use the **delete** form of this command to set the default document path on the HTTP server.

Use the **show** form of this command to display the document path on the HTTP server.

service probes probe type network http port

Sets the server port number for the HTTP probe.

```
set service probes probe probe-name    type network http port port-number
delete service probes probe probe-name  type network http port port-number
show service probes probe probe-name    type network http port
```

The default port is 80.

probe-name

A name for a probe.

port-number

The number of a server port.

Configuration mode.

```
service probes probe probe-name type
network {
    http {
        port port-number
    }
}
```

Use this command to set the server port number for the HTTP probe.

Use the **set** form of this command to set the server port number for the HTTP probe.

Use the **delete** form of this command to set the default server port number, which is 80.

Use the **show** form of this command to display the server port number for the HTTP probe.

service probes probe type network http status-code

Set the HTTP status code as a response from the network target.

```
set service probes probe probe-name type network http status-code code-number
```

```
delete service probes probe probe-name type network http status-code code-number
```

```
show service probes probe probe-name type network http status-code
```

The default status code is 200.

probe-name

A name for a probe.

code-number

An expected HTTP status code.

Configuration mode.

```
service probes probe probe-name type
network {
    http {
        status-code code-number
    }
}
```

```
}
```

Use this command to set the HTTP status code as a response from the network target.

Use the **set** form of this command to set the HTTP status code as a response from the network target.

Use the **delete** form of this command to sets the default HTTP status code as a response from the network target.

Use the **show** form of this command to display the HTTP status code as a response from the network target.

service probes probe type network https

Enables the HTTPS probe.

```
set service probes probe probe-name type network https
```

```
delete service probes probe probe-name type network https
```

```
show service probes probe probe-name type network
```

probe-name

A name for a probe.

https

Specifies an HTTPS request probe.

Configuration mode

```
service probes probe probe-name type
network {
    https
}
```

Use this command to enable the HTTPS probe.

Use the **set** form of this command to enable the HTTPS probe.

Use the **delete** form of this command to remove the HTTPS probe.

Use the **show** form of this command to display the HTTPS probe.

service probes probe type network https path

Sets the document path on the HTTPS server.

```
set service probes probe probe-name type network https path path-text
```

```
delete service probes probe probe-name type network https path path-text
```

```
show service probes probe probe-name type network https path
```

The default value is /.

probe-name

A name for a probe.

path-text

The server document path.

Configuration mode.

```
service probes probe probe-name type
network {
    https {
        path path-text
    }
}
```

Use this command to set the document path on the HTTPS server.

Use the **set** form of this command to set the document path on the HTTPS server.

Use the **delete** form of this command to sets the default document path on the HTTPS server.

Use the **show** form of this command to display the document path on the HTTPS server.

service probes probe type network https port

Sets the server port number for the HTTPS probe.

```
set service probes probe probe-name type network https port port-number
```

```
delete service probes probe probe-name type network https port port-number
```

```
show service probes probe probe-name type network https port
```

The default port is 443.

probe-name

A name for a probe.

port-number

The number of a server port.

Configuration mode.

```
service probes probe probe-name type
network {
    https {
```

```

    port port-number
  }
}

```

Use this command to set the server port number for the HTTPS probe.

Use the **set** form of this command to set the server port number for the HTTPS probe.

Use the **delete** form of this command to set the default server port number, which is 443.

Use the **show** form of this command to display the server port number for the HTTPS probe.

service probes probe type network https status-code

Set the HTTPS status code as a response from the network target.

```
set service probes probe probe-name type network https status-code code-number
```

```
delete service probes probe probe-name type network https status-code code-number
```

```
show service probes probe probe-name type network https status-code
```

The default status code is 200.

probe-name

A name for a probe.

code-number

An expected HTTPS status code.

Configuration mode.

```

service probes probe probe-name type
network {
    https {
        status-code code-number
    }
}

```

Use this command to set the HTTPS status code as a response from the network target.

Use the **set** form of this command to set the HTTPS status code as a response from the network target.

Use the **delete** form of this command to sets the default HTTPS status code as a response from the network target.

Use the **show** form of this command to display the HTTPS status code as a response from the network target.

service probes probe type network https certificate-validation

Sets the certificate validation.

```
set service probes probe probe-name type network https certificate-validation {
true | false }
```

```
delete service probes probe probe-name type network https certificate-
validation { true | false }
```

```
show service probes probe probe-name type network https certificate-validation
```

The default certificate validation is true.

probe-name

A name for a probe.

true

Enforces certificate validation

false

Disables certificate validation

Configuration mode.

```
service probes probe probe-name type
network {
    https {
        certificate-validation { true | false }
    }
}
```

Use this command to set certificate validation.

Use the **set** form of this command to set the certificate validation.

Use the **delete** form of this command to set the default certificate validation, which is true.

Use the **show** form of this command to display the certificate validation.

service probes probe type network snmp

Sets the SNMP probe.

```
set service probes probe probe-name type network snmp
```

```
delete service probes probe probe-name type network snmp
```

```
show service probes probe probe-name type network
```

probe-name

A name for a probe.

Configuration mode

```
service probes probe probe-name type
network {
    snmp
}
```

Use this command to set the SNMP probe.

Use the **set** form of this command to set the SNMP probe.

Use the **delete** form of this command to remove the SNMP probe.

Use the **show** form of this command to display the SNMP probe.

service probes probe type network snmp oid

Sets the Object Identifier (OID) value for the SNMP probe.

```
set service probes probe probe-name type network snmp oid oid-value
```

```
delete service probes probe probe-name type network snmp oid oid-value
```

```
show service probes probe probe-name type network snmp oid
```

None

probe-name

A name for a probe.

oid-value

An object identifier, in numbered-dot notation, to look up.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        oid oid-value
    }
}
```

Use this command to set the OID value for the SNMP probe.

Use the **set** form of this command to set the OID value for the SNMP probe.

Use the **delete** form of this command to remove the OID value for the SNMP probe.

Use the **show** form of this command to display the the OID value for the SNMP probe.

service probes probe type network snmp value

Sets the SNMP MIB value for the SNMP probe.

```
set service probes probe probe-name type network snmp value value-snmp-mib
```

```
delete service probes probe probe-name type network snmp value value-snmp-mib
```

```
show service probes probe probe-name type network snmp value
```

None

probe-name

A name for a probe.


value-snmp-mib

The expected value to be returned from the SNMP MIB.

The value must be in the format that is returned by the `snmpget` request `snmpget: $`

```
snmpget -v2c -c public 10.10.10.10 1.3.6.1.2.1.1.1.0
```

```
SNMPv2-MIB::sysDescr.0 = STRING: Vyatta 4.0.0.R1.11022250.
```

 **Note:** In the preceding sample output, `STRING: Vyatta 4.0.0.R1.11022250` is an example.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        value value-snmp-mib
    }
}
```

Use this command to set the SNMP MIB value for the SNMP probe.

Use the **set** form of this command to set the SNMP MIB value for the SNMP probe.

Use the **delete** form of this command to remove the SNMP MIB value for the SNMP probe.

Use the **show** form of this command to display the the SNMP MIB value for the SNMP probe.

service probes probe type network snmp v1

Sets the version of SNMP to 1.

```
set service probes probe probe-name type network snmp v1
```

```
delete service probes probe probe-name type network snmp v1
```

```
show service probes probe probe-name type network snmp v1
```

None

probe-name

A name for a probe.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        v1
    }
}
```

Use this command to set the version of SNMP to 1.

Use the **set** form of this command to set the version of SNMP to 1.

Use the **delete** form of this command to remove the SNMP version.

Use the **show** form of this command to display the SNMP version.

service probes probe type network snmp v2

Sets the version of the SNMP to 2.

```
set service probes probe probe-name type network snmp v2
```

```
delete service probes probe probe-name type network snmp v2
```

```
show service probes probe probe-name type network snmp v2
```

None

probe-name

A name for a probe.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        v2
    }
}
```

Use this command to set the version of the SNMP to 2.

Use the **set** form of this command to set the version of the SNMP to 2.

Use the **delete** form of this command to remove the version of the SNMP.

Use the **show** form of this command to display the version of the SNMP.

service probes probe type network snmp v1 community

Sets the SNMP v1 community string.

```
set service probes probe probe-name type network snmp v1 community text string
```

```
delete service probes probe probe-name type network snmp v1 community text string
```

```
show service probes probe probe-name type network snmp v1 community
```

The default community string is public.

probe-name

A name for a probe.

text string

An SNMP community string.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        v1 {
            community "text string"
        }
    }
}
```

Use this command to set the community string for version 1 of SNMP.

Use the **set** form of this command to set the community string for version 1 of SNMP.

Use the **delete** form of this command to remove the default SNMP v1 community string.

Use the **show** form of this command to display the community string for version 1 of SNMP.

service probes probe type network snmp v2 community

Sets the community string for version 2 of SNMP.

```
set service probes probe probe-name type network snmp v2 community some text string
```

```
delete service probes probe probe-name type network snmp v2 community some text string
```

```
show service probes probe probe-name type network v2 community
```

The default value is public.

probe-name

A name for a probe.

some text string

An SNMP community string.

Configuration mode.

```
service probes probe probe-name type
network {
    snmp {
        v2 {
            community "some text string"
        }
    }
}
```

Use this command to set the community string for version 2 of SNMP.

Use the **set** form of this command to set the community string for version 2 of SNMP.

Use the **delete** form of this command to remove the community string for version 2 of SNMP.

Use the **show** form of this command to display the community string for version 2 of SNMP.

service probes probe action virtualization guest action

Sets the action for a guest.

```
set service probes probe probe-name action virtualization guest guest-name
action { none | poweroff | reset }
```

```
delete service probes probe probe-name action virtualization guest guest-name
action { none | poweroff | reset }
```

```
show service probes probe probe-name action virtualization guest guest-name
action
```

None

probe-name

A name for a probe.

poweroff

Powers off the guest forcefully.

reset

Reboots the guest forcefully.

none

Takes no action.

Configuration mode.

```
service probes probe probe-name
action {
    virtualization {
        guest guest-name {
            action {none | poweroff | reset}
        }
    }
}
```

Use this command to set the action for a guest.

Use the **set** form of this command to set the action for a guest.

Use the **delete** form of this command to remove the action for a guest.

Use the **show** form of this command to display the action for a guest.

service probes probe action virtualization guest startup-wait

Sets the time to wait after starting a guest before probing.

```
set service probes probe probe-name action virtualization guest guest-name
startup-wait startup-wait-value
```

```
delete service probes probe probe-name action virtualization guest guest-name
startup-wait startup-wait-value
```

```
show service probes probe probe-name action virtualization guest guest-name
startup-wait
```

The default waiting time is 5 minutes.

probe-name

A name for a probe.

guest-name

The name of a guest.

startup-wait-value

The time, in minutes, to wait for the guest to boot and its services to become available. The time ranges from 1 to 30 minutes.

Configuration mode.

```

service probes probe probe-name action virtualization
guest guest-name {
    startup-wait startup-wait-value
}

```

Use this command to set the time to wait after starting a guest before probing.

Use the **set** form of this command to add the time to wait after starting a guest before probing.

Use the **delete** form of this command to set the default time to wait after guest start before probing

Use the **show** form of this command to display the time to wait after starting a guest before probing.

show probes status

Displays the status of the configured probes.

```
show probes status
```

Operational mode.

Use this command to display probe status.

The `show probes status` command displays the following information for probes:

Output field	Description
State	Status of the probe; for example: Running and Paused.
Current failures	Consecutive failure count.
Total failures	Total failures.
Next probe	Next probe date and time.
Last failure	Date and time of the last failure.
Last action	Date and time of the last action.
Last probe message	Last probe message.

The following example shows a probe that is running and has not failed since the probe service was last configured. The `Last probe message` field contains the latest value retrieved through SNMP because this probe is an SNMP probe.

```

tmpuser@node:~$ show probes status
snmp-probe:

```

```

State: Running
  Current failures: 0
  Total failures: 0

```

```
Next probe: 2015-11-12 15:10:24  
Last failure: None  
Last action: None  
Last probe message: STRING: Vyatta 4.0.0.R1.11022250
```


Chapter 15. Connecting Guests in a Service Chain

Service chaining overview

Service chaining is the linking of a logical group of service functions to a packet or flow to realize a service. Service functions in the chain can include network services, such as load balancer, firewall, intrusion prevention, routing, and so on. The criteria for applying a service chain to a packet or flow are based on the packet or flow attributes that span the OSI layer, such as the physical port, IP address header, transport, and application layer information.

The current release of VNF platform supports bridging, policy-based routing (PBR), a cross-connect, and a hybrid method (PBR and cross-connect) of service chaining.

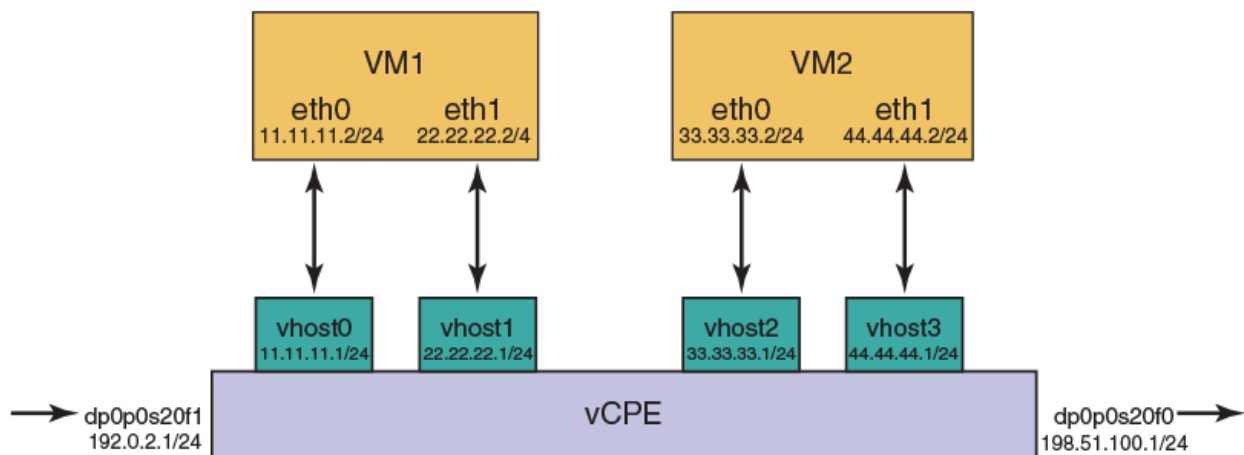
Configuring service chaining by using PBR

Consider a scenario in which you have two guests VM1 and VM2, as illustrated in the following network diagram. These guests can be any services such as vNAT, vFW, vDPI, and so on.

The following list presents the components of the topology.

- dp0p0s20f1 and dp0p0s20f0 are the ingress and egress interfaces, respectively, for VNF platform or the vhost.
- vhost0 and vhost1 are the virtual interfaces for VM1.
- vhost2 and vhost3 are the virtual interfaces for VM2.
- sc1-h1 and sc1-h2 are the PBR policies that are applied to dp0p0s20f1 and vhost1.
- eth0 and eth1 are the interfaces of VM1 and VM2.


Figure 2. Service chaining by using PBR



In the topology, to perform service chaining between VM1 and VM2, the PBR policy is applied on dp0p0s20f1 and vhost1, and the route policy lookup is performed on all packets that are received on the dp0p0s20f1 and vhost1 interfaces. If there is a match, the route lookup is performed on the corresponding PBR tables 1 and 2.

In each service route table, static routes are added with the next hop interface pointing to a service guest.

Table 17. Configuring service chaining

Step	Commands
Configure the PBR policy.	<pre>tmpuser@node# set policy route pbr sc1-h1 rule 1 action 'accept' tmpuser@node# set policy route pbr sc1-h1 rule 1 address-family 'ipv4' tmpuser@node# set policy route pbr sc1-h1 rule 1 table '1' tmpuser@node# set policy route pbr sc1-h2 rule 1 action 'accept' tmpuser@node# set policy route pbr sc1-h2 rule 1 address-family 'ipv4' tmpuser@node# set policy route pbr sc1-h2 rule 1 table '2'</pre>
Configure the static route.	<pre>tmpuser@node# set protocols static table 1 route 0.0.0.0/0 next-hop '11.11.11.2' tmpuser@node# set protocols static table 2 route 0.0.0.0/0 next-hop '33.33.33.2'</pre>
Attach the PBR policy to the corresponding interface.	<pre>tmpuser@node# set interfaces dataplane dp0p0s20f1 policy route pbr 'sc1-h1' tmpuser@node# set interfaces vhost dp0vhost1 policy route pbr 'sc1-h2'</pre>
Configure the IP addresses on the interfaces.	<pre>tmpuser@node# set interfaces dataplane dp0p0s20f1 address '192.0.2.1/24' tmpuser@node# set interfaces dataplane dp0p0s20f0 address '198.51.100.1/24' tmpuser@node# set interfaces vhost dp0vhost0 address '11.11.11.1/24' tmpuser@node# set interfaces vhost dp0vhost1 address '22.22.22.1/24' tmpuser@node# set interfaces vhost dp0vhost2 address '33.33.33.1/24' tmpuser@node# set interfaces vhost dp0vhost3 address '44.44.44.1/24'</pre> <p> Note: The dp0vhost0 interface corresponds to eth0 of the VM1 interface and both the VMs should be on the same subnet.</p>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>

Configuring service chaining by using bridging

Consider a scenario in which you have two guest VM1 and VM2, as illustrated in the following network diagram. These guest VMs can be any services such as vNAT, vFW, vDPI, and so on.

The following list presents the components of the topology.

- dp0p0s20f1 and dp0p0s2f0 are the ingress and egress interfaces, respectively, VNF platform or the vhost.
- vhost0 and vhost1 are the virtual interfaces for VM1.
- vhost2 and vhost3 are the virtual interfaces for VM2.
- dp0p0s20f1 and vhost0 are in the br0 bridge group.
- vhost1 and vhost2 are in the br1 bridge group.


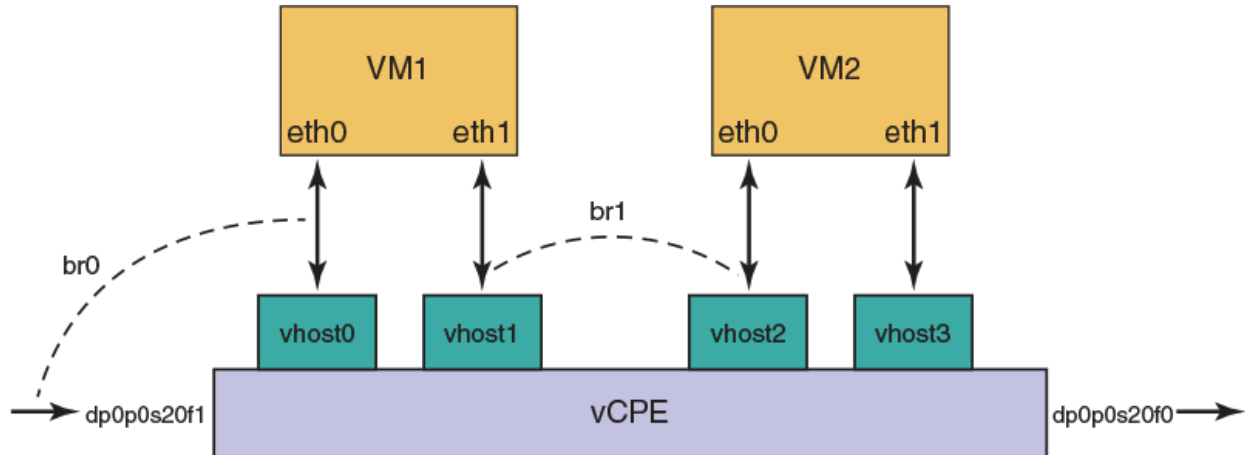
 **Note:** eth1(VM1) and eth0(VM2) should be in the same subnet because vhost1 and vhost2 are associated to the bridge.

Figure 3. Service chaining by using bridging



In the topology, to perform service chaining between VM1 and VM2, the br1 bridge group is applied between dp0p0s20f1 and vhost0, and br2 between vhost1 and vhost2. All packets that are received on the dp0p0s20f1 are forwarded to vhost0, and the packets that are received on vhost1 are forwarded to vhost2.

Table 18. Configuring service chaining by using bridging

Steps	Commands
Configure bridging.	<pre>tmpuser@node# set interfaces bridge br0 tmpuser@node# set interfaces dataplane dp0p0s20f1 bridge-group bridge br0 tmpuser@node# set interfaces vhost dp0vhost0 bridge-group bridge br0 tmpuser@node# set interfaces bridge br1 tmpuser@node# set interfaces vhost dp0vhost1 bridge-group bridge br1 tmpuser@node# set interfaces vhost dp0vhost2 bridge-group bridge br1</pre>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>

Configuring service chaining by using cross-connect

Consider a scenario in which you have two guest VM1 and VM2, as illustrated in the following network diagram. These guest can be any services such as vNAT, vFW, vDPI, and so on.

The following list presents the components of the topology.

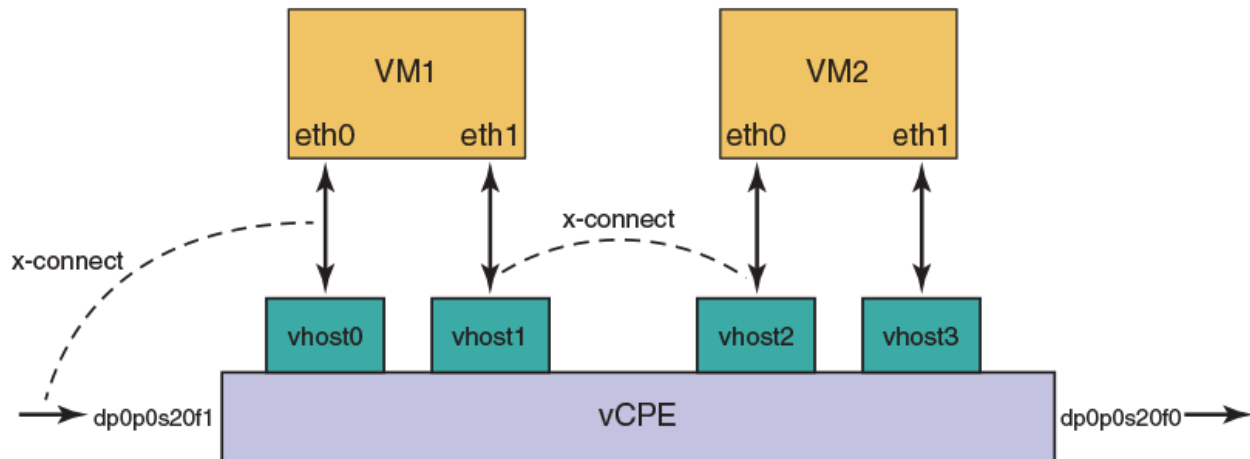
- dp0p0s20f1 and dp0p0s2f0 are the ingress and egress interfaces, respectively, for VNF platform or vhost.
- vhost0 and vhost1 are the virtual interfaces for VM1.

Note: eth1(vm1) and eth0(vm2) should be in the same subnet because vhost1 and vhost2 are associated to the bridge.

- vhost2 and vhost3 are the virtual interfaces for VM2.
- dp0vhost0 and dp0p0s20f2 are in cross-connect.

- dp0vhost1 and dp0vhost2 are in cross-connect.
- dp0vhost2 and dp0vhost1.

Figure 4. Service chaining by using cross-connect



In the topology, to perform service chaining between VM1 and VM2, the cross-connect is applied between dp0vhost0 and dp0p0s20f1, dp0vhost1 and vhost dp0vhost2, and dp0vhost2 and vhost dp0vhost1. All the packets that are received on dp0p0s20f1 are forwarded to dp0vhost0 and the traffic that is received on dp0vhost1 is forwarded to dp0vhost2.

Table 19. Configuring service chaining by using cross-connect

Steps	Commands
Configure cross-connect.	<pre>tmpuser@node# set interfaces vhost dp0p0s20f1 xconnect dataplane dp0vhost0 tmpuser@node# set interfaces vhost dp0vhost0 xconnect dataplane dp0p0s20f1 tmpuser@node# set interfaces vhost dp0vhost1 xconnect vhost dp0vhost2 tmpuser@node# set interfaces vhost dp0vhost2 xconnect vhost dp0vhost1</pre>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>

Configuring service chaining by using a hybrid model

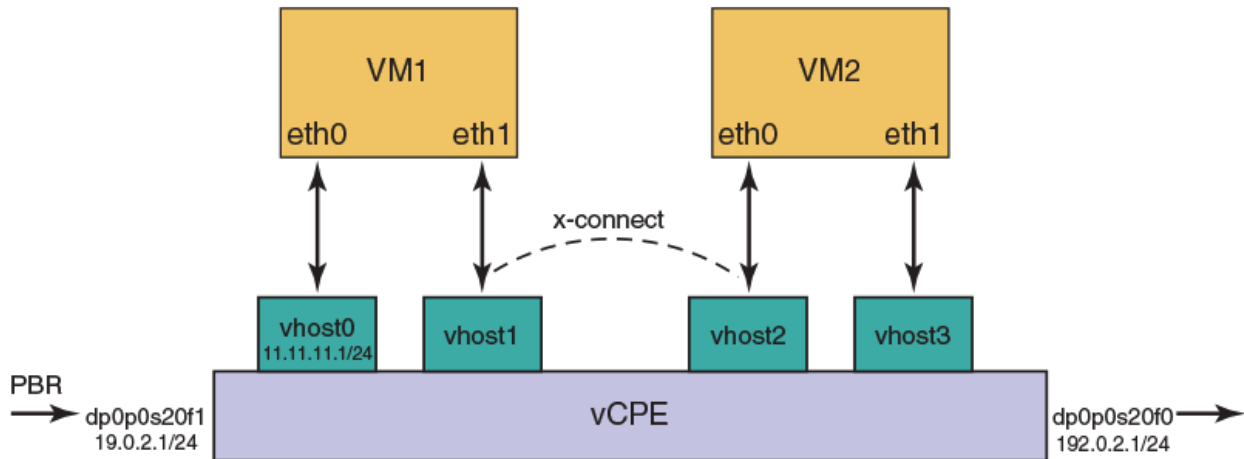
Consider a scenario in which you have two guest VM1 and VM2, as illustrated in the following network diagram. These guest can be any services such as vNAT, vFW, vDPI, and so on.

The following list presents the components of the topology.

- dp0p0s20f1 and dp0p0s2f0 are the ingress and egress interfaces, respectively, for VNF platform and the vhost.
- vhost0 and vhost1 are the virtual interfaces for VM1.
- vhost2 and vhost3 are the virtual interfaces for VM2.
- sc1-h1 is the PBR policies that are applied to dp0p0s20f1.

- dp0vhost1 and vhost dp0vhost2 are in cross-connect.

Figure 5. Service chaining by using a hybrid model



In the topology, to perform service chaining between VM1 and VM2, the PBR policy is applied on dp0p0s20f1, and the route policy lookup is performed on all packets that are received on the dp0p0s20f1 interfaces. If there is a match, the route lookup is performed on the corresponding PBR table 1. All packets that are received on dp0p0s20f2 are forwarded to dp0vhost0, and the traffic received on dp0vhost1 is forwarded to dp0vhost2.

Table 20. Configuring service chaining by using a hybrid model

Steps	Commands
Configure cross-connect.	<pre>tmpuser@node# set policy route pbr sc1-h1 rule 1 action 'accept' tmpuser@node# set policy route pbr sc1-h1 rule 1 address-family 'ipv4' tmpuser@node# set policy route pbr sc1-h1 rule 1 table '1' tmpuser@node# set interfaces vhost dp0vhost1 xconnect vhost dp0vhost2 tmpuser@node# set interfaces vhost dp0vhost2 xconnect vhost dp0vhost1</pre>
Configuring the static route	<pre>tmpuser@node# set protocols static table 1 route 0.0.0.0/0 next-hop '11.11.11.2'</pre>
Attach the PBR policy to the corresponding interface.	<pre>tmpuser@node# set interfaces dataplane dp0p0s20f1 policy route pbr 'sc1-h1'</pre>
Configure the IP addresses on the interfaces.	<pre>tmpuser@node# set interfaces dataplane dp0p0s20f1 address '192.0.2.1/24' tmpuser@node# set interfaces dataplane dp0p0s20f0 address '198.51.100.1/24' tmpuser@node# set interfaces vhost dp0vhost0 address '11.11.11.1/24'</pre>
Commit the configuration.	<pre>tmpuser@node# commit</pre>
Save the configuration.	<pre>tmpuser@node# save</pre>

Chapter 16. Service Chaining Commands

policy route pbr rule

Sets a policy-based route and PBR rule.

```
set policy route pbr group rule number
```

```
delete policy route pbr group rule number
```

```
show policy route pbr group
```

None.

group

A PBR group.

number

The number of a PBR rule, which ranges from 1 through 9999.

Configuration mode.

```
policy {
  route {
    pbr group {
      rule number
    }
  }
}
```

Use this command to set a policy-based route and PBR rule.

Use the **set** version of a command to set a policy-based route and PBR rule.

Use the **delete** version of a command to remove a policy-based route and PBR rule.

Use the **show** version of a command to display a policy-based route.

protocols static table routenext-hop

Sets a protocol static table.

```
set protocols static table table-number route route-address route-  
addressnext-hop address
```

```
delete protocols static table table-number route route-address route-  
addressnext-hop address
```

```
show protocols static table table-numberroute route-addressroute-address
```

None.

table-number

A policy route table number.

route-address

A route address.

next-hop address

A next-hop address address.

Configuration mode.

```
protocols static
  table table-number {
    route route-address {
      next-hop next-hop-address
    }
  }
}
```

Use this command to set a protocol static table.

Use the **set** version of the command to set a protocol static table.

Use the **delete** version of the command to remove a protocol static table.

Use the **show** version of the command to display a protocol static table.

interfaces bridge address

Sets the IP address of a configured bridge.

```
set interfaces bridge brN address ip-address
```

```
delete interfaces bridge brN address ip-address
```

```
show interfaces bridge brN
```

Configuration mode.

brN

A bridge interface name, where N is a number.

ip-address

The IPv4 or IPv6 address for the bridge. The formats of the IPv6 address and IPv4 address are *h:h:h:h:h:h:h/x* and *x.x.x.x/x* respectively.

Configuration mode.

```

interfaces {
    bridge brN {
        address ip-address
    }
}

```

Use this command to set the IP address of a configured bridge.

Use the **set** version of the command to set the IP address of a configured bridge.

Use the **delete** version of the command to remove the IP address of a configured bridge.

Use the **show** version of the command to display the IP address of a configured bridge.

interfaces vhost address

Sets the IP address of a configured vhost.

```
set interfaces vhost interface-name address ip-address
```

```
delete interfaces vhost interface-name address ip-address
```

```
show interfaces vhost interface-name
```

Configuration mode.

interface-name

A vhost interface name.

ip-address

An IPv4 or IPv6 address for the vhost. The formats of the IPv6 address and IPv4 address are *h:h:h:h:h:h/x* and *x.x.x/x* respectively.

Configuration mode.

```

interfaces {
    vhost interface-name {
        address ip-address
    }
}

```

Use this command to set the IP address of a configured vhost.

Use the **set** version of the command to set the IP address of a configured vhost.

Use the **delete** version of the command to remove the IP address of a configured vhost.

Use the **show** version of the command to display the IP address of a configured vhost.

interfaces dataplane address

Sets the IP address of a configured data plane.

```
set interfaces dataplane interface-name address ip-address
```

```
delete interfaces dataplane interface-name address ip-address
```

```
show interfaces dataplane interface-name
```

Configuration mode.

interface-name

A data plane interface name.

ip-address

The IPv4 or IPv6 address for the data plane. The formats of the IPv6 address and IPv4 address are *h:h:h:h:h:h:h/x* and *x.x.x.x/x* respectively.

Configuration mode.

```
interfaces {
  dataplane interface-name {
    address ip-address
  }
}
```

Use this command to set the IP address of a configured data plane.

Use the **set** version of the command to set the IP address of a configured data plane.

Use the **delete** version of the command to remove the IP address of a configured data plane.

Use the **show** version of the command to display the IP address of a configured data plane.

interfaces vhost bridge-group bridge

Sets the bridge group name on a vhost interface.

```
set interfaces vhost interface-name bridge-group bridge pattern
```

```
delete interfaces vhost interface-name bridge-group bridge pattern
```

```
set interfaces vhost interface-name bridge-group
```

Configuration Command.

interface-name

A vhost interface.

bridge-group-name

A bridge group name.

Configuration mode.

```
interfaces {
  vhost interface-name {
    bridge-group {
      bridge bridge-group-name
    }
  }
}
```

Use this command to set the bridge group name on a vhost interface.

Use the **set** version of the command to set the bridge group name on a vhost interface.

Use the **delete** version of the command to remove the bridge group name on a vhost interface.

Use the **show** version of the command to display the bridge group name on a vhost interface.

interfaces dataplane bridge-group bridge

Sets the bridge group name on a data plane interface.

```
set interfaces dataplane interface-name bridge-group bridge pattern
delete interfaces dataplane interface-name bridge-group bridge pattern
set interfaces dataplane interface-name bridge-group
```

Configuration mode.

interface-name

A vhost interface.

bridge-group-name

A bridge group name.

Configuration mode.

```
interfaces {
  dataplane interface-name {
    bridge-group {
      bridge brz
    }
  }
}
```

```

    }
  }
}

```

Use this command to set the bridge group name on a data plane interface.

Use the **set** version of the command to set the bridge group name on a data plane interface.

Use the **delete** version of the command to remove the bridge group name on a data plane interface.

Use the **show** version of the command to display the bridge group name on a data plane interface.

interfaces vhost vif vlan

Sets the associating VLAN to a virtual interface on a vhost interface.

```
set interfaces vhost interface-name vif vif-id vlan vlan-id
```

```
delete interfaces vhost interface-name vif vif-id vlan vlan-id
```

```
show interfaces vhost interface-name vif vif-id vlan
```

None.

interface-name

A vhost interface.

vif-id

A virtual interface ID which ranges from 1 through 99999.

vlan-id

A Virtual LAN ID which ranges from 1 through 4094.

Configuration mode.

```

interfaces {
  vhost interface-name {
    vif vif-id {
      vlan vlan-id
    }
  }
}

```

Use this command to set the associating VLAN to a vif on a vhost interface.

Use the **set** version of the command to set the associating VLAN to a vif on a vhost interface.

Use the **delete** version of the command to remove the associating VLAN to a vif from a vhost interface.

Use the **show** version of the command to display the associating VLAN to a vif on a vhost interface.

interfaces vhost vif inner-vlan

Sets the associating inner VLAN to a vif on a vhost interface.

```
set interfaces vhost interface-name vif vif-id inner-vlan inner-vlan id
delete interfaces vhost interface-name vif vif-id inner-vlan inner-vlan id
show interfaces vhost interface-name vif vif-id inner-vlan
```

None.

interface-name

A vhost interface.

vif-id

A virtual interface ID which ranges from 1 through 99999.

inner-vlan id

A Virtual LAN ID which ranges from 1 through 4094.

Configuration mode.

```
interfaces {
  vhost interface-name {
    vif vif-id {
      inner-vlan inner-vlan id
    }
  }
}
```

Use this command to set the associating inner VLAN to a vif on a vhost interface.

Use the **set** version of the command to set the associating inner VLAN to a vif on a vhost interface.

Use the **delete** version of the command to remove the associating inner VLAN to a vif from a vhost interface.

Use the **show** version of the command to display the associating inner VLAN to a vif on a vhost interface.

interfaces dataplane xconnect dataplane

Sets the cross-connect between two data plane interfaces.

```
set interfaces dataplane interface-name xconnect dataplane interface-name
```

```
delete interfaces dataplane interface-name xconnect dataplane interface-name
```

```
show interfaces dataplane
```

Configuration mode.

interface-name-dataplane

A data plane interface name. The interface names must have one of the following values:

- dpFoN
- dpFsN
- dpFpNsS
- dpFpNpS
- dpFoN.N
- dpFsN.N
- dpFpNsS.N
- dpFpNpS.N

Configuration mode.

```
interfaces {
  dataplane interface-name-dataplane {
    xconnect {
      dataplane interface-name-dataplane
    }
  }
}
```

Use this command to set the cross-connect between two data plane interfaces.

Use the **set** version of the command to set the cross-connect between two data plane interfaces.

Use the **delete** version of the command to remove the cross-connect between two data plane interfaces.

Use the **show** version of the command to display the cross-connect between two data plane interfaces.

interfaces dataplane cross-connect vhost

Sets the cross-connect between an ordered combination of a data plane and a vhost interface.

```

set interfaces dataplane interface-name xconnect vhost interface-name
delete interfaces dataplane interface-name xconnect vhost interface-name
show interfaces dataplane

```

Configuration mode.


interface-name-dataplane

A data plane interface name. The interface names must have one of the following values:

- dpFoN
- dpFsN
- dpFpNsS
- dpFpNpS
- dpFoN.N
- dpFsN.N
- dpFpNsS.N
- dpFpNpS.N

interface-name-vhost

A virtio vhost name.

 **Note:** The interface name vary across different hypervisors and releases

Configuration mode.

```

interfaces {
  dataplane interface-name-dataplane {
    xconnect {
      vhost interface-name-vhost
    }
  }
}

```

Use this command to set the cross-connect between an ordered combination of a data plane and a vhost interface.

Use the **set** version of the command to set the cross-connect between an ordered combination of a data plane and vhost interface.

Use the **delete** version of the command to remove the cross-connect between an ordered combination of a data plane and vhost interface.

Use the **show** version of the command to display the cross-connect between an ordered combination of a data plane and vhost interface.

interfaces vhost xconnect vhost


Sets the cross-connect between two vhost interfaces.

```
set interfaces vhost interface-name xconnect vhost interface-name
delete interfaces vhost interface-name xconnect vhost interface-name
show interfaces vhost
```

Configuration mode.

interface-name-dataplane

A virtio vhost name.

 **Note:** Interface names vary across different hypervisors and releases

Configuration mode.

```
interfaces {
  vhost interface-name-dataplane {
    xconnect {
      vhost interface-name-vhost
    }
  }
}
```

Use this command to set the cross-connect between two vhost interfaces.

Use the **set** version of the command to set the cross-connect between two vhost interfaces.

Use the **delete** version of the command to remove the cross-connect between two vhost interfaces.

Use the **show** version of the command to display the cross-connect between two vhost interfaces.

interfaces vhost xconnect dataplane

Sets the cross-connect between an ordered combination of a vhost and a data plane interface.


```
set interfaces vhost interface-name-vhost xconnect dataplane interface-name-dataplane
delete interfaces vhost interface-name-vhost xconnect dataplane interface-name-dataplane
```

```
show interfaces vhost
```

Configuration mode.

interface-name-vhost

A virtio vhost name.

 **Note:** Interface names vary across different hypervisors and releases.

interface-name-dataplane

A data plane interface name. The interface name must have one of the following values:

- dpFoN
- dpFsN
- dpFpNsS
- dpFpNpS
- dpFoN.N
- dpFsN.N
- dpFpNsS.N
- dpFpNpS.N

Configuration mode.

```
interfaces {
  vhost interface-name-vhost {
    xconnect {
      dataplane interface-name-dataplane
    }
  }
}
```

Use this command to set the cross-connect between an ordered combination of a data plane and a vhost interface.

Use the **set** version of the command to set the cross-connect between an ordered combination of a data plane and vhost interface.

Use the **delete** version of the command to remove the cross-connect between an ordered combination of a data plane and vhost interface.

Use the **show** version of the command to display the cross-connect between an ordered combination of a data plane and vhost interface.

show policy

Displays policy configuration.

```
show policy
```


Operational mode.

Use this command to display information about policy.

The following example shows how to display policy configuration.

```
tmpuser@node:~$ show policy
-----
Rulesets Information: PBR
-----
-----
PBR policy "SC1":
Active on (dp0p0s20f2, in)
rule      action  proto          packets      bytes
-----  -
10        allow  any            0             0
        condition - from 100.1.1.2 to 200.1.1.2 table 1

PBR policy "SC2":
Active on (dp0vhost1, in)
rule      action  proto          packets      bytes
-----  -
10        allow  any            0             0
        condition - from 100.1.1.2 to 200.1.1.2 table 2
```

show policy route

Displays routing policy configuration.

```
show policy route
```

Operational mode.

Use this command to display information about policy route.

The following example shows how to display routing policy configuration.

```
tmpuser@node:~$ show policy
-----
Rulesets Information: PBR
-----
-----
PBR policy "SC1":
Active on (dp0p0s20f2, in)
rule      action  proto          packets      bytes
-----  -
10        allow  any            0             0
        condition - from 100.1.1.2 to 200.1.1.2 table 1

PBR policy "SC2":
Active on (dp0vhost1, in)
```

```

rule      action  proto          packets      bytes
----      -
10       allow   any           0            0
        condition - from 100.1.1.2 to 200.1.1.2 table 2

```

show policy route table

Displays the routing policy configuration table.

```
show policy route table
```

Operational mode.

Use this command to display information about policy route table.

The following example shows how to display the routing policy configuration table.

```

tmpuser@node:~$ show policy route table
PBR Group          Rule  Table
-----
                   SC1   10    1
                   SC2   10    2

```

Chapter 17. Troubleshooting

Inability to reprovision a configured guest automatically

If you have already provisioned a guest and then re-enter the `add virtualization xml` command, you see an error message, which indicates the guest is already configured, similar to the following:

```
tmpuser@node:~$ add virtualization xml /home/vyatta/vr5600.xml image /home/vyatta/vr5600.qcow2 iso /home/vyatta/config.iso
RPC failure: [ERROR] - Guest "vr5600" has already been configured at /opt/vyatta/share/perl5/Vyatta/Configd.pm line 193.
tmpuser@node:~$
```

This operation is by design.

Disk source image error when reprovisioning a guest automatically

If you have already provisioned a guest and then enter the `delete virtualization guest` configuration mode command to delete the specified guest, you cannot deploy the guest again by using the `add virtualization xml` operational mode command. An error message similar to the following is displayed.

```
tmpuser@node:~$ add virtualization xml /home/vyatta/vr5600.xml image /home/vyatta/vr5600.qcow2 iso /home/vyatta/config.iso
RPC failure: [ERROR] - Disk source image "/var/lib/libvirt/images/vr5600/0-vr5600.qcow2" already exists at /opt/vyatta/share/perl5/Vyatta/Configd.pm line 193.
tmpuser@node:~$
```

The issue occurs because the `delete virtualization guest` command deletes the guest configuration only. The command does not delete any disk images that are used by the hypervisor in the `/var/lib/libvirt/images` folder. To resolve the issue, use the `force` option with the `add virtualization xml` command or delete the disk image when you delete the guest.

The `delete virtualization image` operational command is available to delete guest images from the `/var/lib/libvirt/images` folder.

Duplicate IP address error message when provisioning a guest automatically

The XML file defines the IP addresses for vhost interfaces. When you try to deploy a guest automatically by specifying an XML file with IP addresses that are already configured for the VNF platform, an error message similar to the following is displayed.

```
RPC failure: [INFO] - Configured vhost interface "dp0vhost1" [INFO] -
Configured guest "vr5600" [INFO] - Discarded all candidate deployment
configuration [ERROR] - [interfaces] Duplicate address 10.1.1.2 used
on interfaces: dp0vhost0,dp0vhost1 [[interfaces]] failed. Validate
failed! at /opt/vyatta/share/perl5/Vyatta/Configd.pm line 193.
```

To resolve the issue, ensure that the IP addresses used in the XML file are unique for the VNF platform.

Invalid XML file error when provisioning a guest automatically

If the deployment function cannot parse the XML file when provisioning a guest automatically, an error message similar to the following is displayed.

```
tmpuser@node:~$ add virtualization xml /home/vyatta/bad-xml.xml image /
home/vyatta/vr5600.qcow2
RPC failure: [ERROR] - error parsing attribute name, line 2, column 1
[ERROR] - Failed to parse libvirt XML at /opt/vyatta/share/perl5/Vyatta/
Configd.pm line 193.
tmpuser@node:~$
```

There are two parts to the error message:

- The first part of the error message identifies either the location (line and column number) of a general XML parsing error or the invalid XML element, attribute, or value for a specific XML parsing error. General XML parsing errors include wrong XML syntax, such as badly formed elements, missing opening or closing angle-brackets, and incorrect nesting of elements. For specific XML parsing errors, see [Specific XML parsing errors when provisioning a guest automatically](#).
- The second part of the error message indicates that the XML file cannot be parsed.

Specific XML parsing errors when provisioning a guest automatically

If the deployment function cannot parse the XML file when provisioning a guest automatically, an error message is displayed indicating either a general XML error or a specific XML error.

The following table lists specific XML parsing errors and their descriptions. When trying to resolve specific XML parsing errors, open the XML file in a text editor and search for the listed XML element or attribute name.

Legend for the table:

- Text that may vary on a per-error-message basis is shown as ??.
- The XML element names are shown with leading and trailing angle brackets, such as <domain>.
- The XML attribute values are enclosed within single quotation marks, such as 'type'.
- The severity levels of the error messages are: error, warning, and informational.
 - Errors cause the deployment to fail immediately.
 - Warnings cause the specified the element or attribute to be ignored, and the parsing of the XML file continues. Certain elements may be missing from the final configuration that causes the guest to behave in unexpected ways.
 - Informationals are for information purposes only and do not affect the deployment of the guest.

Message	Severity	Cause	Possible solution or corrected XML
<disk> missing 'device' attribute	Warning	The <disk> element must have a 'device' attribute.	<disk device='disk'> <disk device='cdrom'>
<domain> element missing 'type' attribute	Error	The <domain> element must have a 'type' attribute.	<domain type='kvm'>
<graphics> 'passwd' attribute missing, not adding device	Warning	The <graphics> element is missing the 'passwd' element.	<graphics passwd='my-secret-password'>
<graphics> 'port' and 'autoport' attributes both missing	Warning	The <graphics> element must have either a 'port' or an 'autoport' attribute.	<graphics port=????/> <graphics autoport='yes'/>
<graphics> 'tlsPort' is not supported, not adding device	Warning	The <graphics> 'tlsPort' attribute is not supported.	Remove any <graphics> 'tlsPort' attributes from the XML file.
<graphics> 'type' attribute missing	Warning	The <graphics> element is missing the 'type' attribute.	<graphics type='spice'/> <graphics type='vnc'/>
<video> <model> 'type' and 'vram' attributes both missing	Warning	The <video> element does not have a <model> 'type' or a <model> 'vram' attribute.	<video> <model type='cirrus'/> <video> <model type='qxl'/> <video> <model type='vga'/>
<watchdog> element missing 'model' attribute	Warning	The <watchdog> element is missing the 'model' attribute.	<watchdog model='i6300esb'/>
<watchdog> model '??' not supported, not adding hardware	Warning	The <watchdog> element's 'model' attribute value is unsupported; the only supported value is 'i6300esb'.	<watchdog model='i6300esb'/>
No <domain> element	Error	The outermost element of the XML file is not the <domain> element.	Add a <domain> element as the outermost element in the XML file.
No <name> element	Error	The <name> element is missing from the XML file.	Add <name>guest-name</name> to the XML file.

Message	Severity	Cause	Possible solution or corrected XML
Only one graphic device is supported, using the first device.	Warning	More than one <graphic> element is defined in the XML file.	Remove the extraneous <graphic> elements from the XML file.
Only one video device is supported, using the first device	Warning	More than one <video> <model> element is defined in the XML file.	Remove the extraneous <video> <model> elements from the XML file.
Only one watchdog device is supported, using the first device	Warning	More than one <watchdog> element is defined in the XML file.	Remove the extraneous <watchdog> elements from the XML file.
Unrecognised allocation unit: ??	Error	The <memory> 'unit' attribute value is not recognized.	See xxx for valid allocation unit identifiers.
unsupported <disk device='??'> attribute	Warning	The <disk> 'device' attribute value is not 'disk' or 'cdrom'.	<disk device='cdrom'> <disk device='disk'>
unsupported <graphics> 'type' attribute: '??'	Warning	The <graphics> 'type' attribute value is not 'spice' or 'vnc'.	<graphics type='spice' /> <graphics type='vnc' />
unsupported <interface> 'type' attribute: '??'	Warning	The <interface> 'type' attribute value is not 'ethernet' or 'bridge'.	<interface type='ethernet'> <interface type='bridge'>
Unsupported hypervisor: ??	Error	The only <domain> 'type' attribute value supported is 'kvm'.	<domain type='kvm'>
unsupported video model type: '??'	Warning	The <video> <model> 'type' attribute is not recognized.	<video> <model type='cirrus' /> <video> <model type='qxl' /> <video> <model type='vga' />
Watchdog action ?? is not supported, not adding hardware	Warning	The <watchdog> element's 'action' attribute value is unsupported.	<watchdog model='i6300esb' action='none' /> <watchdog model='i6300esb' action='poweroff' /> <watchdog model='i6300esb' action='reset' />

Paused probe state in the show probe status command

If a probe remains in the `Paused` state after it has been running for at least the startup-wait period.

Work-around:

Either of the following actions should recover the probe:

- Restart the probe daemon:
 - `$ sudo systemctl restart probed`
 - `$ sudo systemctl reload probed`
- Manually power off and start the associated guest.

After performing any of these steps, the probe will remain paused while waiting for a guest to start.

Fatal probe state in the show probe status command

A `Fatal` probe state indicates that the probe has experienced an internal failure. All other output for that probe is invalid and that the probe stops running.

This issue is not a routine occurrence; submit a bug if it occurs. If possible, attach a stack trace from `sudojournalctl` to the bug.

Show command failures

The following errors do not occur in normal usage:

```
tmpuser@node:~$ show probes status Could not parse "service probes status
probe" JSON: GetState failure: Probe daemon is not running!
tmpuser@node:~$ show probes status Could not parse "service probes status
probe" JSON: GetState failure: Could not communicate with probe daemon!
```

Work-around: If these errors occur, check for stack traces in `sudo journalctl` which would indicate that the probe daemon has crashed. The following commands may recover the probe daemon:

- `$ sudo systemctl restart probed`
- `$ sudo systemctl reload probed`