



# Cumulus Linux 3.0.1

## User Guide

---

# Table of Contents

<b>Welcome to Cumulus Networks .....</b>	<b>5</b>
<b>What's New in Cumulus Linux 3.0.1 .....</b>	<b>6</b>
Contents .....	6
New Behavior and Functionality .....	7
Not All Features Available on Mellanox Platforms .....	12
Early Access Features .....	14
Removed Features .....	14
<b>Quick Start Guide .....</b>	<b>16</b>
Contents .....	16
What's New in Cumulus Linux 3.0.1 .....	16
Open Source Contributions .....	16
Prerequisites .....	17
Hardware Compatibility List .....	17
Installing Cumulus Linux .....	17
Upgrading Cumulus Linux .....	18
Configuring Cumulus Linux .....	18
Configuring 4x10G Port Configuration (Splitter Cables) .....	21
Testing Cable Connectivity .....	21
Configuring Switch Ports .....	22
Configuring a Loopback Interface .....	24
<b>Installation, Upgrading and Package Management .....</b>	<b>26</b>
Managing Cumulus Linux Disk Images .....	26
Adding and Updating Packages .....	53
Zero Touch Provisioning - ZTP .....	59
<b>System Management .....</b>	<b>72</b>
Setting Date and Time .....	72
Authentication, Authorization, and Accounting .....	75
Netfilter - ACLs .....	95
Managing Application Daemons .....	122
Configuring switchd .....	126
Power over Ethernet - PoE .....	129
Configuring a Global Proxy .....	133
<b>Configuring and Managing Network Interfaces .....</b>	<b>134</b>
Contents .....	134
Commands .....	134
Man Pages .....	135

Configuration Files .....	135
Basic Commands .....	135
ifupdown2 Interface Classes .....	136
Configuring a Loopback Interface .....	138
ifupdown2 Interface Dependencies .....	139
Configuring IP Addresses .....	143
Specifying User Commands .....	145
Sourcing Interface File Snippets .....	145
Using Globs for Port Lists .....	146
Using Templates .....	146
Adding Descriptions to Interfaces .....	147
Caveats and Errata .....	148
Useful Links .....	148
Layer 1 and Switch Port Attributes .....	149
Buffer and Queue Management .....	161
Configuring DHCP Relays .....	167
<b>Layer 1 and Layer 2 Features .....</b>	<b>172</b>
Spanning Tree and Rapid Spanning Tree .....	172
Link Layer Discovery Protocol .....	191
Prescriptive Topology Manager - PTM .....	198
Bonding - Link Aggregation .....	211
Ethernet Bridging - VLANs .....	214
Multi-Chassis Link Aggregation - MLAG .....	244
LACP Bypass .....	261
Virtual Router Redundancy - VRR .....	264
Network Virtualization .....	269
IGMP and MLD Snooping .....	376
<b>Layer 3 Features .....</b>	<b>382</b>
Routing .....	382
Introduction to Routing Protocols .....	387
Network Topology .....	389
Quagga Overview .....	391
Configuring Quagga .....	393
Open Shortest Path First - OSPF - Protocol .....	406
Open Shortest Path First v3 - OSPFv3 - Protocol .....	417
Border Gateway Protocol - BGP .....	419
Bidirectional Forwarding Detection - BFD .....	446
Equal Cost Multipath Load Sharing - Hardware ECMP .....	451
Redistribute Neighbor .....	459
Virtual Routing and Forwarding - VRF .....	469
Management VRF .....	489
<b>Monitoring and Troubleshooting .....</b>	<b>497</b>

Contents .....	497
Commands .....	497
Using the Serial Console .....	497
Diagnostics Using cl-support .....	499
Sending Log Files to a syslog Server .....	501
Next Steps .....	503
Single User Mode - Boot Recovery .....	503
Resource Diagnostics Using cl-resource-query .....	506
Monitoring System Hardware .....	507
Monitoring Virtual Device Counters .....	512
Understanding and Decoding the cl-support Output File .....	517
Troubleshooting Network Interfaces .....	534
Network Troubleshooting .....	544
SNMP Monitoring .....	567
<b>Cumulus Networks Services Demos .....</b>	<b>593</b>
Contents .....	593
Reference Topology .....	593
<b>Docker on Cumulus Linux .....</b>	<b>596</b>
Contents .....	596
Installing Docker .....	596
Caveats when Using Docker with Cumulus Linux .....	596
<b>Index .....</b>	<b>599</b>

# Welcome to Cumulus Networks

We are transforming networking with Cumulus Linux, the industry's first, full-featured Linux operating system for networking hardware. Cumulus Linux is a complete network operating system, based on [Debian Jessie](#) and Linux kernel 4.1. Unlike traditional embedded platforms, Cumulus Linux provides a complete environment pre-installed with scripting languages, server utilities, and monitoring tools. Management tasks are accomplished via SSH using standard Linux commands or over a serial console connection.



This user guide covers Cumulus Linux 3.0.0 through 3.0.1.



This documentation is current as of August 4, 2016 for version 3.0.1. Please visit the [Cumulus Networks Web site](#) for the most up to date documentation.

- [Release Notes for Cumulus Linux 3.0.1](#)
- [What's New in Cumulus Linux 3.0.1 \(see page 6\)](#)
- [Quick Start Guide \(see page 16\)](#)
- [Installation, Upgrading and Package Management \(see page 26\)](#)
- [System Management \(see page 72\)](#)
- [Configuring and Managing Network Interfaces \(see page 134\)](#)
- [Layer 2 Features \(see page 172\)](#)
- [Layer 3 Features \(see page 382\)](#)
- [Monitoring and Troubleshooting \(see page 497\)](#)

# What's New in Cumulus Linux 3.0.1

Cumulus Linux 3.0.1 includes bug fixes only.

Cumulus Linux 3.0 has a host of new features and capabilities. In addition to this chapter, please read the [release notes](#) to learn about known issues with this release.

Cumulus Linux 3.0 includes these new features and platforms:

- [Debian Jessie](#) (upgraded from [Debian Wheezy](#))
- [4.1 kernel](#) (upgraded from 3.2)
- Debian's [`systemctl`](#) and [`systemd`](#) replace the [`service`](#) command for administering services; they also replace [`jdoe`](#) for monitoring
- New installer
- [Quagga reload](#) (see page 391)
- VRF: virtual routing and forwarding (see page 469)
- BGP add-path (see page 430) (TX and RX)
- Redistribute neighbor (see page 459)
- New ASICs and platforms
  - New switching silicon (Broadcom Tomahawk and Trident II+, Mellanox Spectrum)
  - 100G platforms: Dell Z9100, Mellanox SN2700
  - 10GT switches: Dell S4048T

Read on to learn about more new functionality and new behaviors.

## Contents

- [Contents \(see page 6\)](#)

- New Behavior and Functionality (see page 7)
  - Cumulus Linux Now Based on Jessie (see page 7)
  - Quagga Default Configuration Changes (see page 7)
  - PowerPC Switches Not Supported (see page 9)
  - Default snmpd Port Binding (see page 9)
  - iquerySecName and Rouser (see page 9)
  - New Bond Defaults (see page 10)
  - New bridge mdb Command Syntax (see page 10)
  - Adding Static Bridge FDB Entries (see page 10)
  - Printing VLAN Ranges for a Bridge (see page 11)
  - List of Ports for a VLAN No Longer Displayed (see page 11)
  - virtio-net Driver Changes (see page 11)
  - MLAG ad\_actor\_key Setting Change (see page 11)
  - New ARP Refresh Rate (see page 11)
  - switchd Doesn't Start if License Isn't Present (see page 11)
  - SSH to Switch as root User Disabled by Default (see page 11)
  - SSH Output No Longer Truncated (see page 12)
- Not All Features Available on Mellanox Platforms (see page 12)
  - Supported Cables for Mellanox Switches (see page 12)
- Early Access Features (see page 14)
- Removed Features (see page 14)

## New Behavior and Functionality

Cumulus Linux 3.0 marks a significant departure from earlier releases of the operating system. As such, some new functionality and behaviors are to be expected.

### ***Cumulus Linux Now Based on Jessie***

Cumulus Linux is now based on Debian Jessie, instead of Debian Wheezy. For a list of issues you need to be aware of, please read the [Debian documentation](#).

### ***Quagga Default Configuration Changes***

	Description	2.5.x Default Configuration	3.x Default Configuration
ospf log-adjacency-changes	Logs a single message when a peer transitions to/from FULL state		On

	<b>Description</b>	<b>2.5.x Default Configuration</b>	<b>3.x Default Configuration</b>
ospf spf timers	<p>OSPF uses three timers (A, B, C) as an exponential backoff, to prevent consecutive SPFs from hammering the CPU.</p> <ul style="list-style-type: none"> <li>• A: ms from initial event until SPF runs</li> <li>• B: ms between consecutive SPF runs (the number doubles with each SPF, until it reaches the value of C)</li> <li>• C: Maximum ms between SPFs</li> </ul>	<ul style="list-style-type: none"> <li>• A: 200</li> <li>• B: 1000</li> <li>• C: 10000</li> </ul>	<ul style="list-style-type: none"> <li>• A: 0</li> <li>• B: 50</li> <li>• C: 5000</li> </ul>
bgp log-neighbor-changes	Logs a single message when a peer transitions to/from Established state		On
bgp deterministic-med	Ensures path ordering no longer impacts bestpath selection		Enabled
bgp default show-hostname	Displays the hostname in show command output.		Enabled
bgp network import-check			Enabled
bgp keepalive timers		60s	3s
bgp hold timers		180s	9s
bgp timers-connect	Controls how long Cumulus Linux waits between attempts to bring up a peer	120s	10s

Additional configuration changes:

- BGP peer-groups restrictions have been replaced with update-groups, which dynamically examine all peers, and group them if they have the same outbound policy.
- BGP Min Route Advertisement Interval timers for eBGP and iBGP were set to 0 seconds, rather than 30 seconds for eBGP and 5 seconds for iBGP.
- IPv6 Route Advertisements are automatically enabled on an interface with IPv6 addresses, so the step `no ipv6 nd suppress-ra` is no longer needed for BGP unnumbered. The timer interval for RAs remains 600s, which may need to be adjusted to bring up peers quickly.
- A peer needs to be attached to a peer-group only once, when it then inherits all address-families activated for that peer-group.

- The default configuration for `bgp best path as-path multipath-relax` has been changed to `no-as-set`, as the Quagga implementation produced strange routing scenarios when allowed to create an AS\_SET in some situations. An as-set configuration option has been added.
- BGP multipath is enabled by default; the number of maximum paths defaults to 64.
- Simplified BGP unnumbered configuration — a single command can configure a neighbor and attach to peer-group:  
`neighbor <sfpX> interface peer-group <group name>`

## **PowerPC Switches Not Supported**

PowerPC switches are **not** supported under Cumulus Linux 3.0. They are supported under Cumulus Linux 2.5 Extended Service Release (ESR). To see if your switch uses a PowerPC processor, you can either:

- Check the [hardware compatibility list \(HCL\)](#).
- Run `uname -m` in the console on the switch. If it returns `ppc`, it's a PowerPC switch.

## **Default snmpd Port Binding**

In previous releases of Cumulus Linux, the default port binding configuration in `/etc/snmp/snmpd.conf` was:

```
# 2.5.x default agent IP address binding (bind to all interfaces on UDP
port 161)
agentAddress udp::161
```

This meant that the `snmpd` daemon listed and responded to all ports for UDP port 161.

In Cumulus Linux 3.0, the default configuration has been updated to a more secure setting:

```
# 3.x default agent IP address binding (bind to only loopback interface on
UDP port 161)
agentAddress udp:127.0.0.1:161
```

This ensures that by default, the `snmpd` daemon will only listen on the loopback interface on UDP port 161, and will only respond to SNMP requests originating on the switch itself, rather than requests coming into the box on an interface. Since this is really only useful for testing purposes, most customers should change this to binding to a specific IP address.

## **iquerySecName and Rouser**

In 2.5.x, default values for iquerySecName and rouser were configured in `/etc/snmp/snmpd.conf` as follows:

```
iquerySecName internalUser
rouser internalUser
```

In 3.x, the default configuration has been updated to a more secure setting, by commenting out the default user:

```
#iquerySecName internalUser
#rouser internalUser
```

User accounts must now be created manually for SNMP traps to function correctly.

## New Bond Defaults

In order to simplify configurations, many [bond settings](#) (see page 211) have had their defaults changed:

Setting	2.x Default	3.x Default
lacp-rate	none	1
miimon	0	100
min-links	0	1
mode	none	802.3ad
use-carrier	none	1
xmit-hash-policy	none	layer3+4

## New bridge mdb Command Syntax

The syntax of the `bridge mdb` command has changed slightly. Instead of using `vlan <vid>` to specify the VLAN ID of a multicast group on a VLAN-aware bridge, Cumulus Linux uses `vid <vid>`. Similarly, when dumping the MDB with the `bridge mdb show` command, the VLAN ID, if any, is displayed following the `vid` keyword.

## Adding Static Bridge FDB Entries

To add a static bridge FDB entry, make sure to specify `static` in the `bridge fdb` command. For example:

```
cumulus@switch:~$ sudo bridge fdb add 00:01:02:03:04:06 dev eth0 master  
static
```

## Printing VLAN Ranges for a Bridge

In order to print a range of VLANs in a bridge (see page 235), use the `-c` option with `bridge vlan show`:

```
cumulus@switch:~$ bridge -c vlan show
```

## List of Ports for a VLAN No Longer Displayed

The `bridge vlan show <vlanid>` command in the Linux 4.1 kernel no longer displays the list of ports for a VLAN, unlike in the 3.2 kernel, which did show list of ports for a VLAN.

In addition, the `/sys/class/net/<portname>/brport/pvid sysfs` node is no longer present in Cumulus Linux.

## ***virtio-net Driver Changes***

The default speed setting for the virtio-net driver is set to SPEED\_10.

In addition, VLAN Tx offload is enabled in the virtio-net driver by default.

## ***MLAG ad\_actor\_key Setting Change***

In Cumulus Linux 3.0, the `ad_actor_key` parameter for a 10G full-duplex port is set to 13; in Cumulus Linux 2.5.x, the `ad_actor_key` for the same 10G speed and full-duplex port was set to 33.

## ***New ARP Refresh Rate***

For ARP timers, the default `base_reachable_time_ms` in Cumulus Linux 3.0 and later is 14400000 (4 hours); in Cumulus Linux 2.5.x it is 110000 (110 seconds).

## ***switchd Doesn't Start if License Isn't Present***

If a license is not installed on a Cumulus Linux switch, the `switchd` service will not start. If you install the license again, start `switchd` with:

```
cumulus@switch:~$ sudo systemctl start switchd.service
```

## ***SSH to Switch as root User Disabled by Default***

To improve security, the ability to use SSH to connect to a switch as the root user using a password has been disabled by default. To enable it, read [User Accounts \(see page 77\)](#).

## ***SSH Output No Longer Truncated***

In Cumulus Linux 2.5.x, depending upon the number of peers on the network, the output of `show ip bgp summary json` over an SSH session might get truncated. This has been fixed in Cumulus Linux 3.0.

## **Not All Features Available on Mellanox Platforms**

A number of features are not available or are limited on [Mellanox switches](#) at this time. These include:

- ACLs
- CDP
- SPAN (however, ERSPAN is supported)
- VRF
- VXLAN
- Resilient hashing
- 64 MACs (breakout to 25G is limited)
- sFlow
- Specific cables supported

## ***Supported Cables for Mellanox Switches***

Cumulus Networks has tested and suggests using the following cables and transceivers with Mellanox switches at this time:

<b>Speed: 10G (QSA Adapter Used)</b>						
<b>Manufacturer</b>	<b>Label (or internal EEPROM)</b>	<b>Type</b>	<b>Form Factor</b>	<b>Supported Lengths</b>	<b>Supported Speeds</b>	<b>Known Issues?</b>
Mellanox	MFM1T02A-SR	SR	SFP	X	10G	
Mellanox	MFM1T02A-LR	LR	SFP	X	10G	
Mellanox	MC3309130	DAC	SFP	2M, 3M, 5M, 7M	10G	
<b>Speed: 40G</b>						
<b>Manufacturer</b>	<b>Label (or internal EEPROM)</b>	<b>Type</b>	<b>Form Factor</b>	<b>Supported Lengths</b>	<b>Supported Speeds</b>	<b>Known Issues?</b>
Finisar	FTL410QE2C	SR	QSFP	X	40G	

<b>Speed: 10G (QSA Adapter Used)</b>						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MC2210411-SR4	SR	QSFP	X	40G	
Mellanox	MC2210411-LR4	LR	QSFP	X	40G	
Mellanox	MC2210130	DAC	QSFP	1M, 3M, 5M	40G	
Mellanox	MC2210310	AoC	QSFP	10M	40G	
Ampehnol	APF14190032M3A	DAC	QSFP	3M	40G	
JDSU	JQP—04SWAA1	SR	QSFP	X	40G	
<b>Speed: 100G</b>						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MCP1600	DAC	QSFP	2M, 3M	40G/100G	
Mellanox	MMA1B00	SR	QSFP	X	40G/100G	
TE Connectivity	2231368-1	DAC	QSFP	1M, 3M	40G/100G	
Ampehnol	NDAAFF	DAC	QSFP	1M	40G/100G	
<b>Speed: 40G to 4x10G</b>						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MC2609130	DAC	QSFP to 4xSFP	1M	40G to 4x10G	
Ampehnol	NDAQGF-0002 (internal)	DAC	QSFP to 4xSFP	1M, 3M	40G to 4x10G	
<b>Speed: 100G to 4x25G</b>						

Speed: 10G (QSA Adapter Used)						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MCP7F00-A02A	DAC	QSFP to 4xSFP	3M	100G to 4x25G	
10Gtek	CAB-ZQP/4ZSP-P1M	DAC	QSFP to 4xSFP	1M	100G to 4x25G	
Speed: 40G to 1x10G						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MC2309130	DAC	QSFP to 1xSFP	3M, 5M	40G to 1x10G	
Speed: 100G to 2x50G						
Manufacturer	Label (or internal EEPROM)	Type	Form Factor	Supported Lengths	Supported Speeds	Known Issues?
Mellanox	MCP7H00	DAC	QSFP to 2xQSFP	1M, 3M, 5M	100G to 2x50G	Yes (with ConnectX4)

## Early Access Features

The following [early access features](#) are included in Cumulus Linux 3.0:

- Quagga automation optimization (applies configuration modifications only)

## Removed Features

- `cl-img-install`. The [installer](#) (see page 31) has been replaced.
- Disk image slots and `/mnt/persist`: For information and strategies on how to preserve your network configuration across software upgrades, read the [Upgrading Cumulus Linux](#) (see page 41) chapter.
- `cl-ns-mgmt`: This experimental feature was introduced in Cumulus Linux 2.1.1 to help users separate their management network from the in-band network. You should use [management VRF](#) (see page 489) instead.

- The following [LACP bypass \(see page 261\)](#) settings are no longer supported: priority mode, bond-lacp-bypass-period, bond-lap-bypass-priority and bond-lap-bypass-all-active .
- The `clag_enable` and `ad_sys_mac_addr` [bonding \(see page 211\)](#) parameters.
- `cl-brctl`. This utility was simply a symlink to `brctl`, which is what you should use to [configure bridges \(see page 214\)](#), VLANs and the like.
- `jdoe`. Use `systemd` and `systemctl` for [monitoring \(see page 497\)](#) your switches.

# Quick Start Guide

This chapter helps you get up and running with Cumulus Linux quickly and easily.

## Contents

(Click to expand)

- [Contents \(see page 16\)](#)
- [What's New in Cumulus Linux 3.0.1 \(see page 16\)](#)
- [Open Source Contributions \(see page 16\)](#)
- [Prerequisites \(see page 17\)](#)
- [Hardware Compatibility List \(see page 17\)](#)
- [Installing Cumulus Linux \(see page 17\)](#)
- [Upgrading Cumulus Linux \(see page 18\)](#)
- [Configuring Cumulus Linux \(see page 18\)](#)
  - [Login Credentials \(see page 19\)](#)
  - [Serial Console Management \(see page 19\)](#)
  - [Wired Ethernet Management \(see page 19\)](#)
  - [Configuring the Hostname and Time Zone \(see page 19\)](#)
  - [Installing the License \(see page 20\)](#)
- [Configuring 4x10G Port Configuration \(Splitter Cables\) \(see page 21\)](#)
- [Testing Cable Connectivity \(see page 21\)](#)
- [Configuring Switch Ports \(see page 22\)](#)
  - [Layer 2 Port Configuration \(see page 22\)](#)
  - [Layer 3 Port Configuration \(see page 23\)](#)
- [Configuring a Loopback Interface \(see page 24\)](#)

## What's New in Cumulus Linux 3.0.1

Cumulus Linux 3.0.1 contains bug fixes only. The [release notes](#) contain information about the release as well as the fixed and known issues.

## Open Source Contributions

Cumulus Networks has forked various software projects, like CFEngine, [Netdev](#) and some Puppet Labs packages in order to implement various Cumulus Linux features. The forked code resides in the Cumulus Networks [GitHub repository](#).

Cumulus Networks developed and released as open source some new applications as well.

The list of open source projects is on the [open source software](#) page.

## Prerequisites

Prior intermediate Linux knowledge is assumed for this guide. You should be familiar with basic text editing, Unix file permissions, and process monitoring. A variety of text editors are pre-installed, including `vi` and `nano`.

You must have access to a Linux or UNIX shell. If you are running Windows, you should use a Linux environment like [Cygwin](#) as your command line tool for interacting with Cumulus Linux.



If you're a networking engineer but are unfamiliar with Linux concepts, use [this reference guide](#) to see examples of the Cumulus Linux CLI and configuration options, and their equivalent Cisco Nexus 3000 NX-OS commands and settings for comparison. You can also [watch a series of short videos](#) introducing you to Linux in general and some Cumulus Linux-specific concepts in particular.

## Hardware Compatibility List

You can find the most up to date hardware compatibility list (HCL) [here](#). Use the HCL to confirm that your switch model is supported by Cumulus Networks. The HCL is updated regularly, listing products by port configuration, manufacturer, and SKU part number.

## Installing Cumulus Linux

This quick start guide walks you through the steps necessary for getting Cumulus Linux up and running on your switch, which includes:

1. Powering on the switch and entering ONIE, the Open Network Install Environment.
2. Installing Cumulus Linux on the switch via ONIE.
3. Booting into Cumulus Linux and installing the license.
4. Configuring switch ports and a loopback interface.

To install Cumulus Linux, you use [ONIE](#) (Open Network Install Environment), an extension to the traditional U-Boot software that allows for automatic discovery of a network installer image. This facilitates the ecosystem model of procuring switches, with a user's own choice of operating system loaded, such as Cumulus Linux.



If Cumulus Linux 3.0.0 or later is already installed on your switch, and you need to upgrade the software only, you can skip to [Upgrading Cumulus Linux](#) (see page 18) below.

The easiest way to install Cumulus Linux with ONIE is via local HTTP discovery:

1. If your host (like a laptop or server) is IPv6-enabled, make sure it is running a Web server.  
If the host is IPv4-enabled, make sure it is running DHCP as well as a Web server.

2. [Download](#) the Cumulus Linux installation file to the root directory of the Web server. Rename this file `onie-installer`.
3. Connect your host via Ethernet cable to the management Ethernet port of the switch.
4. Power on the switch. The switch downloads the ONIE image installer and boots it. You can watch the progress of the install in your terminal. After the installation finishes, the Cumulus Linux login prompt appears in the terminal window.



These steps describe a flexible unattended installation method. You should not need a console cable. A fresh install via ONIE using a local Web server should generally complete in less than 10 minutes.

You have more options for installing Cumulus Linux with ONIE. Read [Installing a New Cumulus Linux Image](#) (see page 31) to install Cumulus Linux using ONIE in the following ways:

- DHCP/Web server with and without DHCP options
- Web server without DHCP
- FTP or TFTP without a Web server
- Local file
- USB

ONIE supports many other discovery mechanisms using USB (copy the installer to the root of the drive), DHCPv6 and DHCPv4, and image copy methods including HTTP, FTP, and TFTP. For more information on these discovery methods, refer to the [ONIE documentation](#).

After installing Cumulus Linux, you are ready to:

- Log in to Cumulus Linux on the switch.
- Install the Cumulus Linux license.
- Configure Cumulus Linux. This quick start guide provides instructions on configuring switch ports and a loopback interface.

## Upgrading Cumulus Linux

To install Cumulus Linux 3.0.0 or later and you're running a version earlier than 3.0.0, you must perform a complete install, as [described above](#) (see page 17). If you already have Cumulus Linux 3.0.0 or later installed on your switch, read [Upgrading Cumulus Linux](#) (see page 26) for considerations before start the process.

## Configuring Cumulus Linux

When bringing up Cumulus Linux for the first time, the management port makes a DHCPv4 request. To determine the IP address of the switch, you can cross reference the MAC address of the switch with your DHCP server. The MAC address should be located on the side of the switch or on the box in which the unit was shipped.

## Login Credentials

The default installation includes one system account, `root`, with full system privileges, and one user account, `cumulus`, with `sudo` privileges. The `root` account password is set to null by default (which prohibits login), while the `cumulus` account is configured with this default password:

```
CumulusLinux!
```

In this quick start guide, you will use the `cumulus` account to configure Cumulus Linux.



For best security, you should change the default password (using the `passwd` command) before you configure Cumulus Linux on the switch.

All accounts except `root` are permitted remote SSH login; `sudo` may be used to grant a non-root account root-level access. Commands which change the system configuration require this elevated level of access.

For more information about sudo, read [Using sudo to Delegate Privileges \(see page 79\)](#).

## Serial Console Management

Users are encouraged to perform management and configuration over the network, [either in band or out of band \(see page 44\)](#). Use of the serial console is fully supported; however, many customers prefer the convenience of network-based management.

Typically, switches will ship from the manufacturer with a mating DB9 serial cable. Switches with ONIE are always set to a 115200 baud rate.

## Wired Ethernet Management

Switches supported in Cumulus Linux always contain at least one dedicated Ethernet management port, which is named `eth0`. This interface is geared specifically for out-of-band management use. The management interface uses DHCPv4 for addressing by default. You can set a static IP address in the `/etc/network/interfaces` file:

```
auto eth0
iface eth0
    address 192.0.2.42/24
    gateway 192.0.2.1
```

## Configuring the Hostname and Time Zone

To change the hostname, modify the `/etc/hostname` and `/etc/hosts` files with the desired hostname and reboot the switch. First, edit `/etc/hostname`:

```
cumulus@switch:~$ sudo vi /etc/hostname
```

Then replace the 127.0.1.1 IP address in `/etc/hosts` with the new hostname:

```
cumulus@switch:~$ sudo vi /etc/hosts
```

Reboot the switch:

```
cumulus@switch:~$ sudo reboot
```

To update the time zone, update the `/etc/timezone` file with the [correct timezone](#), run `dpkg-reconfigure --frontend noninteractive tzdata`, then reboot the switch:

```
cumulus@switch:~$ sudo vi /etc/timezone
cumulus@switch:~$ sudo dpkg-reconfigure --frontend noninteractive tzdata
cumulus@switch:~$ sudo reboot
```



It is possible to change the hostname without a reboot via a script available on [Cumulus Networks GitHub site](#).

## Installing the License

Cumulus Linux is licensed on a per-instance basis. Each network system is fully operational, enabling any capability to be utilized on the switch with the exception of forwarding on switch panel ports. Only eth0 and console ports are activated on an unlicensed instance of Cumulus Linux. Enabling front panel ports requires a license.

You should have received a license key from Cumulus Networks or an authorized reseller. Here is a sample license key:

```
user@company.com|thequickbrownfoxjumpsoverthelazydog312
```

There are three ways to install the license onto the switch:

- Copy it from a local server. Create a text file with the license and copy it to a server accessible from the switch. On the switch, use the following command to transfer the file directly on the switch, then install the license file:

```
cumulus@switch:~$ scp user@my_server:/home/user/my_license_file.txt .
cumulus@switch:~$ sudo cl-license -i my_license_file.txt
```

- Copy the file to an HTTP server (not HTTPS), then reference the URL when you run `cl-license`:

```
cumulus@switch:~$ sudo cl-license -i <URL>
```

- Copy and paste the license key into the `cl-license` command:

```
cumulus@switch:~$ sudo cl-license -i
<paste license key>
^d
```



You no longer have to reboot the switch to activate the switch ports. Once you install the license, all front panel ports will be active and will show up as `swp1`, `swp2`, and so forth.

## Configuring 4x10G Port Configuration (Splitter Cables)

If you are using 4x10G DAC or AOC cables, edit the `/etc/cumulus/ports.conf` to enable support for these cables then [restart the `switchd` service \(see page 128\)](#) using the `sudo systemctl restart switchd.service` command. For more details, see [Layer 1 and Switch Port Attributes \(see page 149\)](#).

## Testing Cable Connectivity

By default, all data plane ports (every Ethernet port except the management interface, `eth0`) are disabled. To test cable connectivity, administratively enable a port using `ip link set <interface> up`:

```
cumulus@switch:~$ sudo ip link set swp1 up
```

Run the following bash script, as root, to administratively enable all physical ports:

```
cumulus@switch:~$ sudo su -
cumulus@switch:~$ for i in /sys/class/net/*; do iface=`basename $i`; if [[
$iface == swp* ]]; then ip link set $iface up; fi done
```

To view link status, use `ip link show`. The following examples show the output of a port in "admin down", "down" and "up" mode, respectively:

```
# Administratively Down
swp1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN mode
DEFAULT qlen 1000

# Administratively Up but Layer 2 protocol is Down
swp1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state
DOWN mode DEFAULT qlen 500

# Administratively Up, Layer 2 protocol is Up
swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT qlen 500
```

## Configuring Switch Ports

### ***Layer 2 Port Configuration***

Cumulus Linux does not put all ports into a bridge by default. To configure a front panel port or create a bridge, edit the `/etc/network/interfaces` file. After saving the file, to activate the change, use the `ifup` command.

### ***Examples***

In the following configuration example, the front panel port swp1 is placed into a bridge called br0:

```
auto br0
iface br0
  bridge-ports swp1
  bridge-stp on
```

To put a range of ports into a bridge, use the `glob` keyword. For example, add swp1 through swp10, swp12, and swp14 through swp20 to br0:

```
auto br0
iface br0
    bridge-ports glob swp1-10 swp12 glob swp14-20
    bridge-stp on
```

To activate or apply the configuration to the kernel:

```
# First, check for typos:
cumulus@switch:~$ sudo ifquery -a

# Then activate the change if no errors are found:
cumulus@switch:~$ sudo ifup -a
```

To view the changes in the kernel, use the `brctl` command:

```
cumulus@switch:~$ brctl show
bridge name      bridge id          STP enabled     interfaces
br0              8000.089e01cedcc2   yes            swp1
```



A script is available to generate a configuration that [places all physical ports in a single bridge](#).

## **Layer 3 Port Configuration**

To configure a front panel port or bridge interface as a Layer 3 port, edit the `/etc/network/interfaces` file.

In the following configuration example, the front panel port `swp1` is configured a Layer 3 access port:

```
auto swp1
iface swp1
    address 10.1.1.1/30
```

To add an IP address to a bridge interface, include the address under the `iface` configuration in `/etc/network/interfaces`:

```
auto br0
iface br0
```

```
address 10.2.2.1/24
bridge-ports glob swp1-10 swp12 glob swp14-20
bridge-stp on
```

To activate or apply the configuration to the kernel:

```
# First check for typos:
cumulus@switch:~$ sudo ifquery -a

# Then activate the change if no errors are found:
cumulus@switch:~$ sudo ifup -a
```

To view the changes in the kernel use the `ip addr show` command:

```
br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
link/ether 00:02:00:00:00:28 brd ff:ff:ff:ff:ff:ff
inet 10.2.2.1/24 scope global br0

swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
link/ether 44:38:39:00:6e:fe brd ff:ff:ff:ff:ff:ff
inet 10.1.1.1/30 scope global swp1
```

## Configuring a Loopback Interface

Cumulus Linux has a loopback preconfigured in `/etc/network/interfaces`. When the switch boots up, it has a loopback interface, called `lo`, which is up and assigned an IP address of 127.0.0.1.



The loopback interface `lo` must always be specified in `/etc/network/interfaces` and must always be up.

To see the status of the loopback interface (`lo`), use the `ip addr show lo` command:

```
cumulus@switch:~$ ip addr show lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
```

Note that the loopback is up and is assigned an IP address of 127.0.0.1.

To add an IP address to a loopback interface, add it directly under the `iface lo inet loopback` definition in `/etc/network/interfaces`:

```
auto lo
iface lo inet loopback
    address 10.1.1.1
```



If an IP address is configured without a mask, as shown above, the IP address becomes a /32. So, in the above case, 10.1.1.1 is actually 10.1.1.1/32.

Multiple loopback addresses can be configured by adding additional `address` lines:

```
auto lo
iface lo inet loopback
    address 10.1.1.1
    address 172.16.2.1/24
```

# Installation, Upgrading and Package Management

A Cumulus Linux switch can have only one image of the operating system installed. This section discusses installing new and updating existing Cumulus Linux disk images, and configuring those images with additional applications (via packages) if desired.

Zero touch provisioning is a way to quickly deploy and configure new switches in a large-scale environment.

## Managing Cumulus Linux Disk Images

The Cumulus Linux operating system resides on a switch as a *disk image*. This section discusses how to manage the image, including installation and upgrading.

### Contents

(Click to expand)

- [Contents \(see page 26\)](#)
- [Commands \(see page 26\)](#)
- [Installing a New Cumulus Linux Image \(see page 26\)](#)
- [Upgrading Cumulus Linux \(see page 26\)](#)
- [x86 vs ARM Switches \(see page 27\)](#)
- [Reprovisioning the System \(Restart Installer\) \(see page 27\)](#)
- [Uninstalling All Images and Removing the Configuration \(see page 28\)](#)
- [Booting into Rescue Mode \(see page 28\)](#)
- [Inspecting Image File Contents \(see page 29\)](#)
- [Useful Links \(see page 31\)](#)

### Commands

- [apt-get](#)
- [onie-select](#)

### Installing a New Cumulus Linux Image

For details, read the chapter, [Installing a New Cumulus Linux Image \(see page 31\)](#).

### Upgrading Cumulus Linux

There are two ways you can upgrade Cumulus Linux:

- Upgrading only the changed packages, using `apt-get update` and `apt-get upgrade`. **This is the preferred method.**

- Perform a binary (full image) install of the new version, using **ONIE**. This is used when moving between major versions or if you want to install a clean image.

The entire upgrade process is described in [Upgrading Cumulus Linux](#) (see page 41).

## x86 vs ARM Switches

You can easily determine whether your switch is on an x86 or ARM platform by using the `uname -m` command.

For example, on an x86 platform, `uname -m` outputs `x86_64`:

```
cumulus@x86switch$ uname -m
x86_64
```

While on an ARM platform, `uname -m` outputs `armv7l`:

```
cumulus@ARMswitch$ uname -m
armv7l
```

You can also visit the HCL ([hardware compatibility list](#)) to look at your hardware to determine the processor type.

## Reprovisioning the System (Restart Installer)

You can reprovision the system, wiping out the contents of the whole switch.

To initiate the provisioning and installation process, use `onie-select -i`:

```
cumulus@switch:~$ sudo onie-select -i
WARNING:
WARNING: Operating System install requested.
WARNING: This will wipe out all system data.
WARNING:
Are you sure (y/N)? y
Enabling install at next reboot...done.
Reboot required to take effect.
```



A reboot is required for the reinstall to begin.



If you change your mind, you can cancel a pending reinstall operation by using `onie-select -c`:

```
cumulus@switch:~$ sudo onie-select -c  
Cancelling pending install at next reboot...done.
```

## ***Uninstalling All Images and Removing the Configuration***

To remove all installed images and configurations, returning the switch to its factory defaults, use `onie-select -k`:

```
cumulus@switch:~$ sudo onie-select -k  
WARNING:  
WARNING: Operating System uninstall requested.  
WARNING: This will wipe out all system data.  
WARNING:  
Are you sure (y/N)? y  
Enabling uninstall at next reboot...done.  
Reboot required to take effect.
```



A reboot is required for the uninstall to begin.



If you change your mind you can cancel a pending uninstall operation by using `onie-select -c`:

```
cumulus@switch:~$ sudo onie-select -c  
Cancelling pending uninstall at next reboot...done.
```

## ***Booting into Rescue Mode***

If your system becomes broken in some way, you may be able to correct things by booting into ONIE rescue mode. In rescue mode, the file systems are unmounted and you can use various Cumulus Linux utilities to try and fix the problem.

To reboot the system into the ONIE rescue mode, use `onie-select -r`:

```
cumulus@switch:~$ sudo onie-select -r  
WARNING:  
WARNING: Rescue boot requested.
```

**WARNING:**

```
Are you sure (y/N)? y
Enabling rescue at next reboot...done.
Reboot required to take effect.
```



A reboot is required to boot into rescue mode.



If you change your mind you can cancel a pending rescue boot operation by using `onie-select -c`:

```
cumulus@switch:~$ sudo onie-select -c
 Cancelling pending rescue at next reboot...done.
```

## Inspecting Image File Contents

The Cumulus Linux installation disk image file is executable. From a running switch, you can display the contents of the Cumulus Linux image file by passing the `info` option to the image file. For example, if the image file is called `onie-installer` and is located in `/var/lib/cumulus/installer`, you can get information about the disk image by running:

```
cumulus@switch:~$ sudo /var/lib/cumulus/installer/onie-installer info
Verifying image checksum ... OK.
Preparing image archive ... OK.
Control File Contents
=====
Description: Cumulus Linux
OS-Release: 2.1.0-0556262-201406101128-NB
Architecture: amd64
Date: Tue, 10 Jun 2014 11:44:28 -0700
Installer-Version: 1.2
Platforms: im_n29xx_t40n mlx_sx1400_i73612 dell_s6000_s1220
Homepage: http://www.cumulusnetworks.com/

Data Archive Contents
=====
 128 2014-06-10 18:44:26 file.list
    44 2014-06-10 18:44:27 file.list.sha1
 104276331 2014-06-10 18:44:27 sysroot-internal.tar.gz
    44 2014-06-10 18:44:27 sysroot-internal.tar.gz.sha1
 5391348 2014-06-10 18:44:26 vmlinuz-initrd.tar.xz
    44 2014-06-10 18:44:27 vmlinuz-initrd.tar.xz.sha1
cumulus@switch:~$
```

You can also extract the contents of the image file by passing the `extract` option to the image file:

```
cumulus@switch:~$ sudo /var/lib/cumulus/installer/onie-installer
extract PATH
Verifying image checksum ... OK.
Preparing image archive ... OK.
file.list
file.list.sha1
sysroot-internal.tar.gz
sysroot-internal.tar.gz.sha1
vmlinuz-initrd.tar.xz
vmlinuz-initrd.tar.xz.sha1
Success: Image files extracted OK.
cumulus@switch:~$ sudo ls -l
total 107120
-rw-r--r-- 1 1063 3000          128 Jun 10 18:44 file.list
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 file.list.sha1
-rw-r--r-- 1 1063 3000 104276331 Jun 10 18:44 sysroot-internal.tar.gz
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 sysroot-internal.tar.gz.
sha1
-rw-r--r-- 1 1063 3000   5391348 Jun 10 18:44 vmlinuz-initrd.tar.xz
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 vmlinuz-initrd.tar.xz.
sha1
```

Finally, you can verify the contents of the image file by passing the `verify` option to the image file:

```
cumulus@switch:~$ sudo /var/lib/cumulus/installer/onie-installer
verify
Verifying image checksum ... OK.
Preparing image archive ... OK.
file.list
file.list.sha1
sysroot-internal.tar.gz
sysroot-internal.tar.gz.sha1
vmlinuz-initrd.tar.xz
vmlinuz-initrd.tar.xz.sha1
Success: Image files extracted OK.
cumulus@switch:~$ sudo ls -l
total 107120
-rw-r--r-- 1 1063 3000          128 Jun 10 18:44 file.list
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 file.list.sha1
-rw-r--r-- 1 1063 3000 104276331 Jun 10 18:44 sysroot-internal.tar.gz
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 sysroot-internal.tar.gz.
sha1
-rw-r--r-- 1 1063 3000   5391348 Jun 10 18:44 vmlinuz-initrd.tar.xz
-rw-r--r-- 1 1063 3000           44 Jun 10 18:44 vmlinuz-initrd.tar.xz.
sha1
```

## Useful Links

- Open Network Install Environment (ONIE) Home Page

## Installing a New Cumulus Linux Image

Before you install Cumulus Linux, the switch can be in two different states:

- The switch has no image on it (so the switch is only running [ONIE](#)) or you desire or require a clean installation. In this case, you can install Cumulus Linux in one of the following ways, using:
  - DHCP/a Web server with DHCP options (see page 32)
  - DHCP/a Web server without DHCP options (see page 32)
  - A Web server with no DHCP (see page 33)
  - FTP or TFTP without a Web server (see page 33)
  - Local file installation (see page 34)
  - USB (see page 34)
- The switch already has Cumulus Linux installed on it, so you only need to [upgrade it](#) (see page 41)



ONIE is an open source project, equivalent to PXE on servers, that enables the installation of network operating systems (NOS) on bare metal switches.

## Understanding these Examples

The sections in this chapter are ordered from the most repeatable to the least repeatable methods. For instance, DHCP can scale to hundreds of switch installs with zero manual input, compared to something like USB installs. Installing via USB is fine for a single switch here and there but is not scalable.

- You can name your Cumulus Linux installer binary using any of the [ONIE naming schemes mentioned here](#).
- In the examples below, [PLATFORM] can be any supported Cumulus Linux platform, such as `x86_64`, or `arm`.

## Contents

Click to expand...

- Understanding these Examples (see page 31)
- Contents (see page 31)
- Installing via a DHCP/Web Server Method with DHCP Options (see page 32)
- Installing via a DHCP/Web Server Method without DHCP Options (see page 32)
- Installing via a Web Server with no DHCP (see page 33)
- Installing via FTP or TFTP without a Web Server (see page 33)
- Installing via a Local File (see page 34)
- Installing via USB (see page 34)
  - Preparing for USB Installation (see page 34)
  - Instructions for x86 Platforms (see page 36)

- Instructions for ARM Platforms (see page 38)
- Installing a New Image when Cumulus Linux Is already Installed (see page 41)

## **Installing via a DHCP/Web Server Method with DHCP Options**

Installing Cumulus Linux in this manner is as simple as setting up a DHCP/Web server on your laptop and connecting the eth0 management port of the switch to your laptop.

Once you connect the cable, the installation proceeds as follows:

1. The bare metal switch boots up and asks for an address (DHCP request).
2. The DHCP server acknowledges and responds with DHCP option 114 and the location of the installation image.
3. ONIE downloads the Cumulus Linux binary, installs and reboots.
4. Success! You are now running Cumulus Linux.



The most common method is for you to send DHCP option 114 with the entire URL to the Web server (this could be the same system). However, there are many other ways to use DHCP even if you don't have full control over DHCP. See the [ONIE user guide](#) for help.

Here's an example DHCP configuration with an [ISC DHCP server](#):

```
subnet 172.0.24.0 netmask 255.255.255.0 {
    range 172.0.24.20 172.0.24.200;
    option www-server = "http://172.0.24.14/onie-installer-[PLATFORM]";
}
```

Here's an example DHCP configuration with [dnsmasq](#) (static address assignment):

```
dhcp-host=sw4,192.168.100.14,6c:64:1a:00:03:ba,set:sw4
dhcp-option>tag:sw4,114,"http://roz.rtplab.test/onie-installer-[PLATFORM]"
```

Don't have a Web server? There is a [free Apache example](#) you can utilize.

## **Installing via a DHCP/Web Server Method without DHCP Options**

If you have a laptop on same network and the switch can pull DHCP from the corporate network, but you cannot modify DHCP options (maybe it's controlled by another team), do the following:

1. Place the Cumulus Linux binary in a directory on the Web server.
2. Run the `onie-nos-install` command manually, since DHCP options can't be modified:

```
ONIE:/ #onie-nos-install http://10.0.1.251/path/to/cumulus-install-[PLATFORM].bin
```

## ***Installing via a Web Server with no DHCP***

Use the following method if your laptop is on the same network as the switch eth0 interface but no DHCP server is available.

One thing to note is ONIE is in *discovery mode*, so if you are setting a static IPv4 address for the eth0 management port, you need to disable discovery mode or else ONIE may get confused.

1. To disable discovery mode, run:

```
onie# onie-discovery-stop
```

or, on older ONIE versions if that command isn't supported:

```
onie# /etc/init.d/discover.sh stop
```

2. Assign a static address to eth0 via ONIE (using `ip addr add`):

```
ONIE:/ #ip addr add 10.0.1.252/24 dev eth0
```

3. Place the Cumulus Linux installer image in a directory on your Web server.
4. Run the `onie-nos-install` command manually since there are no DHCP options:

```
ONIE:/ #onie-nos-install http://10.0.1.251/path/to/cumulus-install-[PLATFORM].bin
```

## ***Installing via FTP or TFTP without a Web Server***

1. Set up DHCP or static addressing for eth0, as in the examples above.
2. If you are utilizing static addressing, disable ONIE discovery mode.
3. Place the Cumulus Linux installer image into a TFTP or FTP directory.
4. If you are not utilizing DHCP options, run one of the following commands (`tftp` for TFTP or `ftp` for FTP):

```
ONIE# onie-nos-install ftp://local-ftp-server/cumulus-install-[PLATFORM].bin
```

```
ONIE# onie-nos-install tftp://local-tftp-server/cumulus-install-[PLATFORM].bin
```

## Installing via a Local File

1. Set up DHCP or static addressing for eth0, as in the examples above.
2. If you are utilizing static addressing, disable ONIE discovery mode.
3. Use [scp](#) to copy the Cumulus Linux binary to the switch.  
Note: Windows users can use [WinScp](#).
4. Run the following command:

```
ONIE# onie-nos-install /path/to/local/file/cumulus-install-[PLATFORM].bin
```

## Installing via USB

Following the steps below produces a clean installation of Cumulus Linux. This wipes out all pre-existing configuration files that may be present on the switch. Instructions are offered for x86 and ARM platforms, and also cover the installation of a license after the software installation.



Make sure to [back up \(see page 41\)](#) any important configuration files that you may need to restore the configuration of your switch after the installation finishes.

## Preparing for USB Installation

1. Download the appropriate Cumulus Linux image for your x86 or ARM platform from the [Cumulus Networks Downloads page](#).
2. Prepare your flash drive by formatting in one of the supported formats: FAT32, vFAT or EXT2.

Optional: Preparing a USB Drive inside Cumulus Linux



It is possible that you could severely damage your system with the following utilities, so please use caution when performing the actions below!

- a. Insert your flash drive into the USB port on the switch running Cumulus Linux and log in to the switch.

- b. Determine and note at which device your flash drive can be found by using output from `cat /proc/partitions` and `sudo fdisk -l [device]`. For example, `sudo fdisk -l /dev/sdb`.



These instructions assume your USB drive is the `/dev/sdb` device, which is typical if the USB stick was inserted after the machine was already booted. However, if the USB stick was plugged in during the boot process, it is possible the device could be `/dev/sda`. Make sure to modify the commands below to use the proper device for your USB drive!

- c. Create a new partition table on the device:

```
sudo parted /dev/sdb mklabel msdos
```



The `parted` utility should already be installed. However, if it is not, install it with:  
`sudo apt-get install parted`

- d. Create a new partition on the device:

```
sudo parted /dev/sdb -a optimal mkpart primary 0% 100%
```

- e. Format the partition to your filesystem of choice using ONE of the examples below:

```
sudo mkfs.ext2 /dev/sdb1
sudo mkfs.msdos -F 32 /dev/sdb1
sudo mkfs.vfat /dev/sdb1
```



To use `mkfs.msdos` or `mkfs.vfat`, you need to install the `dosfstools` package from the [Debian software repositories](#) (step 3 here shows you how to add repositories from Debian), as they are not included by default.

- f. To continue installing Cumulus Linux, mount the USB drive in order to move files to it.

```
sudo mkdir /mnt/usb
sudo mount /dev/sdb1 /mnt/usb
```

3. Copy the image and license files over to the flash drive and rename the image file to:

- `onie-installer-x86_64`, if installing on an x86 platform

- `onie-installer-arm`, if installing on an ARM platform



You can also use any of the [ONIE naming schemes mentioned here](#).



When using a Mac or Windows computer to rename the installation file the file extension may still be present. Make sure to remove the file extension otherwise ONIE will not be able to detect the file!

4. Insert the USB stick into the switch, then continue with the appropriate instructions below for your x86 or ARM platform.

## ***Instructions for x86 Platforms***

Click to expand x86 instructions...

1. Prepare the switch for installation:

- If the switch is offline, connect to the console and power on the switch.
- If the switch is already online in Cumulus Linux, connect to the console and reboot the switch into the ONIE environment with the `sudo onie-select -i` command, followed by `sudo reboot`. Then skip to step 4 below.
- If the switch is already online in ONIE, use the `reboot` command.



SSH sessions to the switch get dropped after this step. To complete the remaining instructions, connect to the console of the switch. Cumulus Linux switches display their boot process to the console, so you need to monitor the console specifically to complete the next step.

2. Monitor the console and select the ONIE option from the first GRUB screen shown below.



3. Cumulus Linux on x86 uses GRUB chainloading to present a second GRUB menu specific to the ONIE partition. No action is necessary in this menu to select the default option *ONIE: Install OS*.



4. At this point, the USB drive should be automatically recognized and mounted. The image file should be located and automatic installation of Cumulus Linux should begin. Here is some sample output:

```
ONIE: OS Install Mode ...
Version : quanta_common_rangeley-2014.05.05-6919d98-201410171013
Build Date: 2014-10-17T10:13+0800
Info: Mounting kernel filesystems... done.
Info: Mounting LABEL=ONIE-BOOT on /mnt/onie-boot ...
initializing eth0...
scsi 6:0:0:0: Direct-Access SanDisk Cruzer Facet 1.26 PQ: 0
ANSI: 6
sd 6:0:0:0: [sdb] 31266816 512-byte logical blocks: (16.0 GB/14.9
GiB)
sd 6:0:0:0: [sdb] Write Protect is off
sd 6:0:0:0: [sdb] Write cache: disabled, read cache: enabled,
doesn't support DPO or FUA
sd 6:0:0:0: [sdb] Attached SCSI disk
<...snip...
ONIE: Executing installer: file://dev/sdb1/onie-installer-x86_64
Verifying image checksum ... OK.
Preparing image archive ... OK.
Dumping image info...
Control File Contents
=====
Description: Cumulus Linux
OS-Release: 3.0.0-3b46bef-201509041633-build
Architecture: amd64
Date: Fri, 27 May 2016 17:10:30 -0700
Installer-Version: 1.2
Platforms: accton_as5712_54x accton_as6712_32x
mlx_sx1400_i73612 dell_s6000_s1220 dell_s4000_c2338
```

```
dell_s3000_c2338 cel_redstone_xp cel_smallstone_xp cel_pebble
quanta_panther quanta_ly8_rangeley quanta_ly6_rangeley
quanta_ly9_rangeley
Homepage: http://www.cumulusnetworks.com/
```

5. After installation completes, the switch automatically reboots into the newly installed instance of Cumulus Linux.
6. Determine and note at which device your flash drive can be found by using output from `cat /proc/partitions` and `sudo fdisk -l [device]`. For example, `sudo fdisk -l /dev/sdb`.



These instructions assume your USB drive is the `/dev/sdb` device, which is typical if the USB stick was inserted after the machine was already booted. However, if the USB stick was plugged in during the boot process, it is possible the device could be `/dev/sda`. Make sure to modify the commands below to use the proper device for your USB drive!

7. Create a mount point to mount the USB drive to:

```
sudo mkdir /mnt/mountpoint
```

8. Mount the USB drive to the newly created mount point:

```
sudo mount /dev/sdb1 /mnt/mountpoint
```

9. Install your license file with the `cl-license` command:

```
sudo cl-license -i /mnt/mountpoint/license.txt
```

10. Check that your license is installed with the `cl-license` command.

11. Reboot the switch to utilize the new license.

```
sudo reboot
```

## Instructions for ARM Platforms

Click to expand ARM instructions...

1. Prepare the switch for installation:
  - If the switch is offline, connect to the console and power on the switch.
  - If the switch is already online in Cumulus Linux, connect to the console and reboot the switch into the ONIE environment with the `sudo onie-select -i` command, followed by `sudo reboot`. Then skip to step 4 below.

- If the switch is already online in ONIE, use the `reboot` command.



SSH sessions to the switch get dropped after this step. To complete the remaining instructions, connect to the console of the switch. Cumulus Linux switches display their boot process to the console, so you need to monitor the console specifically to complete the next step.

2. Interrupt the normal boot process before the countdown (shown below) completes. Press any key to stop the autobooting.

```

U-Boot 2013.01-00016-gd6bf4a9-dirty (Feb 14 2014 - 16:30:46)
Accton: 1.4.0.5
CPU0: P2020, Version: 2.1, (0x80e20021)
Core: E500, Version: 5.1, (0x80211051)
Clock Configuration:
  CPU0:1200 MHz, CPU1:1200 MHz,
  CCB:600 MHz,
  DDR:400 MHz (800 MT/s data rate) (Asynchronous), LBC:37.500 MHz
  L1: D-cache 32 kB enabled
    I-cache 32 kB enabled
<...snip...
USB: USB2513 hub OK
Hit any key to stop autoboot: 0

```

3. A command prompt appears, so you can run commands. Execute the following command:

```
run onie_bootcmd
```

4. At this point the USB drive should be automatically recognized and mounted. The image file should be located and automatic installation of Cumulus Linux should begin. Here is some sample output:

```

Loading Open Network Install Environment ...
Platform: arm-as4610_54p-r0
Version : 1.6.1.3
WARNING: adjusting available memory to 30000000
## Booting kernel from Legacy Image at ec040000 ...
  Image Name:  as6701_32x.1.6.1.3
  Image Type:  ARM Linux Multi-File Image (gzip compressed)
  Data Size:   4456555 Bytes = 4.3 MiB
  Load Address: 00000000
  Entry Point: 00000000
  Contents:
    Image 0: 3738543 Bytes = 3.6 MiB
    Image 1: 706440 Bytes = 689.9 KiB
    Image 2: 11555 Bytes = 11.3 KiB
Verifying Checksum ... OK

```

```

## Loading init Ramdisk from multi component Legacy Image at
ec040000 ...
## Flattened Device Tree from multi component Image at EC040000
  Booting using the fdt at 0xec47d388
  Uncompressing Multi-File Image ... OK
  Loading Ramdisk to 2ff53000, end 2ffff788 ... OK
  Loading Device Tree to 03ffa000, end 03ffffd22 ... OK
<...snip...
ONIE: Starting ONIE Service Discovery
ONIE: Executing installer: file://dev/sdb1/onie-installer-arm
Verifying image checksum ... OK.
Preparing image archive ... OK.
Dumping image info...
Control File Contents
=====
Description: Cumulus Linux
OS-Release: 3.0.0-3b46bef-201509041633-build
Architecture: arm
Date: Fri, 27 May 2016 17:08:35 -0700
Installer-Version: 1.2
Platforms: accton_as4600_54t, accton_as6701_32x, accton_5652,
accton_as5610_52x, dni_6448, dni_7448, dni_c7448n, cel_kennisis,
cel_redstone, cel_smallstone, cumulus_p2020, quanta_lb9,
quanta_ly2, quanta_ly2r, quanta_ly6_p2020
Homepage: http://www.cumulusnetworks.com/

```

5. After installation completes, the switch automatically reboots into the newly installed instance of Cumulus Linux.
6. Determine and note at which device your flash drive can be found by using output from `cat /proc/partitions` and `sudo fdisk -l [device]`. For example, `sudo fdisk -l /dev/sdb`.



These instructions assume your USB drive is the `/dev/sdb` device, which is typical if the USB stick was inserted after the machine was already booted. However, if the USB stick was plugged in during the boot process, it is possible the device could be `/dev/sda`. Make sure to modify the commands below to use the proper device for your USB drive!

7. Create a mount point to mount the USB drive to:

```
sudo mkdir /mnt/mountpoint
```

8. Mount the USB drive to the newly created mount point:

```
sudo mount /dev/sdb1 /mnt/mountpoint
```

9. Install your license file with the `cl-license` command:

```
sudo cl-license -i /mnt/mountpoint/license.txt
```

10. Check that your license is installed with the `cl-license` command.
11. Reboot the switch to utilize the new license.

```
sudo reboot
```

## ***Installing a New Image when Cumulus Linux Is already Installed***

Follow these upgrade steps for both major and minor releases, where:

- A major release upgrade is 2.X.X to 3.X.X (e.g. 1.5.1 to 2.5.0)
- A minor release upgrade is X.2.X to X.3.X (e.g. 2.2.0 to 2.5.5)

For more information, see [Upgrading Cumulus Linux \(see page 50\)](#).

## ***Upgrading Cumulus Linux***

Cumulus Networks software melds the Linux host world with the networking devices world. Each world comes with its own paradigm on how to upgrade software. Before we discuss the various ways to upgrade Cumulus Linux switches, let's review the general considerations and strategies used to upgrade network devices and Linux hosts.

## **Contents**

Click to expand...

- [Contents \(see page 41\)](#)
- [Upgrades: Comparing the Network Device Worldview vs. the Linux Host Worldview \(see page 42\)](#)
  - [Manual vs. Automated Configuration \(see page 42\)](#)
  - [Pre-deployment Testing of Production Environments \(see page 42\)](#)
  - [Locations of Configuration Data vs. Executables \(see page 42\)](#)
  - [Upgrade Procedure \(see page 43\)](#)
  - [Rollback Procedure \(see page 43\)](#)
  - [Third Party Packages \(see page 43\)](#)
- [Upgrading Cumulus Linux Devices: Strategies and Processes \(see page 44\)](#)
  - [Automated Configuration Is Preferred over Manual Configuration \(see page 44\)](#)
  - [Out-of-Band Management Is Worth the Investment \(see page 44\)](#)
  - [Pre-Deployment Testing of New Releases Is Advised and Enabled \(see page 44\)](#)
  - [Understanding the Locations of Configuration Data is Required for Successful Upgrades, Migration, and Backup \(see page 44\)](#)
  - [Upgrading Switches in an MLAG Pair \(see page 48\)](#)
- [Upgrading Cumulus Linux: Choosing between a Binary Install vs. Package Upgrade \(see page 48\)](#)
  - [Upgrading Using Package Installs \(`apt-get update && apt-get upgrade`\) \(see page 48\)](#)

- Upgrading via Binary Install (ONIE) (see page 50)
- Rolling Back a Cumulus Linux Installation (see page 50)
- Third Party Package Considerations (see page 51)
- Installation and Upgrade Workflow in Cumulus Linux 3.0 (see page 51)
- Caveats When Migrating Configuration Files Between Cumulus Linux 2.5.z and 3.0 (see page 51)
- Using the Config File Migration Script to Identify and Move Files to Cumulus Linux 3.0 (see page 52)
- Using Automation Tools to Back Up 2.5.z Configurations (see page 53)

## ***Upgrades: Comparing the Network Device Worldview vs. the Linux Host Worldview***

### ***Manual vs. Automated Configuration***

Historically, *network devices* were configured in place, and most network devices required customized configurations, which led predominantly to configuring the hardware manually. A lack of standardization between vendors, device types, and device roles hampered the development of APIs and automation tools. However, in the case of very large data centers, configurations became uniform and repeatable, and therefore scriptable. Some larger enterprises had to develop their own custom scripts to roll out data center network configurations. Virtually no industry-standard provisioning tools existed.

In contrast to data center network devices, *Linux hosts* in the data center number in the thousands and tend to have similar configurations. This increased scale led Linux sysadmins long ago to move to common tools to automate installation and configuration, since manually installing and configuring hosts did not work at the scale of a data center. Nearly all tasks are done via commonly available provisioning and orchestration tools.

### ***Pre-deployment Testing of Production Environments***

Historically, the cost of *network device* testing has been hampered by the cost of a single device. Setting up an appropriately sized lab topology can be very expensive. As a result, it is difficult to do comprehensive topology-based testing of a release before deploying it. Thus, many network admins cannot or will not do comprehensive system testing of a new release before deploying it.

Alternatively, the cost of a *Linux host* is cheap (or nearly free when using virtualization), so rigorous testing of a release before deploying it is not encumbered by budgeting concerns. Most sysadmins extensively test new releases in the complete application environment.

### ***Locations of Configuration Data vs. Executables***

*Network devices* generally separate configuration data from the executable code. On bootup, the executable code looks into a different file system and retrieves the configuration file or files, parses the text and uses that data to configure the software options for each software subsystem. The model is very centralized, with the executables generally being packaged together, and the configuration data following a set of rules that can be read by a centralized parser. Each vendor controls the configuration format for the entire box, since each vendor generally supports only their own software. This made sense since the platform was designed as an application-specific appliance.

Since a *Linux host* is a general purpose platform, with applications running on top of it, the locations of the files are much more distributed. Applications install and read their configuration data from text files usually stored in the /etc directory tree. Executables are generally stored in one of several *bin* directories, but the bin and etc directories are often on the same physical device. Since each *module* (application or executable) was often developed by a different organization and shared with the world, each module was responsible

for its own configuration data format. Most applications are community supported, and while there are some generally accepted guiding principles on how their configuration data is formatted, no central authority exists to control or ensure compliance.

## Upgrade Procedure

Both network admins and sysadmins generally plan upgrades only to gain new functionality or to get bug fixes when the workarounds become too onerous. The goal is to reduce the number of upgrades as much as possible.

The *network device* upgrade paradigm is to leave the configuration data in place, and *replace the executable files* either all at once from a single binary image or in large chunks (subsystems). A full release upgrade comes with risk due to unexpected behavior changes in subsystems where the admin did not anticipate or need changes.

The *Linux host* upgrade paradigm is to independently *upgrade a small list of packages* while leaving most of the OS untouched. Changing a small list of packages reduces the risk of unintended consequences. Usually upgrades are a "forward only" paradigm, where the sysadmins generally plan to move to the latest code within the same major release when needed. Every few years, when a new kernel train is released, a major upgrade is planned. A major upgrade involves wiping and replacing the entire OS and migrating configuration data.

## Rollback Procedure

Even the most well planned and tested upgrades can result in unforeseen problems, and sometimes the best solution to new problems is to roll back to the previous state.

Since *network devices* clearly separate data and executables, generally the process is to *overwrite the new release executable* with the previously running executable. If the configuration was changed by the newer release, then you either have to manually back out or repair the changes, or restore from an already backed up configuration.

The *Linux host* scenario can be more complicated. There are three main approaches:

- Back out individual packages: If the problematic package is identified, the sysadmin can downgrade the affected package directly. In rare cases the configuration files may have to be restored from backup, or edited to back out any changes that were automatically made by the upgrade package.
- Flatten and rebuild: If the OS becomes unusable, you can use orchestration tools to reinstall the previous OS release from scratch and then automatically rebuild the configuration.
- Backup and restore: Another common strategy is to restore to a previous state via a backup captured before the upgrade.

## Third Party Packages

Third party packages are rare in the *network device* world. Because the network OS is usually proprietary, third party packages are usually packaged by the network device vendor and upgrades of those packages is handled by the network device upgrade system.

Third party packages in *Linux host* world often use the same package system as the distribution into which it is to be installed (for example, Debian uses `apt-get`). Or the package may be compiled and installed by the sysadmin. Configuration and executable files generally follow the same filesystem hierarchy standards as other applications.

## **Upgrading Cumulus Linux Devices: Strategies and Processes**

Because Cumulus Linux is both Linux *and* a network device, it has characteristics of both paradigms. The following describes the Cumulus Linux paradigm with respect to upgrade planning and execution.

### **Automated Configuration Is Preferred over Manual Configuration**

Because Cumulus Linux *is* Linux, Cumulus Networks recommends that even with small networks or test labs, network admins should make the jump to deploy, provision, configure, and upgrade switches using automation from the very beginning. The small up front investment of time spent learning an orchestration tool, even to provision a small number of Cumulus Linux devices, will pay back dividends for a long time. The biggest gain is realized during the upgrade process, where the network admin can quickly upgrade dozens of devices in a repeatable manner.

Switches, like servers, should be treated like *cattle, not pets*.

### **Out-of-Band Management Is Worth the Investment**

Because network devices are reachable via the IP addresses on the front panel ports, many network admins of small-to-medium sized networks use *in-band* networks to manage their switches. In this design, management traffic like SSH, SNMP, and console server connections use the same networks that regular network traffic traverses — there is no separation between the *management plane* and the *data plane*. Larger data centers create a separate *out-of-band* network with a completely separate subnet and reachability path to attach to the management ports — that is accessible via eth0 and the serial console.

This is a situation where smaller companies should learn from the big companies. A separate management network isn't free, but it is relatively cheap. With an inexpensive [Cumulus RMP](#) management switch, an inexpensive console server, and a separate cable path, up to 48 devices can be completely controlled via the out-of-band network in the case of a network emergency.

There are many scenarios where in-band networking can fail and leave the network admin waiting for someone to drive to the data center or remote site to connect directly to the console of a misconfigured or failing device. The cost of one outage would usually more than pay for the investment in a separate network. For even more security, attach remote-controllable power distribution units (PDUs) in each rack to the management network, so you can have complete control to remote power cycle every device in that rack.

### **Pre-Deployment Testing of New Releases Is Advised and Enabled**

White box switches and virtualization (Cumulus VX) bring the cost of networking devices down, so the ability for network admins to test their own procedures, configurations, applications, and network topology in an appropriately-sized lab topology becomes extremely affordable.

### **Understanding the Locations of Configuration Data is Required for Successful Upgrades, Migration, and Backup**

As with other Linux distributions, the `/etc` directory is the primary location for all configuration data in Cumulus Linux. The following list is a likely set of files that should be backed up and migrated to a new release, but any file that has been changed would need to be examined. Cumulus Networks recommends you consider making the following files and directories part of a backup strategy.

## Network Configuration Files

File Name and Location	Explanation	Cumulus Linux Documentation	Debian Documentation
/etc/network/	Network configuration files, most notably /etc/network/interfaces and /etc/network/interfaces.d/	Configuring and Managing Network Interfaces (see page 134)	<a href="http://wiki.debian.org/NetworkConfiguration">wiki.debian.org/NetworkConfiguration</a>
/etc/resolv.conf	DNS resolution	Not unique to Cumulus Linux: <a href="http://wiki.debian.org/NetworkConfiguration#The_resolv.conf_configuration_file">wiki.debian.org/NetworkConfiguration#The_resolv.conf_configuration_file</a>	<a href="http://www.debian.org/doc/manuals/debian-reference/ch05.en.html">www.debian.org/doc/manuals/debian-reference/ch05.en.html</a>
/etc/quagga/	Routing application (responsible for BGP and OSPF)	Quagga Overview (see page 391)	<a href="http://packages.debian.org/wheezy/quagga">packages.debian.org/wheezy/quagga</a>
/etc/hostname	Configuration file for the hostname of the switch	Quick Start Guide#ConfiguringtheHostnameandTimeZone (see page 19)	<a href="http://wiki.debian.org/HowTo/ChangeHostname">wiki.debian.org/HowTo/ChangeHostname</a>
/etc/cumulus/ports.conf	Breakout cable configuration file	Layer 1 and Switch Port Attributes#ConfiguringBreakoutPorts (see page)	N/A; please read the guide on breakout cables
/etc/cumulus/switchd.conf	Switchd configuration	Configuring switchd (see page 126)	N/A; please read the guide on switchd configuration

## Additional Commonly Used Files

File Name and Location	Explanation	Cumulus Linux Documentation	Debian Documentation
/etc/motd	Message of the day		<a href="http://wiki.debian.org/motd#Wheezy">wiki.debian.org/motd#Wheezy</a>

File Name and Location	Explanation	Cumulus Linux Documentation	Debian Documentation
		Not unique to Cumulus Linux	
/etc/passwd	User account information	Not unique to Cumulus Linux	<a href="http://www.debian.org/doc/manuals/debian-reference/ch04.en.html">www.debian.org/doc/manuals/debian-reference/ch04.en.html</a>
/etc/shadow	Secure user account information	Not unique to Cumulus Linux	<a href="http://www.debian.org/doc/manuals/debian-reference/ch04.en.html">www.debian.org/doc/manuals/debian-reference/ch04.en.html</a>
/etc/group	Defines user groups on the switch	Not unique to Cumulus Linux	<a href="http://www.debian.org/doc/manuals/debian-reference/ch04.en.html">www.debian.org/doc/manuals/debian-reference/ch04.en.html</a>
/etc/llpd.conf	Link Layer Discover Protocol (LLDP) daemon configuration	Link Layer Discovery Protocol (see page 191)	<a href="http://packages.debian.org/wheezy/llpd">packages.debian.org/wheezy/llpd</a>
/etc/llpd.d/	Configuration directory for llpd	Link Layer Discovery Protocol (see page 191)	<a href="http://packages.debian.org/wheezy/llpd">packages.debian.org/wheezy/llpd</a>
/etc/nsswitch.conf	Name Service Switch (NSS) configuration file	LDAP Authentication and Authorization (see page 85)	<a href="http://wiki.debian.org/LDAP/NSS">wiki.debian.org/LDAP/NSS</a>
/etc/ssh/	SSH configuration files	SSH for Remote Access (see page 76)	<a href="http://wiki.debian.org/SSH">wiki.debian.org/SSH</a>
/etc/ldap/ldap.conf	Lightweight Directory Access Protocol configuration file	LDAP Authentication and Authorization (see page 85)	<a href="http://www.debian.org/doc/manuals/debian-reference/ch04.en.html">www.debian.org/doc/manuals/debian-reference/ch04.en.html</a>

- If you are using the root user account, consider including /root/.
- If you have custom user accounts, consider including /home/<username>/.

## Files That Should Never Be Migrated Between Versions or Boxes

File Name and Location	Explanation
/etc/adjtime	System clock adjustment data. NTP manages this automatically. It is incorrect when the switch hardware is replaced. Do not copy.

File Name and Location	Explanation
/etc/bcm.d/	Per-platform hardware configuration directory, created on first boot. Do not copy.
/etc/blkid.tab	Partition table. It should not be modified manually. Do not copy.
/etc/blkid.tab.old	A previous partition table; it should not be modified manually. Do not copy.
/etc/cumulus /init	Platform hardware-specific files. Do not copy.
/etc/default /clagd	Created and managed by <code>ifupdown2</code> . Do not copy.
/etc/default /grub	Grub <code>init</code> table; it should not be modified manually.
/etc/default /hwclock	Platform hardware-specific file. Created during first boot. Do not copy.
/etc/init	Platform initialization files. Do not copy.
/etc/init.d/	Platform initialization files. Do not copy.
/etc/fstab	Static info on filesystem. Do not copy.
/etc/image-release	System version data. Do not copy.
/etc/os-release	System version data. Do not copy.
/etc/lsb-release	System version data. Do not copy.
/etc/lvm/archive	Filesystem files. Do not copy.
/etc/lvm/backup	Filesystem files. Do not copy.
/etc/modules	Created during first boot. Do not copy.
/etc/modules-load.d/	Created during first boot. Do not copy.
/etc/sensors.d	Platform-specific sensor data. Created during first boot. Do not copy.

File Name and Location	Explanation
/root/.ansible	Ansible tmp files. Do not copy.
/home/cumulus/.ansible	Ansible tmp files. Do not copy.

## Upgrading Switches in an MLAG Pair

If you have a pair of Cumulus Linux switches as part of an **MLAG** (multi-chassis link aggregation) pair (see [page 244](#)), you should only upgrade each switch when it is in the *secondary role*. The upgrade path is as follows:

1. Upgrade Cumulus Linux on the switch already in the secondary role. This is the switch with the higher `clagd-priority` value.
2. Set the switch in the secondary role into the primary role by setting its `clagd-priority` to a value lower than the `clagd-priority` setting on the switch in the primary role.

```
cumulus@switch:~$ sudo clagctl priority VALUE
```

3. Upgrade the switch that just took on the secondary role.
4. Put that switch into the primary role again, if you so choose.

```
cumulus@switch:~$ sudo clagctl priority VALUE
```

For more information about setting the priority, see [Understanding Switch Roles](#) (see [page 247](#)).

## Upgrading Cumulus Linux: Choosing between a Binary Install vs. Package Upgrade

Network admins have two ways to upgrade Cumulus Linux:

- Upgrading only the changed packages, using `apt-get update` and `apt-get upgrade`. **This is the preferred method.**
- Performing a binary (full image) install of the new version, using ONIE. This is used when moving between major versions or if you want to install a clean image.

There are advantages and disadvantages to using these methods, which are outlined below.

### Upgrading Using Package Installs (`apt-get update && apt-get upgrade`)

Pros:

- Configuration data stays in place while the binaries are upgraded.

- Third-party apps stay in place.

Cons:

- This method works only if you are upgrading to a later minor release (like 3.0.x to 3.1.y) or a later maintenance release (like 3.0.1 to 3.0.2) from an earlier release in the same major release family **only** (like 2.5.2 to 2.5.5 or 3.0.0 to 3.0.1).
- Rollback is quite difficult and tedious.
- You can't choose the exact release version that you want to run.
- When you upgrade, you upgrade all packages to the latest available version.
- The upgrade process takes a while to complete, and various switch functions may be intermittently available during the upgrade.
- Some upgrade operations will terminate SSH sessions on the in-band (front panel) ports, leaving the user unable to monitor the upgrade process. As a workaround, use the [dtach tool](#).
- Just like the binary install method, you may have to reboot after the upgrade, lengthening the downtime.
- After you upgrade, user names and group names created by packages may be different on different switches, depending the configuration and package installation history.

To upgrade the switch by updating the packages:

1. Back up the configurations off the switch.
2. Fetch the latest update meta-data from the repository.

```
cumulus@switch$ sudo apt-get update
```

3. Upgrade all the packages to the latest distribution.

```
cumulus@switch$ sudo apt-get upgrade
```

4. Reboot the switch if you are notified by the upgrade messages that a System restart is required.

```
cumulus@switch$ sudo apt-get upgrade
... upgrade messages here ...

*** System restart required ***

cumulus@switch$ sudo reboot
```

5. Verify correct operation with the old configurations on new version.



If you are curious about which packages will be upgraded, use `apt-get upgrade --dry-run`. This will display the list of packages that will be upgraded without upgrading anything.



After you successfully upgrade Cumulus Linux, you may notice some results that you may or may not have expected:

- `apt-get upgrade` always updates the operating system to the most current version, so if you are currently running Cumulus Linux 3.0.1 and run `apt-get upgrade` on that switch, the packages will get upgraded to the latest versions contained in the latest 3.y.z release.
- When you run `cat /etc/image-release`, the output still shows the version of Cumulus Linux from the last binary install. So if you installed Cumulus Linux 3.0.0 as a full image install and then upgraded to 3.0.2 using `apt-get upgrade`, the output from `/etc/image-release` still shows: `IMAGE_RELEASE=3.0.0`. To see the current version of all the Cumulus Linux packages running on the switch, use `dpkg --list` or `dpkg -l`.

## ***Upgrading via Binary Install (ONIE)***

Pros:

- You choose the exact version that you want to upgrade to.
- This is the only method for upgrading to a new major (X.0) version. For example, when you are upgrading from 2.5.5 to 3.0.

Cons:

- Configuration data must be moved to the new OS via ZTP while the OS is first booted, or soon afterwards via out-of-band management.
- Moving the configuration file can go wrong in various ways:
  - Identifying all the locations of config data is not always an easy task. See section above on [Understanding the Locations of Configuration Data \(see page 44\)](#).
  - Config file changes in the new version may cause merge conflicts that go undetected.
- If config files aren't restored correctly, the user may be unable to attach to the switch from in-band management. Hence, out-of-band connectivity (eth0 or console) is recommended.
- The installer takes a while to complete.
- Third-party apps must be reinstalled and reconfigured afterwards.

To upgrade the switch by running a binary install:

1. Back up the configurations off the switch.
2. Install the binary image, following the instructions at [Installing a New Cumulus Linux Image \(see page 31\)](#).
3. Restore the configuration files to the new version — ideally via automation.
4. Verify correct operation with the old configurations on the new version.
5. Reinstall third party apps and associated configurations.

## ***Rolling Back a Cumulus Linux Installation***

Rolling back to an earlier release after upgrading the packages on the switch follows the same procedure as described for the Linux host OS rollback above. There are three main strategies, and all require detailed planning and execution:

- Back out individual packages: If the problematic package is identified, the network admin can downgrade the affected package directly. In rare cases the configuration files may have to be restored from backup, or edited to back out any changes that were automatically made by the upgrade package.
- Flatten and rebuild: If the OS becomes unusable, you can use orchestration tools to reinstall the previous OS release from scratch and then automatically rebuild the configuration.
- Backup and restore: Another common strategy is to restore to a previous state via a backup captured before the upgrade.

Which method you employ is specific to your deployment strategy, so providing detailed steps for each scenario is outside the scope of this document.

## **Third Party Package Considerations**

Note that if you install any third party apps on a Cumulus Linux switch, any configuration data will likely be installed into the `/etc` directory, but it is not guaranteed. It is the responsibility of the network admin to understand the behavior and config file information of any third party packages installed on a Cumulus Linux switch.

After you upgrade the OS using a full binary install, you will need to reinstall any third party packages or any Cumulus Linux add-on packages, such as `vxsnd` or `vxrd`.

## **Installation and Upgrade Workflow in Cumulus Linux 3.0**

Cumulus Linux 3.0 completely embraces the Linux and Debian upgrade workflow. In this paradigm, a base image is installed using an installer, then any upgrades within that release train (major version, like 3.y.z) are done using `apt-get update && apt-get upgrade`. Any packages that have been changed since the base install get upgraded in place from the repository.

The huge advantage of this approach is that all switch configuration files remain untouched, or in rare cases merged (using the Debian merge function) during the package upgrade.

However, when upgrading a switch from a previous release train — for example, Cumulus Linux 2.5 — a mechanism is required to migrate the configuration files over to the new 3.0 installation. This is the perfect opportunity to use automation and orchestration tools to backup the configuration files, examine them to verify correctness with 3.0, and then to redeploy the configuration files on the new 3.0 installation.

## **Caveats When Migrating Configuration Files Between Cumulus Linux 2.5.z and 3.0**

Generally, the configuration files in Cumulus Linux 2.5.z should be able to migrate to version 3.0 without any problems, but there are some known issues listed below and there may be additional issues with a customer's particular setup.

Known caveats when migrating files from version 2.x to 3.0:

- Some configuration files should never be migrated between versions or while replacing hardware. The [Files that Should Never be Migrated \(see page 46\)](#) table above contains a list of files that should never be migrated.
- `/etc/passwd` and `/etc/shadow` should not be migrated to 3.0 directly. The example below and the ansible script included with [Config File Migration Script](#) explicitly excludes these two files from the backup archive. The default password for the 'cumulus' user will need to be changed, and any locally created users should be added to the 3.0 installation after upgrade.

- `/etc/apt/sources.list` must be completely updated with a new 3.0 repository and repository structure. Repositories from Cumulus Linux 2.5 must be removed. If there are any custom repositories on the switch, they need to be migrated into the 3.0 `sources.list` file or the `sources.d/` directory.

## **Using the Config File Migration Script to Identify and Move Files to Cumulus Linux 3.0**

You can use the [Config File Migration Script](#) with the `--backup` option to create a backup archive of configuration files in version 2.5, copy them off the box, then install them on the version 3.0 switch. Note that you need to follow the previous section about caveats when migrating configuration files.

**!** You **cannot** use the Config File Migration Script to upgrade from Cumulus Linux 3.0.0 to 3.0.1. Use `apt-get` instead, as documented in the [release notes](#).

The example provided below excludes `/etc/apt`, `/etc/passwd` and `/etc/shadow` from the backup archive.

1. Back up the version 2.5.z files.

**Optional:** Use the Ansible playbook included with the [Config File Migration script](#) to automate the backup of all your Cumulus Linux 2.5 switches. See the section below on [Using Automation Tools to Backup Configurations](#) (see page 53) for more details.

```
# Make a temp dir
loc=$(mktemp -d)
# Create a backup archive to the temp dir
sudo ./config_file_changes --backup --backup_dir $loc --exclude
/etc/apt,/etc/passwd,/etc/shadow
# Copy the archive and log file to an external server
sudo scp -r $loc/* user@my_external_server:.
```

2. Install Cumulus Linux 3.0 onto the switch using ONIE (see page 31).
3. Reinstall the files from the config file archive to the newly installed 3.0 switch.

```
# On the switch, copy the config file archive back from the
server:
scp user@my_external_server:PATH/SWITCHNAME-config-archive-
DATE_TIME.tar.gz .
# Untar the archive to the root of the box
sudo tar -C / -xvf SWITCHNAME-config-archive-DATE_TIME.tar.gz
```

**!** Be aware that version 2.5.z configurations are not guaranteed to work in Cumulus Linux 3.0. You should test the restoration and proper operation of the Cumulus Linux 2.5.z configuration in Cumulus Linux 3.0 on a non-production switch or in a Cumulus VX image, since every deployment is unique.

## Using Automation Tools to Back Up 2.5.z Configurations

Adopting the use of orchestration tools like Ansible, Chef or Puppet for configuration management greatly increases the speed and accuracy of the next major upgrade; they also enable the quick swap of failed switch hardware. Included with the [Config Migration Script](#) is an Ansible playbook that can be used to create a backup archive of Cumulus Linux 2.5.z switch configuration files and to retrieve them to a central server — automating step 1 of the previous section for all deployed Cumulus Linux 2.5.z switches. This is a quick start on the road to setting up automated configuration and control for your deployment. For more details on integrating automation into your Cumulus Linux deployment, see the [Automation Solutions](#) section on [cumulusnetworks.com](http://cumulusnetworks.com).

## Adding and Updating Packages

You use the Advanced Packaging Tool (APT) to manage additional applications (in the form of packages) and to install the latest updates.

### Contents

(Click to expand)

- [Contents \(see page 53\)](#)
- [Commands \(see page 53\)](#)
- [Updating the Package Cache \(see page 53\)](#)
- [Listing Available Packages \(see page 55\)](#)
- [Adding a Package \(see page 56\)](#)
- [Listing Installed Packages \(see page 57\)](#)
- [Upgrading to Newer Versions of Installed Packages \(see page 57\)](#)
  - [Upgrading a Single Package \(see page 57\)](#)
  - [Upgrading All Packages \(see page 57\)](#)
- [Adding Packages from Another Repository \(see page 58\)](#)
- [Configuration Files \(see page 59\)](#)
- [Useful Links \(see page 59\)](#)

### Commands

- `apt-get`
- `apt-cache`
- `dpkg`

### Updating the Package Cache

To work properly, APT relies on a local cache of the available packages. You must populate the cache initially, and then periodically update it with `apt-get update`:

```
cumulus@switch:~$ sudo apt-get update
Get:1 http://repo3.cumulusnetworks.com CumulusLinux-3 InRelease [7,624 B]
Get:2 http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
InRelease [7,555 B]
Get:3 http://repo3.cumulusnetworks.com CumulusLinux-3-updates InRelease
[7,660 B]
Get:4 http://repo3.cumulusnetworks.com CumulusLinux-3/cumulus Sources [20 B]
Get:5 http://repo3.cumulusnetworks.com CumulusLinux-3/upstream Sources [20
B]
Get:6 http://repo3.cumulusnetworks.com CumulusLinux-3/cumulus amd64
Packages [38.4 kB]
Get:7 http://repo3.cumulusnetworks.com CumulusLinux-3/upstream amd64
Packages [445 kB]
Get:8 http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/cumulus Sources [20 B]
Get:9 http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/upstream Sources [11.8 kB]
Get:10 http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/cumulus amd64 Packages [20 B]
Get:11 http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/upstream amd64 Packages [8,941 B]
Get:12 http://repo3.cumulusnetworks.com CumulusLinux-3-updates/cumulus
Sources [20 B]
Get:13 http://repo3.cumulusnetworks.com CumulusLinux-3-updates/upstream
Sources [776 B]
Get:14 http://repo3.cumulusnetworks.com CumulusLinux-3-updates/cumulus
amd64 Packages [38.4 kB]
Get:15 http://repo3.cumulusnetworks.com CumulusLinux-3-updates/upstream
amd64 Packages [444 kB]
Ign http://repo3.cumulusnetworks.com CumulusLinux-3/cumulus Translation-
en_US
Ign http://repo3.cumulusnetworks.com CumulusLinux-3/cumulus Translation-en
Ign http://repo3.cumulusnetworks.com CumulusLinux-3/upstream Translation-
en_US
Ign http://repo3.cumulusnetworks.com CumulusLinux-3/upstream Translation-en
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/cumulus Translation-en_US
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/cumulus Translation-en
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/upstream Translation-en_US
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-security-updates
/upstream Translation-en
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-updates/cumulus
Translation-en_US
```

```

Ign http://repo3.cumulusnetworks.com CumulusLinux-3-updates/cumulus
Translation-en
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-updates/upstream
Translation-en_US
Ign http://repo3.cumulusnetworks.com CumulusLinux-3-updates/upstream
Translation-en
Fetched 1,011 kB in 1s (797 kB/s)
Reading package lists... Done

```

## ***Listing Available Packages***

Once the cache is populated, use `apt-cache` to search the cache to find the packages you are interested in or to get information about an available package. Here are examples of the `search` and `show` sub-commands:

```

cumulus@switch:~$ apt-cache search tcp
socat - multipurpose relay for bidirectional data transfer
quagga-doc - documentation files for quagga
fakeroot - tool for simulating superuser privileges
quagga - BGP/OSPF/RIP routing daemon
tcpdump - command-line network traffic analyzer
openssh-server - secure shell (SSH) server, for secure access from remote
machines
openssh-sftp-server - secure shell (SSH) sftp server module, for SFTP
access from remote machines
python-dpkt - Python packet creation / parsing module
libfakeroot - tool for simulating superuser privileges - shared libraries
openssh-client - secure shell (SSH) client, for secure access to remote
machines
rsyslog - reliable system and kernel logging daemon
libwrap0 - Wietse Venema's TCP wrappers library
netbase - Basic TCP/IP networking system

cumulus@switch:~$ apt-cache show tcpdump
Package: tcpdump
Status: install ok installed
Priority: optional
Section: net
Installed-Size: 1092
Maintainer: Romain Francoise <rfrancoise@debian.org>
Architecture: amd64
Multi-Arch: foreign
Version: 4.6.2-5+deb8u1
Depends: libc6 (>= 2.14), libpcap0.8 (>= 1.5.1), libssl1.0.0 (>= 1.0.0)

```

```
Description: command-line network traffic analyzer
This program allows you to dump the traffic on a network. tcpdump
is able to examine IPv4, ICMPv4, IPv6, ICMPv6, UDP, TCP, SNMP, AFS
BGP, RIP, PIM, DVMRP, IGMP, SMB, OSPF, NFS and many other packet
types.

.
It can be used to print out the headers of packets on a network
interface, filter packets that match a certain expression. You can
use this tool to track down network problems, to detect attacks
or to monitor network activities.

Description-md5: f01841bfda357d116d7ff7b7a47e8782
Homepage: http://www.tcpdump.org/
cumulus@switch:~$
```



The search commands look for the search terms not only in the package name but in other parts of the package information. Consequently, it will match on more packages than you would expect.

## ***Adding a Package***

In order to add a new package, first ensure the package is not already installed in the system:

```
cumulus@switch:~$ dpkg -l | grep {name of package}
```

If the package is installed already, ensure it's the version you need. If it's an older version, then update the package from the Cumulus Linux repository:

```
cumulus@switch:~$ sudo apt-get update
```

If the package is not already on the system, add it by running `apt-get install`. This retrieves the package from the Cumulus Linux repository and installs it on your system together with any other packages that this package might depend on.

For example, the following adds the package `tcpreplay` to the system:

```
cumulus@switch:~$ sudo apt-get install tcpreplay
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
tcpreplay
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
```

```

Need to get 436 kB of archives.
After this operation, 1008 kB of additional disk space will be used.
Get:1 https://repo.cumulusnetworks.com/ CumulusLinux-1.5/main tcpreplay
amd64 4.6.2-5+deb8u1 [436 kB]
Fetched 436 kB in 0s (1501 kB/s)
Selecting previously unselected package tcpreplay.
(Reading database ... 15930 files and directories currently installed.)
Unpacking tcpreplay (from .../tcpreplay_4.6.2-5+deb8u1_amd64.deb) ...
Processing triggers for man-db ...
Setting up tcpreplay (4.6.2-5+deb8u1) ...
cumulus@switch:~$
```

## ***[Listing Installed Packages](#)***

The APT cache contains information about all the packages available on the repository. To see which packages are actually installed on your system, use `dpkg`. The following example lists all the packages on the system that have "tcp" in their package names:

```

cumulus@switch:~$ dpkg -l \*tcp\*
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-await/Trig-
pend
| / Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
|| / Name                           Version            Architecture
Description
=====
un  tcpd                         <none>           <none>
  (no description available)
ii  tcpdump                        4.6.2-5+deb8u1    amd64
  command-line network traffic analyzer
cumulus@switch:~$
```

## ***[Upgrading to Newer Versions of Installed Packages](#)***

### ***[Upgrading a Single Package](#)***

A single package can be upgraded by simply installing that package again with `apt-get install`. You should perform an update first so that the APT cache is populated with the latest information about the packages.

To see if a package needs to be upgraded, use `apt-cache show <pkgname>` to show the latest version number of the package. Use `dpkg -l <pkgname>` to show the version number of the installed package.

## Upgrading All Packages

You can update all packages on the system by running `apt-get update`, then `apt-get upgrade`. This upgrades all installed versions with their latest versions but will not install any new packages.

## Adding Packages from Another Repository

As shipped, Cumulus Linux searches the Cumulus Linux repository for available packages. You can add additional repositories to search by adding them to the list of sources that `apt-get` consults. See `man sources.list` for more information.

-  For several packages, Cumulus Networks has added features or made bug fixes and these packages must not be replaced with versions from other repositories. Cumulus Linux has been configured to ensure that the packages from the Cumulus Linux repository are always preferred over packages from other repositories.

If you want to install packages that are not in the Cumulus Linux repository, the procedure is the same as above with one additional step.

-  Packages not part of the Cumulus Linux Repository have generally not been tested, and may not be supported by Cumulus Linux support.

Installing packages outside of the Cumulus Linux repository requires the use of `apt-get`, but, depending on the package, `easy-install` and other commands can also be used.

To install a new package, please complete the following steps:

1. First, ensure package is not already installed in the system. Use the `dpkg` command:

```
cumulus@switch:~$ dpkg -l | grep {name of package}
```

2. If the package is installed already, ensure it's the version you need. If it's an older version, then update the package from the Cumulus Linux repository:

```
cumulus@switch:~$ sudo apt-get update
cumulus@switch:~$ sudo apt-get install {name of package}
```

3. If the package is not on the system, then most likely the package source location is also **not** in the `/etc/apt/sources.list` file. If the source for the new package is **not** in `sources.list`, please edit and add the appropriate source to the file. For example, add the following if you wanted a package from the Debian repository that is **not** in the Cumulus Linux repository:

```
deb http://http.us.debian.org/debian jessie main
deb http://security.debian.org/ jessie/updates main
```

Otherwise, the repository may be listed in `/etc/apt/sources.list` but is commented out, as can be the case with the early-access repository:

```
#deb http://repo3.cumulusnetworks.com/repo CumulusLinux-3-early-access
cumulus
```

To uncomment the repository, remove the # at the start of the line, then save the file:

```
deb http://repo3.cumulusnetworks.com/repo CumulusLinux-3-early-access
cumulus
```

4. Run `apt-get update` then install the package:

```
cumulus@switch:~$ sudo apt-get update
cumulus@switch:~$ sudo apt-get install {name of package}
```

## Configuration Files

- `/etc/apt/apt.conf`
- `/etc/apt/preferences`
- `/etc/apt/sources.list`

## Useful Links

- [Debian GNU/Linux FAQ, Ch 8 Package management tools](#)
- [man pages for apt-get, dpkg, sources.list, apt\\_preferences](#)

## Zero Touch Provisioning - ZTP

*Zero touch provisioning (ZTP)* enables network devices to be quickly deployed in large-scale environments. Data center engineers only need to rack and stack the switch, install Cumulus Linux via ONIE, then connect the switch to the management network — or alternatively, insert a USB stick and boot the switch. From here, the provisioning process can start automatically and deploy a configuration.

The provisioning framework allows for a one-time, user-provided script to be executed. This script can be used to add the switch to a configuration management (CM) platform such as [Puppet](#), [Chef](#), [CFEngine](#) or even a custom, home-grown tool.

In addition, you can use the `ztp` command in Cumulus Linux to manually invoke your provisioning script.

ZTP in Cumulus Linux can occur automatically in one of the following ways, in this order:

- Via a local file
- Using a USB drive inserted into the switch (ZTP-USB)

- Via DHCP

Each method is discussed in greater detail below.



The standard Cumulus Linux license requires you to page through the license file before accepting the terms, which can hinder an unattended installation like zero touch provisioning. To request a license without the EULA, email [licensing@cumulusnetworks.com](mailto:licensing@cumulusnetworks.com).

## Contents

Click to expand...

- Commands (see page 60)
- Zero Touch Provisioning Using a Local File (see page 60)
- Zero Touch Provisioning Using USB (ZTP-USB) (see page 61)
- Zero Touch Provisioning over DHCP (see page 61)
  - Triggering ZTP over DHCP (see page 62)
  - Configuring The DHCP Server (see page 62)
  - Detailed Look at HTTP Headers (see page 63)
- Writing ZTP Scripts (see page 63)
  - Example ZTP Scripts (see page 64)
- Testing and Debugging ZTP Scripts (see page 65)
- Manually Using the ztp Command (see page 69)
- Notes (see page 71)

## Commands

- ztp

## Zero Touch Provisioning Using a Local File

ZTP only looks once for a ZTP script on the local file system when the switch boots. ZTP searches for an install script that matches an ONIE-style waterfall in `/var/lib/cumulus/ztp`, looking for the most specific name first, and ending at the most generic:

- `'cumulus-ztp-' + architecture + '-' + vendor + '_' + model + '-r' + revision`
- `'cumulus-ztp-' + architecture + '-' + vendor + '_' + model`
- `'cumulus-ztp-' + vendor + '_' + model`
- `'cumulus-ztp-' + architecture`
- `'cumulus-ztp'`

For example:

```
cumulus-ztp-amd64-cel_pebble-rUNKNOWN  
cumulus-ztp-amd64-cel_pebble  
cumulus-ztp-cel_pebble  
cumulus-ztp-amd64  
cumulus-ztp
```

You can also trigger the ZTP process manually by running the `ztp --run <URL>` command, where the URL is the path to the ZTP script.

## Zero Touch Provisioning Using USB (ZTP-USB)



This feature has been tested only with "thumb" drives, not an actual external large USB hard drive.

If the `ztp` process did not discover a local script, it tries once to locate an inserted but unmounted USB drive. If it discovers one, it begins the ZTP process.

Cumulus Linux supports the use of a FAT32, FAT16, or VFAT-formatted USB drive as an installation source for ZTP scripts. You must plug in the USB stick **before** you power up the switch.

At minimum, the script should:

- Install the Cumulus Linux operating system and license.
- Copy over a basic configuration to the switch.
- Restart the switch or the relevant server to get `switchd` up and running with that configuration.

Follow these steps to perform zero touch provisioning using USB:

1. Copy the Cumulus Linux license and installation image to the USB stick.
2. The `ztp` process searches the root filesystem of the newly mounted device for filenames matching an **ONIE-style waterfall** (see the patterns and examples above), looking for the most specific name first, and ending at the most generic.
3. The script's contents are parsed to ensure it contains the **CUMULUS-AUTOPROVISIONING** flag (see [example scripts \(see page 64\)](#)).



The USB device is mounted to a temporary directory under `/tmp` (for example, `/tmp/tmpigGgjf/`). In order to reference files on the USB, use the environment variable `ZTP_USB_MOUNTPOINT` to refer to the USB root partition.

## Zero Touch Provisioning over DHCP

If the `ztp` process did not discover a local/ONIE script or applicable USB drive, it checks DHCP every 10 seconds for up to 5 minutes for the presence of a ZTP URL specified in `/var/run/ztp.dhcp`. The URL can be any of HTTP, HTTPS, FTP or TFTP.

For ZTP using DHCP, provisioning initially takes place over the management network and is initiated via a DHCP hook. A DHCP option is used to specify a configuration script. This script is then requested from the Web server and executed locally on the switch.

The zero touch provisioning process over DHCP follows these steps:

1. The first time you boot Cumulus Linux, eth0 is configured for DHCP and makes a DHCP request.
2. The DHCP server offers a lease to the switch.
3. If option 239 is present in the response, the zero touch provisioning process itself will start.
4. The zero touch provisioning process requests the contents of the script from the URL, sending additional [HTTP headers \(see page 63\)](#) containing details about the switch.
5. The script's contents are parsed to ensure it contains the `CUMULUS-AUTOPROVISIONING` flag (see [example scripts \(see page 64\)](#)).
6. If provisioning is necessary, then the script executes locally on the switch with root privileges.
7. The return code of the script gets examined. If it is 0, then the provisioning state is marked as complete in the autoprovisioning configuration file.

## **Triggering ZTP over DHCP**

If provisioning has not already occurred, it is possible to trigger the zero touch provisioning process over DHCP when eth0 is set to use DHCP and one of the following events occur:

- Booting the switch
- Plugging a cable into or unplugging it from the eth0 port
- Disconnecting then reconnecting the switch's power cord

You can also run the `ztp --run <URL>` command, where the URL is the path to the ZTP script.

## **Configuring The DHCP Server**

During the DHCP process over eth0, Cumulus Linux will request DHCP option 239. This option is used to specify the custom provisioning script.

For example, the `/etc/dhcp/dhcpd.conf` file for an ISC DHCP server would look like:

```
option cumulus-provision-url code 239 = text;

subnet 192.0.2.0 netmask 255.255.255.0 {
  range 192.0.2.100 192.168.0.200;
  option cumulus-provision-url "http://192.0.2.1/demo.sh";
}
```

Additionally, the hostname of the switch can be specified via the `host-name` option:

```
subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.100 192.168.0.200;
  option cumulus-provision-url "http://192.0.2.1/demo.sh";
  host dc1-tor-swl { hardware ethernet 44:38:39:00:1a:6b; fixed-
    address 192.168.0.101; option host-name "dc1-tor-swl"; }
```

}

## Detailed Look at HTTP Headers

The following HTTP headers are sent in the request to the webserver to retrieve the provisioning script:

Header	Value	Example
-----	-----	-----
User-Agent		CumulusLinux-
AutoProvision/ <b>0.4</b>		
CUMULUS-ARCH	CPU architecture	x86_64
CUMULUS-BUILD		<b>3.0.0-5c6829a-2013</b>
<b>09251712-final</b>		
CUMULUS-LICENSE-INSTALLED	Either 0 or 1	<b>1</b>
CUMULUS-MANUFACTURER		odm
CUMULUS-PRODUCTNAME		switch_model
CUMULUS-SERIAL		XYZ123004
CUMULUS-VERSION		<b>3.0.0</b>
CUMULUS-PROV-COUNT		0
CUMULUS-PROV-MAX		32

## Writing ZTP Scripts



Remember to include the following line in any of the supported scripts which are expected to be run via the autoprovisioning framework.

```
# CUMULUS-AUTOPROVISIONING
```

This line is required somewhere in the script file in order for execution to occur.

The script must contain the **CUMULUS-AUTOPROVISIONING** flag. This can be in a comment or remark and does not need to be echoed or written to **stdout**.

The script can be written in any language currently supported by Cumulus Linux, such as:

- Perl
- Python
- Ruby
- Shell

The script must return an exit code of 0 upon success, as this triggers the autoprovisioning process to be marked as complete in the autoprovisioning configuration file.

## Example ZTP Scripts

The following script install Cumulus Linux and its license from USB and applies a configuration:

```
#!/bin/bash
function error() {
    echo -e "\e[0;33mERROR: The Zero Touch Provisioning script failed
while running the command $BASH_COMMAND at line $BASH_LINENO.\e[0m" >&
2
    exit 1
}

# Log all output from this script
exec >/var/log/autoprovision 2>&1

trap error ERR

#Add Debian Repositories
echo "deb http://http.us.debian.org/debian jessie main" >> /etc/apt
/sources.list
echo "deb http://security.debian.org/ jessie/updates main" >> /etc/apt
/sources.list

#Update Package Cache
apt-get update -y

#Install netshow diagnostics commands
apt-get install -y netshow htop nmap

#Load interface config from usb
cp ${ZTP_USB_MOUNTPOINT}/interfaces /etc/network/interfaces

#Load port config from usb
#   (if breakout cables are used for certain interfaces)
cp ${ZTP_USB_MOUNTPOINT}/ports.conf /etc/cumulus/ports.conf

#Install a License from usb and restart switchd
cl-license -i ${ZTP_USB_MOUNTPOINT}/license.txt && systemctl restart
switchd.service

#Reload interfaces to apply loaded config
ifreload -a

#Output state of interfaces
netshow interface

# CUMULUS-AUTOPROVISIONING
exit 0
```

Here is a simple script to install puppet:

```

#!/bin/bash
function error() {
    echo -e "\e[0;33mERROR: The Zero Touch Provisioning script failed
while running the command $BASH_COMMAND at line $BASH_LINENO.\e[0m" >&
2
    exit 1
}
trap error ERR
apt-get update -y
apt-get upgrade -y
apt-get install puppet -y
sed -i /etc/default/puppet -e 's/START=no/START=yes/'
sed -i /etc/puppet/puppet.conf -e 's/\\[main\\]/\\[main\\]
\npluginsync=true/'
systemctl restart puppet.service
# CUMULUS-AUTOPROVISIONING
exit 0

```

This script illustrates how to specify an internal APT mirror and `puppet` master:

```

#!/bin/bash
function error() {
    echo -e "\e[0;33mERROR: The Zero Touch Provisioning script failed
while running the command $BASH_COMMAND at line $BASH_LINENO.\e[0m" >&
2
    exit 1
}
trap error ERR
sed -i /etc/apt/sources.list -e 's/repo.cumulusnetworks.com/labrepo.
mycompany.com/'
apt-get update -y
apt-get upgrade -y
apt-get install puppet -y
sed -i /etc/default/puppet -e 's/START=no/START=yes/'
sed -i /etc/puppet/puppet.conf -e 's/\\[main\\]/\\[main\\]
\npluginsync=true/'
sed -i /etc/puppet/puppet.conf -e 's/\\[main\\]/\\[main\\]
\nserver=labpuppet.mycompany.com/'
systemctl restart puppet.service
# CUMULUS-AUTOPROVISIONING
exit 0

```

Now `puppet` can take over management of the switch, configuration authentication, changing the default root password, and setting up interfaces and routing protocols.

Several ZTP example scripts are available in the [Cumulus GitHub repository](#).

## Testing and Debugging ZTP Scripts

There are a few commands you can use to test and debug your ZTP scripts.

You can use verbose mode to debug your script and see where your script failed. Include the `-v` option when you run `ztp`:

```
cumulus@switch:~$ sudo ztp -v -r http://192.0.2.1/demo.sh
Attempting to provision via ZTP Manual from http://192.0.2.1/demo.sh

Broadcast message from root@dell-s6000-01 (ttyS0) (Tue May 10 22:44:17
2016):

ZTP: Attempting to provision via ZTP Manual from http://192.0.2.1/demo.sh
ZTP Manual: URL response code 200
ZTP Manual: Found Marker CUMULUS-AUTOPROVISIONING
ZTP Manual: Executing http://192.0.2.1/demo.sh
error: ZTP Manual: Payload returned code 1
error: Script returned failure
```

You can also run `ztp -s` to get more information about the current state of ZTP.

```
ZTP INFO:

State          enabled
Version        1.0
Result         Script Failure
Date          Tue May 10 22:42:09 2016 UTC
Method         ZTP DHCP
URL           http://192.0.2.1/demo.sh
```

If ZTP ran when the switch booted and not manually, you can run the `sudo systemctl -l status ztp.service` then `journalctl -l -u ztp.service` to see if any failures occur:

```
cumulus@switch:~$ sudo systemctl -l status ztp.service
ztp.service - Cumulus Linux ZTP
   Loaded: loaded (/lib/systemd/system/ztp.service; enabled)
   Active: failed (Result: exit-code) since Wed 2016-05-11 16:38:45
             UTC; 1min 47s ago
     Docs: man:ztp(8)
   Process: 400 ExecStart=/usr/sbin/ztp -b (code=exited, status=1/FAILU
RE)
```

```

Main PID: 400 (code=exited, status=1/FAILURE)

May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP USB: Device not found
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Looking for
or ZTP Script provided by DHCP
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: Attempting to
provision via ZTP DHCP from http://192.0.2.1/demo.sh
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: URL
response code 200
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Found
Marker CUMULUS-AUTOPROVISIONING
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP:
Executing http://192.0.2.1/demo.sh
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Payload
returned code 1
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: Script returned
failure
May 11 16:38:45 dell-s6000-01 systemd[1]: ztp.service: main process
exited, code=exited, status=1/FAILURE
May 11 16:38:45 dell-s6000-01 systemd[1]: Unit ztp.service entered
failed state.
cumulus@switch:~$ 
cumulus@switch:~$ sudo journalctl -l -u ztp.service --no-pager
-- Logs begin at Wed 2016-05-11 16:37:42 UTC, end at Wed 2016-05-11 16
:40:39 UTC. --
May 11 16:37:45 cumulus ztp[400]: ztp [400]: /var/lib/cumulus/ztp:
Sate Directory does not exist. Creating it...
May 11 16:37:45 cumulus ztp[400]: ztp [400]: /var/run/ztp.lock: Lock
File does not exist. Creating it...
May 11 16:37:45 cumulus ztp[400]: ztp [400]: /var/lib/cumulus/ztp
/ztp_state.log: State File does not exist. Creating it...
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Looking for
ZTP local Script
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Waterfall
search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-dell_s6000_s1220-
rUNKNOWN
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Waterfall
search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-dell_s6000_s1220
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Waterfall
search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-dell
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Waterfall
search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP LOCAL: Waterfall
search for /var/lib/cumulus/ztp/cumulus-ztp
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP USB: Looking for
unmounted USB devices
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP USB: Parsing
partitions
May 11 16:37:45 cumulus ztp[400]: ztp [400]: ZTP USB: Device not found
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Looking for
or ZTP Script provided by DHCP

```

```

May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: Attempting to
provision via ZTP DHCP from http://192.0.2.1/demo.sh
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: URL
response code 200
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Found
Marker CUMULUS-AUTOPROVISIONING
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP:
Executing http://192.0.2.1/demo.sh
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: ZTP DHCP: Payload
returned code 1
May 11 16:38:45 dell-s6000-01 ztp[400]: ztp [400]: Script returned
failure
May 11 16:38:45 dell-s6000-01 systemd[1]: ztp.service: main process
exited, code=exited, status=1/FAILURE
May 11 16:38:45 dell-s6000-01 systemd[1]: Unit ztp.service entered
failed state.

```

Instead of running journalctl, you can see the log history by running:

```

cumulus@switch:~$ cat /var/log/syslog | grep ztp
2016-05-11T16:37:45.132583+00:00 cumulus ztp [400]: /var/lib/cumulus
/ztp: State Directory does not exist. Creating it...
2016-05-11T16:37:45.134081+00:00 cumulus ztp [400]: /var/run/ztp.
lock: Lock File does not exist. Creating it...
2016-05-11T16:37:45.135360+00:00 cumulus ztp [400]: /var/lib/cumulus
/ztp/ztp_state.log: State File does not exist. Creating it...
2016-05-11T16:37:45.185598+00:00 cumulus ztp [400]: ZTP LOCAL:
Looking for ZTP local Script
2016-05-11T16:37:45.485084+00:00 cumulus ztp [400]: ZTP LOCAL:
Waterfall search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-
dell_s6000_s1220-rUNKNOWN
2016-05-11T16:37:45.486394+00:00 cumulus ztp [400]: ZTP LOCAL:
Waterfall search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-
dell_s6000_s1220
2016-05-11T16:37:45.488385+00:00 cumulus ztp [400]: ZTP LOCAL:
Waterfall search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64-dell
2016-05-11T16:37:45.489665+00:00 cumulus ztp [400]: ZTP LOCAL:
Waterfall search for /var/lib/cumulus/ztp/cumulus-ztp-x86_64
2016-05-11T16:37:45.490854+00:00 cumulus ztp [400]: ZTP LOCAL:
Waterfall search for /var/lib/cumulus/ztp/cumulus-ztp
2016-05-11T16:37:45.492296+00:00 cumulus ztp [400]: ZTP USB: Looking for
unmounted USB devices
2016-05-11T16:37:45.493525+00:00 cumulus ztp [400]: ZTP USB: Parsing
partitions
2016-05-11T16:37:45.636422+00:00 cumulus ztp [400]: ZTP USB: Device
not found
2016-05-11T16:38:43.372857+00:00 cumulus ztp [1805]: Found ZTP DHCP
Request
2016-05-11T16:38:45.696562+00:00 cumulus ztp [400]: ZTP DHCP: Looking
for ZTP Script provided by DHCP

```

```

2016-05-11T16:38:45.698598+00:00 cumulus ztp [400]: Attempting to
provision via ZTP DHCP from http://192.0.2.1/demo.sh
2016-05-11T16:38:45.816275+00:00 cumulus ztp [400]: ZTP DHCP: URL
response code 200
2016-05-11T16:38:45.817446+00:00 cumulus ztp [400]: ZTP DHCP: Found
Marker CUMULUS-AUTOPROVISIONING
2016-05-11T16:38:45.818402+00:00 cumulus ztp [400]: ZTP DHCP:
Executing http://192.0.2.1/demo.sh
2016-05-11T16:38:45.834240+00:00 cumulus ztp [400]: ZTP DHCP: Payload
returned code 1
2016-05-11T16:38:45.835488+00:00 cumulus ztp [400]: Script returned
failure
2016-05-11T16:38:45.876334+00:00 cumulus systemd[1]: ztp.service:
main process exited, code=exited, status=1/FAILURE
2016-05-11T16:38:45.879410+00:00 cumulus systemd[1]: Unit ztp.service
entered failed state.

```

If you see that the issue is a script failure, you can modify the script and then run `ztp` manually using `ztp -v -r <URL/path to that script>`, as above.

```

cumulus@switch:~$ sudo ztp -v -r http://192.0.2.1/demo.sh
Attempting to provision via ZTP Manual from http://192.0.2.1/demo.sh

Broadcast message from root@dell-s6000-01 (ttyS0) (Tue May 10 22:44:17
2016):

ZTP: Attempting to provision via ZTP Manual from http://192.0.2.1/demo.sh
ZTP Manual: URL response code 200
ZTP Manual: Found Marker CUMULUS-AUTOPROVISIONING
ZTP Manual: Executing http://192.0.2.1/demo.sh
error: ZTP Manual: Payload returned code 1
error: Script returned failure
cumulus@switch:~$ sudo ztp -s
State      enabled
Version    1.0
Result     Script Failure
Date       Tue May 10 22:44:17 2016 UTC
Method     ZTP Manual
URL       http://192.0.2.1/demo.sh

```

## ***Manually Using the ztp Command***

To enable zero touch provisioning, use the `-e` option:

```
cumulus@switch:~$ sudo ztp -e
```



Enabling **ztp** means that **ztp** will try to occur the next time the switch boots. However, if ZTP already occurred on a previous boot up or if a manual configuration has been found, ZTP will just exit without trying to look for any script.

ZTP checks for these manual configurations during bootup:

- Password changes
- Users and groups changes
- Packages changes
- Interfaces changes
- The presence of an installed license

When the switch is booted for the very first time, ZTP records the state of some important files that are most likely going to be modified after that the switch is configured. If ZTP is still enabled after a reboot, ZTP will compare the recorded state to the current state of these files. If they do not match, ZTP considers that the switch has already been provisioned and exits. These files are only erased after a reset.

To reset **ztp** to its original state, use the **-R** option. This removes the **ztp** directory and **ztp** runs the next time the switch reboots.

```
cumulus@switch:~$ sudo ztp -R
```

To disable zero touch provisioning, use the **-d** option:

```
cumulus@switch:~$ sudo ztp -d
```

To force provisioning to occur and ignore the status listed in the configuration file use the **-r** option:

```
cumulus@switch:~$ sudo ztp -r cumulus-ztp.sh
```

To see the current **ztp** state, use the **-s** option:

```
cumulus@switch:~$ sudo ztp -s
ZTP INFO:
State disabled
Version 1.0
Result success
Date Thu May 5 16:49:33 2016 UTC
```

Method Switch manually configured

URL None

## Notes

- During the development of a provisioning script, the switch may need to be reset.
- You can use the Cumulus Linux `onie-select -i` command to cause the switch to reprovision itself and install a network operating system again using ONIE.

# System Management

## Setting Date and Time

Setting the time zone, date and time requires root privileges; use `sudo`.

### Contents

This chapter covers ...

- Commands (see page 72)
- Setting the Time Zone (see page 72)
  - Alternative: Use the Guided Wizard to Find and Apply a Time Zone (see page 73)
- Setting the Date and Time (see page 73)
- Setting Time Using NTP (see page 74)
- Specifying the NTP Source Interface (see page 75)
- Configuration Files (see page 75)
- Useful Links (see page 75)

### Commands

- `date`
- `dpkg-reconfigure tzdata`
- `hwclock`
- `ntpd` (daemon)
- `ntpq`

### Setting the Time Zone

To see the current time zone, list the contents of `/etc/timezone`:

```
cumulus@switch:~$ cat /etc/timezone
US/Eastern
```

Edit the file to add your desired time zone. A list of valid time zones can be found at the following [link](#).

Use the following command to apply the new time zone immediately.

```
dpkg-reconfigure --frontend noninteractive tzdata
```

## Alternative: Use the Guided Wizard to Find and Apply a Time Zone

To set the time zone, run `dpkg-reconfigure tzdata` as root:

```
cumulus@switch:~$ sudo dpkg-reconfigure tzdata
```

Then navigate the menus to enable the time zone you want. The following example selects the US/Pacific time zone:

```
cumulus@switch:~$ sudo dpkg-reconfigure tzdata

Configuring tzdata
-----

Please select the geographic area in which you live. Subsequent
configuration
questions will narrow this down by presenting a list of cities, representing
the time zones in which they are located.

 1. Africa      4. Australia   7. Atlantic   10. Pacific   13. Etc
 2. America     5. Arctic       8. Europe     11. SystemV
 3. Antarctica  6. Asia        9. Indian     12. US

Geographic area: 12

Please select the city or region corresponding to your time zone.

 1. Alaska      4. Central     7. Indiana-Starke 10. Pacific
 2. Aleutian    5. Eastern     8. Michigan      11. Pacific-New
 3. Arizona     6. Hawaii      9. Mountain     12. Samoa

Time zone: 10

Current default time zone: 'US/Pacific'
Local time is now:      Mon Jun 17 09:27:45 PDT 2013.
Universal Time is now:  Mon Jun 17 16:27:45 UTC 2013.
```

For more info see the Debian [System Administrator's Manual – Time](#).

## Setting the Date and Time

The switch contains a battery backed hardware clock that maintains the time while the switch is powered off and in between reboots. When the switch is running, the Cumulus Linux operating system maintains its own software clock.

During boot up, the time from the hardware clock is copied into the operating system's software clock. The software clock is then used for all timekeeping responsibilities. During system shutdown the software clock is copied back to the battery backed hardware clock.

You can set the date and time on the software clock using the `date` command. First, determine your current time zone:

```
cumulus@switch$ date +%Z
```



If you need to reconfigure the current time zone, refer to the instructions above.

Then, to set the system clock according to the time zone configured:

```
cumulus@switch$ sudo date -s "Tue Jan 12 00:37:13 2016"
```

See `man date(1)` for if you need more information.

You can write the current value of the system (software) clock to the hardware clock using the `hwclock` command:

```
cumulus@switch$ sudo hwclock -w
```

See `man hwclock(8)` if you need more information.

You can find a good overview of the software and hardware clocks in the Debian [System Administrator's Manual – Time](#), specifically the section [Setting and showing hardware clock](#).

## Setting Time Using NTP

The `ntpd` daemon running on the switch implements the NTP protocol. It synchronizes the system time with time servers listed in `/etc/ntp.conf`. It is started at boot by default. See `man ntpd(8)` for `ntpd` details.

By default, `/etc/ntp.conf` contains some default time servers. Edit `/etc/ntp.conf` to add or update time server information. See `man ntp.conf(5)` for details on configuring `ntpd` using `ntp.conf`.

To set the initial date and time via NTP before starting the `ntpd` daemon, use `ntpd -q` (This is same as `ntpdate`, which is to be retired and not available).



`ntpd -q` can hang if the time servers are not reachable.

To verify that `ntpd` is running on the system:

```
cumulus@switch:~$ ps -ef | grep ntp
ntp      4074      1  0 Jun20 ?          00:00:33 /usr/sbin/ntpd -p /var/run
/ntpd.pid -g -u 101:102
```

To check the NTP peer status:

```
cumulus@switch:~$ ntpq -p      remote                  refid      st t when poll
reach   delay   offset   jitter
=====
==

*level1f.cs.unc. .PPS.           1 u    225 1024  377    92.505  -1.296
1.139
+ip.tcp.lv        193.11.166.8     2 u    29 1024  377   192.701   2.424
1.227
-host-86.3.217.2 131.107.13.100  2 u 1024 1024  367   240.622   11.250
7.785
+li290-38.member 128.138.141.172  2 u    553 1024  377   38.944  -0.810
1.139
```

## ***Specifying the NTP Source Interface***

You can change the source interface that NTP uses if you want to use something other than the default of eth0. Edit `ntp.conf` and edit the entry under the **# Specify interfaces** comment:

```
# Specify interfaces
interface listen bridge10
```

## ***Configuration Files***

- `/etc/default/ntp` — `ntpd init.d` configuration variables
- `/etc/ntp.conf` — default NTP configuration file

## ***Useful Links***

- [Debian System Administrator's Manual – Time](#)
- <http://www.ntp.org>
- [http://en.wikipedia.org/wiki/Network\\_Time\\_Protocol](http://en.wikipedia.org/wiki/Network_Time_Protocol)
- <http://wiki.debian.org/NTP>

# Authentication, Authorization, and Accounting

- SSH for Remote Access (see page 76)
- User Accounts (see page 77)
- Using sudo to Delegate Privileges (see page 79)
- PAM and NSS (see page 85)

## SSH for Remote Access

You use **SSH** to securely access a Cumulus Linux switch remotely.



By default, you cannot use the root account to SSH to a Cumulus Linux switch unless you generate an **SSH key** (see page 78) or **set a password** (see page 78) for the account.

## Contents

(Click to expand)

- Contents (see page 76)
- Access Using Passkey (Basic Setup) (see page 76)
  - Completely Passwordless System (see page 77)
- Useful Links (see page 77)

## Access Using Passkey (Basic Setup)

Cumulus Linux uses the openSSH package to provide SSH functionality. The standard mechanisms of generating passwordless access just applies. The example below has the *cumulus* user on a machine called *management-station* connecting to a switch called *cumulus-switch1*.

First, on *management-station*, generate the SSH keys:

```
cumulus@management-station:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/cumulus/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/cumulus/.ssh/id_rsa.
Your public key has been saved in /home/cumulus/.ssh/id_rsa.pub.
The key fingerprint is:
8c:47:6e:00:fb:13:b5:07:b4:1e:9d:f4:49:0a:77:a9 cumulus@management-
station
The key's randomart image is:
+--[ RSA 2048]----+
|    . .= o o.   |
|    o . o *..   |
```

```

|   . o = =.o
|   . O oE
|   + S
|   +
|
|   |
|   |
+-----+

```

Next, append the public key in `~/.ssh/id_rsa.pub` into `~/.ssh/authorized_keys` in the target user's home directory:

```

cumulus@management-station:~$ scp .ssh/id_rsa.pub cumulus@cumulus-switch1:.
ssh/authorized_keys
Enter passphrase for key '/home/cumulus/.ssh/id_rsa':
id_rsa.pub

```

## Completely Passwordless System

When generating the passphrase and its associated keys, as in the first step above, do not enter a passphrase. Follow all the other instructions.

## Useful Links

- <http://www.debian-administration.org/articles/152>

## User Accounts

By default, Cumulus Linux has two user accounts: *cumulus* and *root*.

The *cumulus* account:

- Default password is *CumulusLinux!*
- Is a user account in the *sudo* group with *sudo* privileges
- User can log in to the system via all the usual channels like console and *SSH* (see page 76)

The *root* account:

- Default password is disabled by default
- Has the standard Linux *root* user access to everything on the switch
- Disabled password prohibits login to the switch by *SSH*, *telnet*, *FTP*, and so forth

For best security, you should change the default password (using the ***passwd*** command) before you configure Cumulus Linux on the switch.

You can add more user accounts as needed. Like the *cumulus* account, these accounts must use ***sudo*** to execute privileged commands (see page 79), so be sure to include them in the *sudo* group.

To access the switch without any password requires booting into a single shell/user mode (see page 503).

## Enabling Remote Access for the root User

As mentioned above, the root user does not have a password set for it, and it cannot log in to a switch via SSH. This default account behavior is consistent with Debian. In order to connect to a switch using the root account, you can do one of two things for the account:

- Generate an SSH key
- Set a password

### Generating an SSH Key for the root Account

1. First, in a terminal on your host system (not the switch), check to see if a key already exists:

```
$ ls -al ~/.ssh/
```

The key is named something like `id_dsa.pub`, `id_rsa.pub` or `id_ecdsa.pub`.

2. If a key doesn't exist, generate a new one by first creating the RSA key pair:

```
$ ssh-keygen -t rsa
```

3. A prompt appears, asking you to *Enter file in which to save the key (/root/.ssh/id\_rsa)*:. Press Enter to use the root user's home directory, or else provide a different destination.
4. You are prompted to *Enter passphrase (empty for no passphrase)*:. This is optional but it does provide an extra layer of security.
5. The public key is now located in `/root/.ssh/id_rsa.pub`. The private key (identification) is now located in `/root/.ssh/id_rsa`.
6. Copy the public key to the switch. SSH to the switch as the cumulus user, then run:

```
cumulus@switch:~$ sudo mkdir -p /root/.ssh
cumulus@switch:~$ echo <SSH public key string> | sudo tee -a /root/.ssh
/authorized_keys
```

## Setting the root User Password

1. Run:

```
cumulus@switch:~$ sudo passwd root
```

2. Change the `/etc/ssh/sshd_config` file's `PermitRootLogin` setting from `without-password` to `yes`

```
cumulus@switch:~$ sudo vi /etc/ssh/sshd_config
```

```
...  
  
# Authentication:  
LoginGraceTime 120  
PermitRootLogin yes  
StrictModes yes  
  
...
```

3. Restart the `ssh` service:

```
cumulus@switch:~$ sudo systemctl reload ssh.service
```

## Using sudo to Delegate Privileges

By default, Cumulus Linux has two user accounts: `root` and `cumulus`. The `cumulus` account is a normal user and is in the group `sudo`.

You can add more user accounts as needed. Like the `cumulus` account, these accounts must use `sudo` to execute privileged commands.

## Contents

(Click to expand)

- [Contents \(see page 79\)](#)
- [Commands \(see page 79\)](#)
- [Using sudo \(see page 79\)](#)
- [sudoers Examples \(see page 80\)](#)
- [Configuration Files \(see page 85\)](#)
- [Useful Links \(see page 85\)](#)

## Commands

- `sudo`
- `visudo`

## Using sudo

`sudo` allows you to execute a command as superuser or another user as specified by the security policy. See `man sudo(8)` for details.

The default security policy is `sudoers`, which is configured using `/etc/sudoers`. Use `/etc/sudoers.d/` to add to the default sudoers policy. See `man sudoers(5)` for details.



Use **visudo** only to edit the **sudoers** file; do not use another editor like **vi** or **emacs**. See **man visudo(8)** for details.

Errors in the **sudoers** file can result in losing the ability to elevate privileges to root. You can fix this issue only by power cycling the switch and booting into single user mode. Before modifying **sudoers**, enable the root user by setting a password for the root user.

By default, users in the **sudo** group can use **sudo** to execute privileged commands. To add users to the **sudo** group, use the **useradd(8)** or **usermod(8)** command. To see which users belong to the **sudo** group, see **/etc/group (man group(5))**.

Any command can be run as **sudo**, including **su**. A password is required.

The example below shows how to use **sudo** as a non-privileged user *cumulus* to bring up an interface:

```
cumulus@switch:~$ ip link show dev swp1
3: swp1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master br0 state
DOWN mode DEFAULT qlen 500
link/ether 44:38:39:00:27:9f brd ff:ff:ff:ff:ff:ff

cumulus@switch:~$ ip link set dev swp1 up
RTNETLINK answers: Operation not permitted

cumulus@switch:~$ sudo ip link set dev swp1 up
Password:

cumulus@switch:~$ ip link show dev swp1
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br0 state UP mode DEFAULT qlen 500
link/ether 44:38:39:00:27:9f brd ff:ff:ff:ff:ff:ff
```

## ***sudoers Examples***

The following examples show how you grant as few privileges as necessary to a user or group of users to allow them to perform the required task. For each example, the system group *noc* is used; groups are prefixed with an %.

When executed by an unprivileged user, the example commands below must be prefixed with **sudo**.

Category	Privilege	Example Command	<b>sudoers</b> Entry
Monitoring	Switch port info	<code>ethtool -m swp1</code>	<code>%noc ALL=(ALL) NOPASSWD: /sbin/ethtool</code>

Category	Privilege	Example Command	sudoers Entry
Monitoring	System diagnostics	<code>cl-support</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/cumulus/bin/cl-support</code>
Monitoring	Routing diagnostics	<code>cl-resource-query</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/cumulus/bin/cl-resource-query</code>
Image management	Install images	<code>onie-select http://lab/install.bin</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/cumulus/bin/onie-select</code>
Package management	Any apt-get command	<code>apt-get update</code> or <code>apt-get install</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/bin/apt-get</code>
Package management	Just apt-get update	<code>apt-get update</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/bin/apt-get update</code>
Package management	Install packages	<code>apt-get install mtr-tiny</code>	<code>%noc ALL=(ALL) NOPASSWD:/usr/bin/apt-get install *</code>
Package management	Upgrading	<code>apt-get upgrade</code>	

Category	Privilege	Example Command	sudoers Entry
			<pre>%noc ALL=(ALL) NOPASSWD:/usr /bin/apt-get upgrade</pre>
Netfilter	Install ACL policies	<pre>cl-acltool -i</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /cumulus/bin/cl-acltool</pre>
Netfilter	List iptables rules	<pre>iptables -L</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/iptables</pre>
L1 + 2 features	Any LLDP command	<pre>lldpcli show neighbors / configure</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /sbin/lldpcli</pre>
L1 + 2 features	Just show neighbors	<pre>lldpcli show neighbors</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /sbin/lldpcli show neighbours*</pre>
Interfaces	Modify any interface	<pre>ip link set dev swp1 {up down}</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ip link set *</pre>
Interfaces	Up any interface	<pre>ifup swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ifup</pre>

Category	Privilege	Example Command	sudoers Entry
Interfaces	Down any interface	<pre>ifdown swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ifdown</pre>
Interfaces	Up/down only swp2	<pre>ifup swp2 / ifdown swp2</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ifup swp2,/sbin /ifdown swp2</pre>
Interfaces	Any IP address chg	<pre>ip addr {add del} 192.0.2.1/30 dev swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ip addr *</pre>
Interfaces	Only set IP address	<pre>ip addr add 192.0.2.1/30 dev swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/ip addr add *</pre>
Ethernet bridging	Any bridge command	<pre>brctl addbr br0 / brctl delif br0 swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/brctl</pre>
Ethernet bridging	Add bridges and ints	<pre>brctl addbr br0 / brctl addif br0 swp1</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/brctl addbr *,/sbin /brctl addif *</pre>

Category	Privilege	Example Command	sudoers Entry
Spanning tree	Set STP properties	<pre>mstpctl setmaxage br2 20</pre>	<pre>%noc ALL=(ALL) NOPASSWD: /sbin/mstpctl</pre>
Troubleshooting	Restart switchd	<pre>systemctl restart switchd. service</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /sbin/service switchd *</pre>
Troubleshooting	Restart any service	<pre>systemctl cron switchd.service</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /sbin/service</pre>
Troubleshooting	Packet capture	<pre>tcpdump</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /sbin/tcpdump</pre>
L3	Add static routes	<pre>ip route add 10.2.0.0/16 via 10.0.0.1</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/bin /ip route add *</pre>
L3	Delete static routes	<pre>ip route del 10.2.0.0/16 via 10.0.0.1</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/bin /ip route del *</pre>
L3	Any static route chg	<pre>ip route *</pre>	

Category	Privilege	Example Command	sudoers Entry
			<pre>%noc ALL=(ALL) NOPASSWD:/bin /ip route *</pre>
L3	Any iproute command	<pre>ip *</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/bin /ip</pre>
L3	Non- modal OSPF	<pre>cl-ospf area 0.0.0.1 range 10.0.0.0/24</pre>	<pre>%noc ALL=(ALL) NOPASSWD:/usr /bin/cl-ospf</pre>

## Configuration Files

- /etc/sudoers - default security policy
- /etc/sudoers.d/ - default security policy

## Useful Links

- sudo
- Adding Yourself to sudoers

## LDAP Authentication and Authorization

Cumulus Linux uses Pluggable Authentication Modules (PAM) and Name Service Switch (NSS) for user authentication.

NSS specifies the order of information sources used to resolve names for each service. Using this with authentication and authorization, it provides the order and location used for user lookup and group mapping on the system. PAM handles the interaction between the user and the system, providing login handling, session setup, authentication of users and authorization of a user actions.

NSS enables PAM to use LDAP for providing user authentication, group mapping and information for other services on the system.

## Contents

This chapter covers ...

- Configuring LDAP Authentication (see page 86)
- Installing libnss-ldapd (see page 86)
- Configuring nslcd.conf (see page 87)
  - Connection (see page 88)
  - Search Function (see page 88)
  - Search Filters (see page 88)
  - Attribute Mapping (see page 89)
  - Example Configuration (see page 89)
- Troubleshooting (see page 89)
  - Using nslcd Debug Mode (see page 89)
  - Common Problems (see page 91)
- Configuring LDAP Authorization (see page 92)
- Active Directory Configuration (see page 92)
- LDAP Verification Tools (see page 93)
  - Identifying a User with the id Command (see page 93)
  - Using getent (see page 93)
  - Using LDAP search (see page 94)
  - LDAP Browsers (see page 95)
- References (see page 95)

## Configuring LDAP Authentication

There are 3 common ways of configuring LDAP authentication on Linux:

- libnss-ldap
- libnss-ldapd
- libnss-sss

This chapter covers using **libnss-ldapd** only. From internal testing, this library worked best with Cumulus Linux and was the easiest to configure, automate and troubleshoot.

## Installing libnss-ldapd



The **libnss-ldapd** and **ldap-utils** packages are not available in the Cumulus Networks repository. You must install them from the Debian repository. You need to configure the switch to reference the Debian repository. To do so, edit the **/etc/apt/sources.list** file and adding the following line:

```
deb http://ftp.us.debian.org/debian/ jessie main
```

If *nested group support* is required, **libnss-ldapd** must be version 0.9 or higher. For Cumulus Linux 3.x, you should add the **jessie-backports** repo instead of the Jessie repo:

```
deb http://ftp.us.debian.org/debian/ jessie-backports main
```

Then run `apt-get update` to sync with the Debian repo.

Once you reference the Debian repository, install `libnss-ldapd`, `libpam-ldapd` and `ldap-utils`. Run:

```
cumulus@switch:~$ sudo apt-get install libnss-ldapd libpam-ldapd ldap-utils
```

This brings up an interactive prompt asking questions about the LDAP URI, search base distinguished name (DN) and services that should have LDAP lookups enabled. This creates a very basic LDAP configuration, using anonymous bind, and initiating the search for a user under the base DN specified.



Alternatively, these parameters can be pre-seeded using the `debconf-utils`. To use this method, run `apt-get install debconf-utils` and create the pre-seeded parameters using `debconf-set-selections` with the appropriate answers. Run `debconf-show <pkg>` to check the settings. Here is an [example of how to preseed answers to the installer questions using debconf-set-selections](#).

Once the install is complete, the *name service LDAP caching daemon* (`nslcd`) will be running. This is the service that handles all of the LDAP protocol interactions, and caches the information returned from the LDAP server. In `/etc/nsswitch.conf`, `ldap` has been appended and is the secondary information source for `passwd`, `group` and `shadow`. The local files (`/etc/passwd`, `/etc/groups` and `/etc/shadow`) are used first, as specified by the `compat` source.

```
passwd: compat ldap
group: compat ldap
shadow: compat ldap
```



You are strongly advised to keep `compat` as the first source in NSS for `passwd`, `group` and `shadow`. This prevents you from getting locked out of the system.

## Configuring `nslcd.conf`

You need to update the main configuration file (`/etc/nslcd.conf`) after installation to accommodate the expected LDAP server settings. The [nslcd.conf man page](#) details all the available configuration options. Some of the more important options are related to security and how the queries are handled.

## Connection

The LDAP client starts a session by connecting to the LDAP server, by default, on TCP and UDP port 389, or on port 636 for LDAPS. Depending on the configuration, this connection may be unauthenticated (anonymous bind); otherwise, the client must provide a bind user and password. The variables used to define the connection to the LDAP server are the URI and bind credentials.

The URI is mandatory, and specifies the LDAP server location using the FQDN or IP address. It also designates whether to use `ldap://` for clear text transport, or `ldaps://` for SSL/TLS encrypted transport. Optionally, an alternate port may also be specified in the URI. Typically, in production environments, it is best to utilize the LDAPS protocol. Otherwise all communications are clear text and not secure.

After the connection to the server is complete, the BIND operation authenticates the session. The BIND credentials are optional, and if not specified, an anonymous bind is assumed. This is typically not allowed in most production environments. Configure authenticated (Simple) BIND by specifying the user (`binddn`) and password (`bindpw`) in the configuration. Another option is to use SASL (Simple Authentication and Security Layer) BIND, which provides authentication services using other mechanisms, like Kerberos. Contact your LDAP server administrator for this information since it depends on the configuration of the LDAP server and what credentials are created for the client device.

```
# The location at which the LDAP server(s) should be reachable.
uri ldaps://ldap.example.com
# The DN to bind with for normal lookups.
binddn cn=CLswitch,ou=infra,dc=example,dc=com
bindpw CuMuLuS
```

## Search Function

When an LDAP client requests information about a resource, it must connect and bind to the server. Then it performs one or more resource queries depending on what it is looking up. All search queries sent to the LDAP server are created using the configured search `base`, `filter`, and the desired entry (`uid=myuser`) being searched for. If the LDAP directory is large, this search may take a significant amount of time. It is a good idea to define a more specific search base for the common `maps` (`passwd` and `group`).

```
# The search base that will be used for all queries.
base dc=example,dc=com
# Mapped search bases to speed up common queries.
base passwd ou=people,dc=example,dc=com
base group ou=groups,dc=example,dc=com
```

## Search Filters

It is also common to use search filters to specify criteria used when searching for objects within the directory. This is used to limit the search scope when authenticating users. The default filters applied are:

```
filter passwd (objectClass=posixAccount)
filter group (objectClass=posixGroup)
```

## Attribute Mapping

The *map* configuration allows for overriding the attributes pushed from LDAP. To override an attribute for a given *map*\*<sup>\*</sup>, specify the attribute name and the new value. One example of how this is useful is ensuring the shell is *bash* and the home directory is */home/cumulus*:

```
map      passwd homeDirectory "/home/cumulus"
map      passwd shell "/bin/bash"
```



\*In LDAP, the **map** refers to one of the supported maps specified in the manpage for **nslcd.conf** (such as *passwd* or *group*).

## Example Configuration

Here is an [example configuration](#) using Cumulus Linux.

## Troubleshooting

### Using nsLCD Debug Mode

When setting up LDAP authentication for the first time, Cumulus Networks recommends you turn off this service using `sudo systemctl stop nsLCD.service` and run it in debug mode. Debug mode works whether you are using LDAP over SSL (port 636) or an unencrypted LDAP connection (port 389).

```
cumulus@switch:~$ sudo systemctl stop nsLCD.service
cumulus@switch:~$ sudo nsLCD -d
```

Once you enable debug mode, run the following command to test LDAP queries:

```
cumulus@switch:~$ sudo getent myuser
```

If LDAP is configured correctly, the following messages appear after you run the `getent` command:

```

nslcd: DEBUG: accept() failed (ignored): Resource temporarily unavailable
nslcd: [8elf29] DEBUG: connection from pid=11766 uid=0 gid=0
nslcd: [8elf29] <passwd(all)> DEBUG: myldap_search(base="dc=example,
dc=com", filter="(objectClass=posixAccount)")
nslcd: [8elf29] <passwd(all)> DEBUG: ldap_result(): uid=myuser,ou=people,
dc=example,dc=com
nslcd: [8elf29] <passwd(all)> DEBUG: ldap_result(): ... 152 more results
nslcd: [8elf29] <passwd(all)> DEBUG: ldap_result(): end of results (162
total)
  
```

In the output above, `<passwd(all)>` indicates that the entire directory structure was queried.

A specific user can be queried using the command:

```
cumulus@switch:~$ sudo getent passwd myuser
```

You can replace `myuser` with any username on the switch. The following debug output indicates that user `myuser` exists:

```

nslcd: DEBUG: add_uri(ldap://10.50.21.101)
nslcd: version 0.8.10 starting
nslcd: DEBUG: unlink() of /var/run/nslcd/socket failed (ignored): No such
file or directory
nslcd: DEBUG: setgroups(0,NULL) done
nslcd: DEBUG: setgid(110) done
nslcd: DEBUG: setuid(107) done
nslcd: accepting connections
nslcd: DEBUG: accept() failed (ignored): Resource temporarily unavailable
nslcd: [8b4567] DEBUG: connection from pid=11369 uid=0 gid=0
nslcd: [8b4567] <passwd="myuser"> DEBUG: myldap_search(base="
dc=cumulusnetworks,dc=com", filter="(&(objectClass=posixAccount)
(uid=myuser))")
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_initialize
(ldap://<ip_address>)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_rebind_proc()
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option
(LDAP_OPT_PROTOCOL_VERSION,3)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option(LDAP_OPT_DEREF,0)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option(LDAP_OPT_TIMELIMIT,
0)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option(LDAP_OPT_TIMEOUT,0)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option
(LDAP_OPT_NETWORK_TIMEOUT,0)
  
```

```

nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option(LDAP_OPT_REFERRALS,
LDAP_OPT_ON)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_set_option(LDAP_OPT_RESTART,
LDAP_OPT_ON)
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_simple_bind_s(NULL,NULL)
(uri="ldap://<ip_address>")
nslcd: [8b4567] <passwd="myuser"> DEBUG: ldap_result(): end of results (0
total)

```

Notice how the <passwd="myuser"> shows that the specific *myuser* user was queried.

## Common Problems

### SSL/TLS

- The FQDN of the LDAP server URI does not match the FQDN in the CA-signed server certificate exactly.
- **nslcd** cannot read the SSL certificate, and will report a "Permission denied" error in the debug during server connection negotiation. Check the permission on each directory in the path of the root SSL certificate. Ensure that it is readable by the **nslcd** user.

### NSCD

- If the **nscd cache** daemon is also enabled and you make some changes to the user from LDAP, you may want to clear the cache using the commands:

```

nscd --invalidate = passwd
nscd --invalidate = group

```

- The **nscd** package works with **nslcd** to cache name entries returned from the LDAP server. This may cause authentication failures. To work around these issues:
  1. Disable **nscd** by running:

```
cumulus@switch:~$ sudo nscd -K
```

2. Restart the **nslcd** service:

```
cumulus@switch:~$ sudo systemctl restart nslcd.service
```

3. Try the authentication again.

## LDAP

- The search filter returns wrong results. Check for typos in the search filter. Use `ldapsearch` to test your filter.
- Optionally, configure the basic LDAP connection and search parameters in `/etc/ldap/ldap.conf`

```
# ldapsearch -D 'cn=CLadmin' -w 'CuMuLuS' "(&
(ObjectClass=inetOrgUser)(uid=myuser))"
```

- When a local username also exists in the LDAP database, the order of the information sources in `/etc/nsswitch` can be updated to query LDAP before the local user database. This is generally not recommended. For example, the configuration below ensures that LDAP is queried before the local database.

```
# /etc/nsswitch.conf
passwd:      ldap  compat
```

## Configuring LDAP Authorization

Linux uses the `sudo` command to allow non-administrator users — like the default *cumulus* user account — to perform privileged operations. To control the users authorized to use sudo, the `/etc/sudoers` file and files located in the `/etc/sudoers.d/` directory have a series of rules defined. Typically, the rules are based on groups, but can also be defined for specific users. Therefore, sudo rules can be added using the group names from LDAP. For example, if a group of users were associated with the group *netadmin*, a rule can be added to give those users sudo privileges. Refer to the sudoers manual (`man sudoers`) for a complete usage description. Here's an illustration of this in `/etc/sudoers`:

```
# 
The basic structure of a user specification is "who where = (as_whom)
what".
%sudo ALL=(ALL:ALL) ALL
%netadmin ALL=(ALL:ALL) ALL
```

## Active Directory Configuration

Active Directory (AD) is a fully featured LDAP-based NIS server created by Microsoft. It offers unique features that classic OpenLDAP servers lack. Therefore, it can be more complicated to configure on the client and each version of AD is a little different in how it works with Linux-based LDAP clients. Some more advanced configuration examples, from testing LDAP clients on Cumulus Linux with Active Directory (AD/LDAP), are available in our [knowledge base](#).

## LDAP Verification Tools

Typically, password and group information is retrieved from LDAP and cached by the LDAP client daemon. To test the LDAP interaction, these command line tools can be used to trigger an LDAP query from the device. This helps to create the best filters and verify the information sent back from the LDAP server.

### Identifying a User with the id Command

The `id` command performs a username lookup by following the lookup information sources in NSS for the `passwd` service. This simply returns the user ID, group ID and the group list retrieved from the information source. In the following example, the user `cumulus` is locally defined in `/etc/passwd`, and `myuser` is on LDAP. The NSS configuration has the `passwd` map configured with the sources `compat ldap`:

```
cumulus@switch:~$ id cumulus
uid=1000(cumulus) gid=1000(cumulus) groups=1000(cumulus),4(adm),27(sudo)
cumulus@switch:~$ id myuser
uid=1230(myuser) gid=3000(Development) groups=3000(Development),500
(Employees),27(sudo)
```

### Using getent

The `getent` command retrieves all records found via NSS for a given map. It can also get a specific entry under that map. Tests can be done with the `passwd`, `group`, `shadow` or any other map configured in `/etc/nsswitch.conf`. The output from this command is formatted according to the map requested. Thus, for the `passwd` service, the structure of the output is the same as the entries in `/etc/passwd`. The same can be said for the `group` map will output the same as `/etc/group`. In this example, looking up a specific user in the `passwd` map, the user `cumulus` is locally defined in `/etc/passwd`, and `myuser` is only in LDAP.

```
cumulus@switch:~$ getent passwd cumulus
cumulus:x:1000:1000::/home/cumulus:/bin/bash
cumulus@switch:~$ getent passwd myuser
myuser:x:1230:3000:My Test User:/home/myuser:/bin/bash
```

In the next example, looking up a specific group in the `group` service, the group `cumulus` is locally defined in `/etc/groups`, and `netadmin` is on LDAP.

```
cumulus@switch:~$ getent group cumulus
cumulus:x:1000:
cumulus@switch:~$ getent group netadmin
netadmin:*:502:matthew,mark,luke,john
```

Running the command `getent passwd` or `getent group` without a specific request, returns **all** local and LDAP entries for the `passwd` and `group` maps, respectively.

## Using LDAP search

The `ldapsearch` command performs LDAP operations directly on the LDAP server. This does not interact with NSS. This command helps display what the LDAP daemon process is receiving back from the server. The command has many options. The simplest uses anonymous bind to the host and specifies the search DN and what attribute to lookup.

```
cumulus@switch:~$ ldapsearch -H ldap://ldap.example.com -b dc=example,  
dc=com -x uid=myuser
```

Click here to expand output of command

```
# extended LDIF  
#  
# LDAPv3  
# base <dc=example,dc=com> with scope subtree  
# filter: uid=myuser  
# requesting: ALL  
#  
# myuser, people, example.com  
dn: uid=myuser,ou=people,dc=example,dc=com  
cn: My User  
displayName: My User  
gecos: myuser  
gidNumber: 3000  
givenName: My  
homeDirectory: /home/myuser  
initials: MU  
loginShell: /bin/bash  
mail: myuser@example.com  
objectClass: inetOrgPerson  
objectClass: posixAccount  
objectClass: shadowAccount  
objectClass: top  
shadowExpire: -1  
shadowFlag: 0  
shadowMax: 999999  
shadowMin: 8  
shadowWarning: 7  
sn: User
```

```
uid: myuser
uidNumber: 1234

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

## LDAP Browsers

There are some GUI LDAP clients that help to work with LDAP servers. These are free tools to help graphically show the structure of the LDAP database.

- [Apache Directory Studio](#)
- [LDAPManager](#)

## References

- <https://wiki.debian.org/LDAP/PAM>
- <https://raw.githubusercontent.com/arthurdejong/nss-pam-ldapd/master/nsldap.conf>
- <http://backports.debian.org/Instructions/>

## Netfilter - ACLs

[Netfilter](#) is the packet filtering framework in Cumulus Linux as well as most other Linux distributions. [iptables](#), [ip6tables](#) and [ebtables](#) are userspace tools in Linux to administer filtering rules for IPv4 packets, IPv6 packets and Ethernet frames (layer 2 using MAC addresses) respectively. [cl-acltool](#) is the userspace tool to administer filtering rules on Cumulus Linux, and is the only tool for configuring ACLs in Cumulus Linux.

[cl-acltool](#) operates on a series of configuration files, and uses [iptables](#), [ip6tables](#) and [ebtables](#) to install rules into the kernel. In addition to programming rules in the kernel, [cl-acltool](#) programs rules in hardware for interfaces involving switch port interfaces, which [iptables](#), [ip6tables](#) and [ebtables](#) cannot do on their own.

## Contents

(Click to expand)

- [Commands \(see page 97\)](#)
- [Files \(see page 97\)](#)
- [Understanding Traffic Rules In Cumulus Linux \(see page 97\)
  - \[Understanding Chains \\(see page 97\\)\]\(#\)
  - \[Understanding Tables \\(see page 98\\)\]\(#\)
  - \[Understanding Rules \\(see page 100\\)\]\(#\)](#)

- How Rules Are Parsed and Applied (see page 101)
- Rule Placement in Memory (see page 102)
- Enabling Nonatomic Updates (see page 102)
- Using iptables/ip6tables/ebtables Directly (see page 103)
- Estimating the Number of Rules (see page 104)
- Installing and Managing ACL Rules with cl-acltool (see page 104)
  - Installing Packet Filtering (ACL) Rules (see page 105)
  - Specifying which Policy Files to Install (see page 107)
  - Hardware Limitations on Number of Rules (see page 108)
- Supported Rule Types (see page 109)
  - iptables/ip6tables Rule Support (see page 110)
  - ebttables Rule Support (see page 111)
  - Other Unsupported Rules (see page 111)
- Common Examples (see page 112)
  - Policing Control Plane and Data Plane Traffic (see page 112)
  - Setting DSCP on Transit Traffic (see page 113)
  - Verifying DSCP Values on Transit Traffic (see page 114)
  - Checking the Packet and Byte Counters for ACL Rules (see page 114)
  - Filtering Specific TCP Flags (see page 116)
- Example Scenario (see page 116)
  - Switch 1 Configuration (see page 117)
  - Switch 2 Configuration (see page 118)
  - Egress Rule (see page 118)
  - Ingress Rule (see page 118)
  - Input Rule (see page 119)
  - Output Rule (see page 119)
  - Combined Rules (see page 119)
  - Layer 2-only Rules/ebtables (see page 119)
  - Useful Links (see page 120)
- Caveats and Errata (see page 120)
  - Not All Rules Supported (see page 120)
  - Log Actions Cannot Be Forwarded (see page 120)
  - Tomahawk Hardware Limitations (see page 120)
  - iptables Interactions with cl-acltool (see page 120)
  - Where to Assign Rules (see page 121)
  - Generic Error Message Displayed after ACL Rule Installation Failure (see page 121)

## Commands

- cl-acltool
- ebtables
- iptables
- ip6tables

## Files

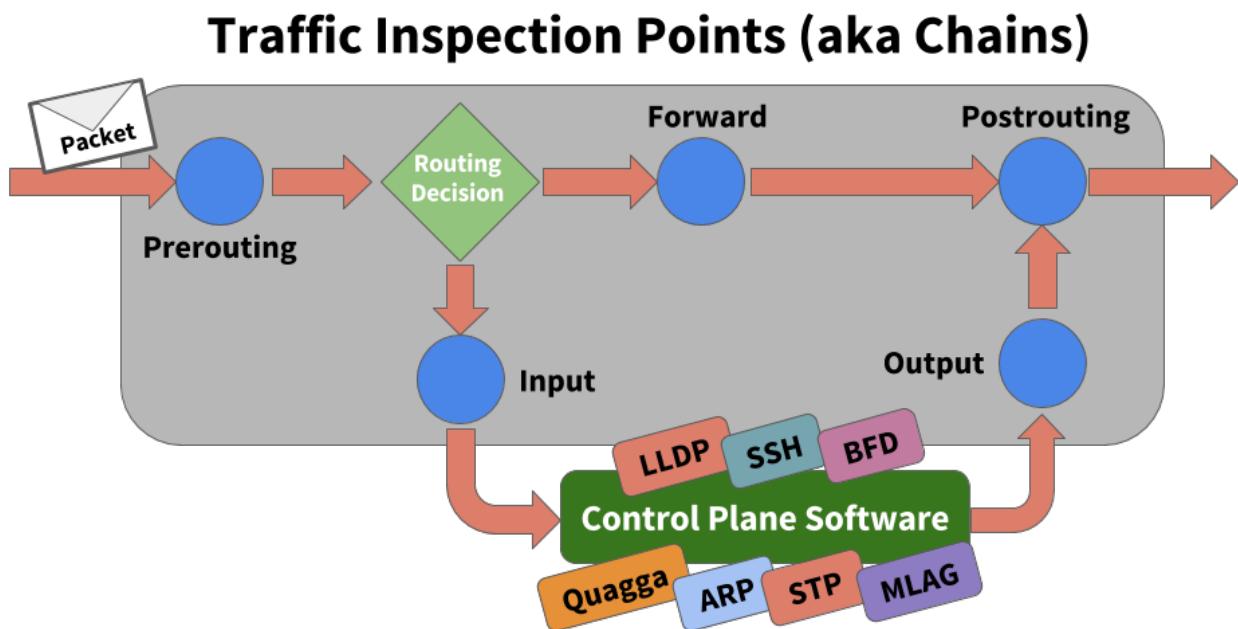
- /etc/cumulus/acl/policy.conf
- /etc/cumulus/acl/policy.d/

## Understanding Traffic Rules In Cumulus Linux

### Understanding Chains

Netfilter describes the mechanism for which packets are classified and controlled in the Linux kernel. Cumulus Linux uses the Netfilter framework to control the flow of traffic to, from and across the switch. Netfilter does not require a separate software daemon to run because it is part of the Linux kernel itself. Netfilter asserts policies at layers 2, 3 and 4 of the [OSI model](#) by inspecting packet and frame headers based on a list of rules. Rules are defined using syntax provided by the `iptables`, `ip6tables` and `ebtables` userspace applications.

The rules created by these programs inspect or operate on packets at several points in the life of the packet through the system. These five points are known as *chains* and are shown here:



The chains and their uses are:

- **PREROUTING:** Touches packets before they are routed
- **INPUT:** Touches packets once they are determined to be destined for the local system but before they are received by the control plane software
- **FORWARD:** Touches transit traffic as it moves through the box
- **OUTPUT:** Touches packets that are sourced by the control plane software before they are put on the wire
- **POSTROUTING:** Touches packets immediately before they are put on the wire but after the routing decision has been made

## ***Understanding Tables***

When building rules to affect the flow of traffic, the individual chains can be accessed by *tables*. Linux provides three tables by default:

- **Filter:** Classifies traffic or filters traffic
- **NAT:** Applies Network Address Translation rules

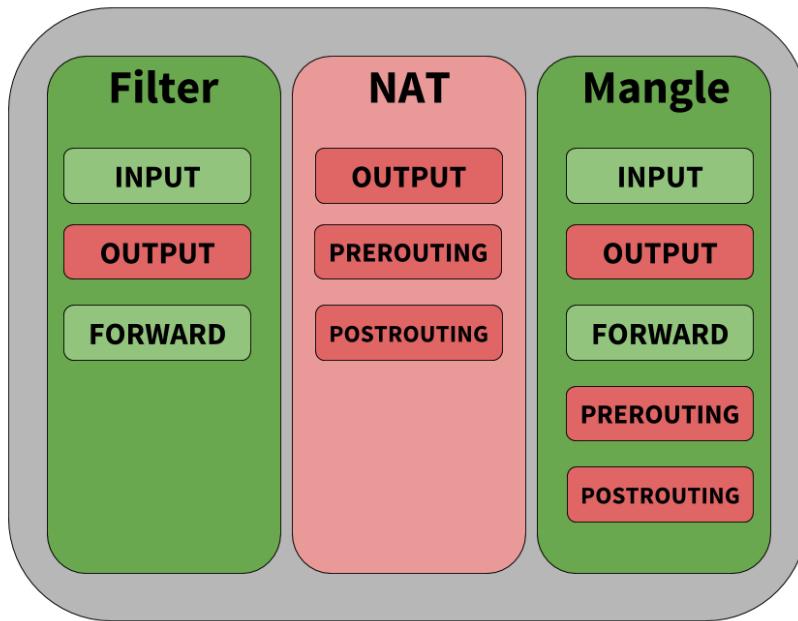


Cumulus Linux does not support NAT.

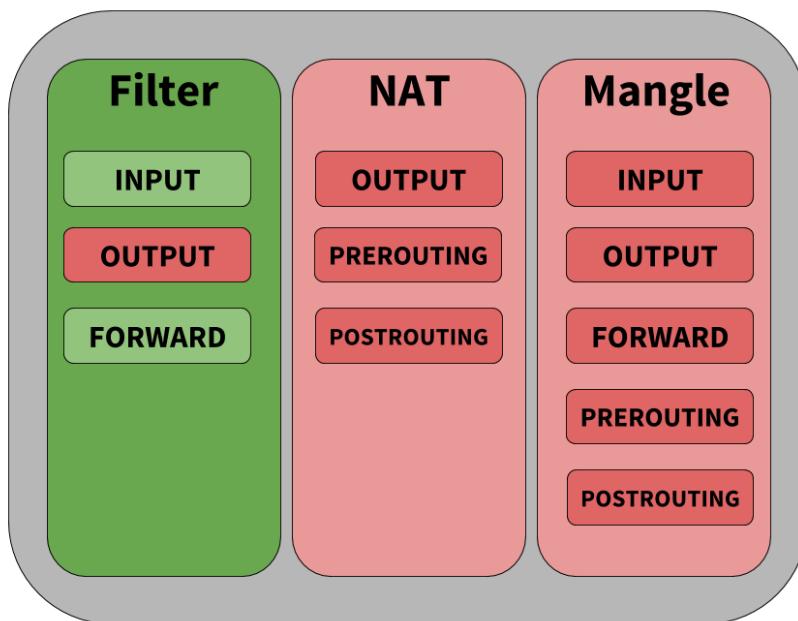
- **Mangle:** Alters packets as they move through the switch

Each table has a set of default chains that can be used to modify or inspect packets at different points of the path through the switch. Chains contain the individual rules to influence traffic. Each table and the default chains they support are shown below. Tables and chains in green are supported by Cumulus Linux, those in red are not supported (that is, they are not hardware accelerated) at this time.

## IPtables/IP6tables Table Support



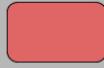
## EBtables Table Support



## Legend



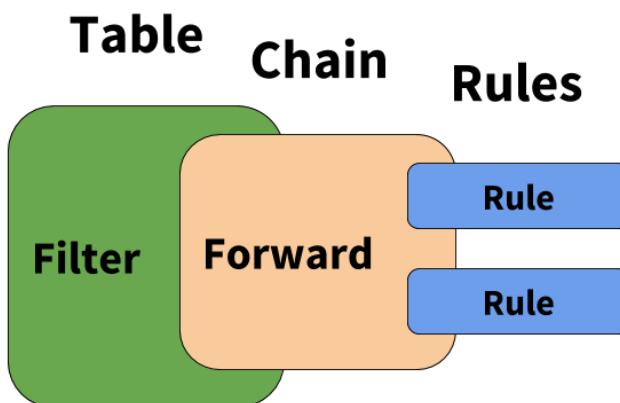
**Hardware Accelerated**



**NOT Hardware Accelerated**

### Understanding Rules

Rules are the items that actually classify traffic to be acted upon. Rules are applied to chains, which are attached to tables, similar to the graphic below.



Rules have several different components; the examples below highlight those different components.

(Sets SSH as high priority traffic)

-t mangle	-A FORWARD	-p tcp --dport 22	-j	DSCP --set-dscp 46
-----------	------------	-------------------	----	--------------------

Table	Chain	Matches	Jump	Targets
-------	-------	---------	------	---------

	-A INPUT	-i swp1 -p tcp --dport bgp	-j	POLICE --set-mode pkt
				--set-rate 2000 --set-burst 2000 --set-class 7

(Police and Prioritize BGP Traffic)

- **Table:** The first argument is the *table*. Notice the second example does not specify a table, that is because the filter table is implied if a table is not specified.
- **Chain:** The second argument is the *chain*. Each table supports several different chains. See Understanding Tables above.

- **Matches:** The third argument(s) are called the *matches*. You can specify multiple matches in a single rule. However, the more matches you use in a rule, the more memory that rule consumes.
- **Jump:** The *jump* specifies the target of the rule; that is, what action to take if the packet matches the rule. If this option is omitted in a rule, then matching the rule will have no effect on the packet's fate, but the counters on the rule will be incremented.
- **Target(s):** The *target* can be a user-defined chain (other than the one this rule is in), one of the special built-in targets that decides the fate of the packet immediately (like `DROP`), or an extended target. See the [Supported Rule Types and Common Usages](#) (see page 109) section below for examples of different targets.

## How Rules Are Parsed and Applied

All the rules from each chain are read from `iptables`, `ip6tables` and `ebtables` and entered in order into either the filter table or the mangle table. The rules are read from the kernel in the following order:

- IPv6 (`ip6tables`)
- IPv4 (`iptables`)
- `ebtables`

When rules are combined and put into one table, the order determines the relative priority of the rules; `iptables` and `ip6tables` have the highest precedence and `ebtables` has the lowest.

The Linux packet forwarding construct is an overlay for how the silicon underneath processes packets; to that end, here are some things to be aware of:

- The order of operations for how rules are processed is not perfectly maintained when you compare how `iptables` and the switch silicon process packets. The switch silicon reorders rules when `switchd` writes to the ASIC, whereas traditional `iptables` executes the list of rules in order.
- When processing traffic, rules affecting the FORWARD chain that specify an ingress interface are performed prior to rules that match on an egress interface. As a workaround, rules that only affect the egress interface can have an ingress interface wildcard (currently, only `swp+` and `bond+` are supported as wildcard names; see below) that matches any interface applied so that you can maintain order of operations with other input interface rules. Take the following rules, for example:

```
-A FORWARD -i $PORTA -j ACCEPT
-A FORWARD -o $PORTA -j ACCEPT    <-- This rule is performed LAST
(because of egress interface matching)
-A FORWARD -i $PORTB -j DROP
```

If you modify the rules like this, they are performed in order:

```
-A FORWARD -i $PORTA -j ACCEPT
-A FORWARD -i swp+ -o $PORTA -j ACCEPT    <-- These rules are
performed in order (because of wildcard match on ingress interface)
-A FORWARD -i $PORTB -j DROP
```

- When using rules that do a mangle and a filter lookup for a packet, Cumulus Linux does them in parallel and combines the action.

- If a switch port is assigned to a bond, any egress rules must be assigned to the bond.
- When using the OUTPUT chain, rules must be assigned to the source. For example, if a rule is assigned to the switch port in the direction of traffic but the source is a bridge (VLAN), the traffic won't be affected by the rule and must be applied to the bridge.
- If all transit traffic needs to have a rule applied, use the FORWARD chain, not the OUTPUT chain.
- **ebtables** rules are put into either the IPv4 or IPv6 memory space depending on whether the rule utilizes IPv4 or IPv6 to make a decision. Layer 2-only rules, which match the MAC address, are put into the IPv4 memory space.

## Rule Placement in Memory

INPUT and ingress (**FORWARD -i**) rules occupy the same memory space. A rule counts as ingress if the **-i** option is set. If both input and output options (**-i** and **-o**) are set, the rule is considered as ingress and occupies that memory space. For example:

```
-A FORWARD -i swp1 -o swp2 -s 10.0.14.2 -d 10.0.15.8 -p tcp -j ACCEPT
```



If you set an output flag with the INPUT chain you will get an error. For example, running **cl-acltool -i** on the following rule:

```
-A FORWARD,INPUT -i swp1 -o swp2 -s 10.0.14.2 -d 10.0.15.8 -p tcp -j ACCEPT
```

generates the following error:

```
error: line 2 : output interface specified with INPUT chain error
processing rule '-A FORWARD,INPUT -i swp1 -o swp2 -s 10.0.14.2 -d
10.0.15.8 -p tcp -j ACCEPT'
```

However, simply removing the **-o** option and interface would make it a valid rule.

## Enabling Nonatomic Updates

You can enable nonatomic updates for **switchd**, which offer better scaling because all hardware resources are used to actively impact traffic. With atomic updates, half of the hardware resources are on standby and do not actively impact traffic.

To always start **switchd** with nonatomic updates:

1. Edit **/etc/cumulus/switchd.conf**.
2. Add the following line to the file:

```
acl.non_atomic_update_mode = TRUE
```

3. Restart `switchd`:

```
cumulus@switch:~$ sudo systemctl restart switchd.service
```



During nonatomic updates, traffic is stopped first, and enabled after the new configuration is written into the hardware completely.

## **Using `iptables/ip6tables/ebtables` Directly**

Using `iptables/ip6tables/ebtables` directly is not recommended because any rules installed in these cases only are applied to the Linux kernel and are not hardware accelerated via synchronization to the switch silicon. Also running `cl-acltool -i` (the installation command) resets all rules and deletes anything that is not stored in `/etc/cumulus/acl/policy.conf`.

For example, performing:

```
cumulus@switch:~$ sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

Appears to work, and the rule appears when you run `cl-acltool -L`:

```
cumulus@switch:~$ sudo cl-acltool -L ip
-----
Listing rules of type iptables:
-----
TABLE filter :
Chain INPUT (policy ACCEPT 72 packets, 5236 bytes)
 pkts bytes target prot in out source destination
 0 0 DROP icmp -- any any anywhere anywhere icmp echo-request
```

However, the rule is not synced to hardware when applied in this fashion and running `cl-acltool -i` or `reboot` removes the rule without replacing it. To ensure all rules that can be in hardware are hardware accelerated, place them in `/etc/cumulus/acl/policy.conf` and install them by running `cl-acltool -i`.

## Estimating the Number of Rules

To estimate the number of rules that could be created from an ACL entry, first determine if that entry is an ingress or an egress. Then determine if it is IPv4-mac or IPv6 type rule. This determines the slice to which the rule belongs. Then use the following to determine how many entries are used up for each type.

By default, each entry occupies one double wide entry, except if the entry is one of the following:

- An entry with multiple comma-separated input interfaces is split into one rule for each input interface (listed after `--in-interface` below). For example, this entry splits into two rules:

```
-A FORWARD --in-interface swp1s0,swp1s1 -p icmp -j ACCEPT
```

- An entry with multiple comma-separated output interfaces is split into one rule for each output interface (listed after `--out-interface` below). this entry splits into two rules:

```
-A FORWARD --in-interface swp+ --out-interface swp1s0,swp1s1 -p  
icmp -j ACCEPT
```

- An entry with both input and output comma-separated interfaces is split into one rule for each combination of input and output interface (listed after `--in-interface` and `--out-interface` below). For example, this entry splits into four rules:

```
-A FORWARD --in-interface swp1s0,swp1s1 --out-interface swp1s2,  
swp1s3 -p icmp -j ACCEPT
```

- An entry with multiple L4 port ranges is split into one rule for each range (listed after `--dports` below). For example, this entry splits into two rules:

```
-A FORWARD --in-interface swp+ -p tcp -m multiport --dports 1050:  
1051,1055:1056 -j ACCEPT
```

## Installing and Managing ACL Rules with `cl-acltool`

You manage Cumulus Linux ACLs with `cl-acltool`. Rules are first written to the `iptables` chains, as described above, and then synced to hardware via `switchd`.

To examine the current state of chains and list all installed rules, run:

```
cumulus@switch:~$ sudo cl-acltool -L all  
-----  
Listing rules of type iptables:
```

```
-----  
TABLE filter :  
Chain INPUT (policy ACCEPT 90 packets, 14456 bytes)  
pkts bytes target prot opt in out source destination  
0 0 DROP all -- swp+ any 240.0.0.0/5 anywhere  
0 0 DROP all -- swp+ any loopback/8 anywhere  
0 0 DROP all -- swp+ any base-address.mcast.net/8 anywhere  
0 0 DROP all -- swp+ any 255.255.255.255 anywhere ...
```

To list installed rules using native `iptables`, `ip6tables` and `ebtables`, run these commands:

```
cumulus@switch:~$ sudo iptables -L  
cumulus@switch:~$ sudo ip6tables -L  
cumulus@switch:~$ sudo ebtables -L
```

To flush all installed rules, run:

```
cumulus@switch:~$ sudo cl-acltool -F all
```

To flush only the IPv4 `iptables` rules, run:

```
cumulus@switch:~$ sudo cl-acltool -F ip
```

If the install fails, ACL rules in the kernel and hardware are rolled back to the previous state. Errors from programming rules in the kernel or ASIC are reported appropriately.

## Installing Packet Filtering (ACL) Rules

`cl-acltool` takes access control list (ACL) rules input in files. Each ACL policy file contains `iptables`, `ip6tables` and `ebtables` categories under the tags [`iptables`], [`ip6tables`] and [`ebtables`] respectively.

Each rule in an ACL policy must be assigned to one of the rule categories above.

See `man cl-acltool(5)` for ACL rule details. For `iptables` rule syntax, see `man iptables(8)`. For `ip6tables` rule syntax, see `man ip6tables(8)`. For `ebtables` rule syntax, see `man ebtables(8)`.

See `man cl-acltool(5)` and `man cl-acltool(8)` for further details on using `cl-acltool`; however, some examples are listed here, and more are listed [later in this chapter \(see page \)](#).



By default:

- ACL policy files are located in `/etc/cumulus/acl/policy.d/`.

- All `*.rules` files in this directory are included in `/etc/cumulus/acl/policy.conf`.
- All files included in this `policy.conf` file are installed when the switch boots up.
- The `policy.conf` file expects rules files to have a `.rules` suffix as part of the file name.

Here is an example ACL policy file:

```
[iptables]
-A INPUT --in-interface swp1 -p tcp --dport 80 -j ACCEPT
-A FORWARD --in-interface swp1 -p tcp --dport 80 -j ACCEPT

[ip6tables]
-A INPUT --in-interface swp1 -p tcp --dport 80 -j ACCEPT
-A FORWARD --in-interface swp1 -p tcp --dport 80 -j ACCEPT

[ebtables]
-A INPUT -p IPv4 -j ACCEPT
-A FORWARD -p IPv4 -j ACCEPT
```

You can use wildcards or variables to specify chain and interface lists to ease administration of rules.

**⚠ Interface Wildcards** – Currently only `swp+` and `bond+` are supported as wildcard names. There may be kernel restrictions in supporting more complex wildcards like `swp1+` etc.

```
INGRESS = swp+
INPUT_PORT_CHAIN = INPUT,FORWARD

[iptables]
-A $INPUT_PORT_CHAIN --in-interface $INGRESS -p tcp --dport 80 -j ACCEPT

[ip6tables]
-A $INPUT_PORT_CHAIN --in-interface $INGRESS -p tcp --dport 80 -j ACCEPT

[ebtables]
-A INPUT -p IPv4 -j ACCEPT
```

ACL rules for the system can be written into multiple files under the default `/etc/cumulus/acl/policy.conf` directory. The ordering of rules during installation follows the sort order of the files based on their file names.

Use multiple files to stack rules. The example below shows two rules files separating rules for management and datapath traffic:

```

cumulus@switch:~$ ls /etc/cumulus/acl/policy.d/ 00sample_mgmt.rules
01sample_datapath.rules
cumulus@switch:~$ cat /etc/cumulus/acl/policy.d/00sample_mgmt.rules

INGRESS_INTF = swp+
INGRESS_CHAIN = INPUT

[iptables]
# protect the switch management
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -s 10.0.14.2 -d 10.0.15.8 -p
tcp -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -s 10.0.11.2 -d 10.0.12.8 -p
tcp -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -d 10.0.16.8 -p udp -j DROP

cumulus@switch:~$ cat /etc/cumulus/acl/policy.d/01sample_datapath.rules
INGRESS_INTF = swp+
INGRESS_CHAIN = INPUT, FORWARD

[iptables]
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -s 192.0.2.5 -p icmp -j
ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -s 192.0.2.6 -d 192.0.2.4 -j
DROP
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -s 192.0.2.2 -d 192.0.2.8 -j
DROP

```

Install all ACL policies under a directory:

```

cumulus@switch:~$ sudo cl-acltool -i -P ./rules
Reading files under rules
Reading rule file ./rules/01_http_rules.txt ...
Processing rules in file ./rules/01_http_rules.txt ...
Installing acl policy ...
Done.

```

Install all rules and policies included in `/etc/cumulus/acl/policy.conf`:

```

cumulus@switch:~$ sudo cl-acltool -i

```

## Specifying which Policy Files to Install

By default, any `.rules` file you configure in `/etc/cumulus/acl/policy.d/` get installed by Cumulus Linux. To add other policy files to an ACL, you need to include them in `/etc/cumulus/acl/policy.conf`. For example, in order for Cumulus Linux to install a rule in a policy file called `01_new.rules`, you would add `include /etc/cumulus/acl/policy.d/01_new.rules` to `policy.conf`, as in this example:

```
cumulus@switch:~$ sudo vi /etc/cumulus/acl/policy.conf

#
# This file is a master file for acl policy file inclusion
#
# Note: This is not a file where you list acl rules.
#
# This file can contain:
# - include lines with acl policy files
#   example:
#       include <filepath>
#
# see manpage cl-acltool(5) and cl-acltool(8) for how to write policy files
#

include /etc/cumulus/acl/policy.d/*.rules
include /etc/cumulus/acl/policy.d/01_new.rules
```

## Hardware Limitations on Number of Rules

The maximum number of rules that can be handled in hardware is a function of the platform type (ASIC, such as Tomahawk or Spectrum) and a mix of IPv4 and/or IPv6. See the [HCL](#) to determine which platform type applies to a particular switch.

If the maximum number of rules for a particular table is exceeded, `cl-acltool -i` generates the following error:

```
error: hw sync failed (sync_acl hardware installation failed) Rolling back
.. failed.
```

## Broadcom Tomahawk Limits

Direction	Atomic Mode IPv4 Rules	Atomic Mode IPv6 Rules	Nonatomic Mode IPv4 Rules	Nonatomic Mode IPv6 Rules
Ingress	512	512	1024	1024

Direction	Atomic Mode IPv4 Rules	Atomic Mode IPv6 Rules	Nonatomic Mode IPv4 Rules	Nonatomic Mode IPv6 Rules
Egress	512	256	1024	512

### Broadcom Trident II+ Limits

Direction	Atomic Mode IPv4 Rules	Atomic Mode IPv6 Rules	Nonatomic Mode IPv4 Rules	Nonatomic Mode IPv6 Rules
Ingress	4096	4096	8192	8192
Egress	512	256	1024	512

### Broadcom Trident II Limits

Direction	Atomic Mode IPv4 Rules	Atomic Mode IPv6 Rules	Nonatomic Mode IPv4 Rules	Nonatomic Mode IPv6 Rules
Ingress	1024	1024	2048	2048
Egress	512	256	1024	512

### Mellanox Spectrum Limits

Unlike Broadcom ASICs, the Mellanox Spectrum ASIC hardware counts ACL rules differently. The ASIC also has one common [TCAM](#) for both ingress and egress, and it may be used for other non-ACL-related resources.

Further, when a rule is installed in software then sent to the ASIC, the number of rules allowed is determined by the size of the rule itself. The Spectrum ASIC allocates 324KB total for rules, broken down as follows:

- 18K x 18 byte rules
- 9K x 36 byte rules
- 6K x 54 byte rules

### Supported Rule Types

The `iptables`/`ip6tables`/`ebtables` construct tries to layer the Linux implementation on top of the underlying hardware but they are not always directly compatible. Here are the supported rules for chains in `iptables`, `ip6tables` and `ebtables`.



To learn more about any of the options shown in the tables below, run `iptables -h [name of option]`. The same help syntax works for options for `ip6tables` and `ebtables`.

Click here to see Example of Help Syntax for an Etables target

```
root@leaf1# ebtables -h tricolorpolice
<...snip...
tricolorpolice option:
--set-color-mode STRING setting the mode in blind or aware
--set-cir INT setting committed information rate in kbits per
second
--set-cbs INT setting committed burst size in kbyte
--set-pir INT setting peak information rate in kbits per
second
--set-ebs INT setting excess burst size in kbyte
--set-conform-action-dscp INT setting dscp value if the
action is accept for conforming packets
--set-exceed-action-dscp INT setting dscp value if the action
is accept for exceeding packets
--set-violate-action STRING setting the action (accept/drop) f
or violating packets
--set-violate-action-dscp INT setting dscp value if the
action is accept for violating packets
Supported chains for the filter table:
INPUT FORWARD OUTPUT
```

## *iptables/ip6tables Rule Support*

Rule Element	Supported	Unsupported
<b>Matches</b>	<ul style="list-style-type: none"> <li>Src/Dst, IP protocol</li> <li>In/out interface</li> <li>IPv4: icmp, ttl,</li> <li>IPv6: icmp6, frag, hl,</li> <li>IP common: tcp (<a href="#">with flags (see page 116)</a>), udp, multiport, TOS, DSCP, addrtpe</li> </ul>	<ul style="list-style-type: none"> <li>Rules with input/output Ethernet interfaces are ignored</li> <li>Inverse matches</li> </ul>
<b>Standard Targets</b>	<ul style="list-style-type: none"> <li>ACCEPT, DROP</li> </ul>	<ul style="list-style-type: none"> <li>RETURN, QUEUE, STOP, Fall Thru, Jump</li> </ul>
<b>Extended Targets</b>	<ul style="list-style-type: none"> <li>LOG (IPv4/IPv6); UID is not supported for LOG</li> <li>TCP SEQ, TCP options or IP options</li> <li>ULOG</li> <li>SETQOS</li> <li>DSCP</li> </ul>	

Rule Element	Supported	Unsupported
	<p><i>Unique to Cumulus Linux:</i></p> <ul style="list-style-type: none"> <li>• SPAN</li> <li>• ERSPAN (IPv4/IPv6)</li> <li>• POLICE</li> <li>• TRICOLORPOLICE</li> <li>• SETCLASS</li> </ul>	

## ebtables Rule Support

Rule Element	Supported	Unsupported
<b>Matches</b>	<ul style="list-style-type: none"> <li>• ether type</li> <li>• input interface/wildcard</li> <li>• output interface/wildcard</li> <li>• src/dst MAC</li> <li>• IP: src, dest, tos, proto, sport, dport</li> <li>• IPv6: tclass, icmp6: type, icmp6: code range, src /dst addr, sport, dport</li> </ul>	<ul style="list-style-type: none"> <li>• Inverse matches</li> <li>• Proto length</li> <li>• VLAN</li> <li>• 802.1p (CoS)</li> </ul>
<b>Standard Targets</b>	<ul style="list-style-type: none"> <li>• ACCEPT, DROP</li> </ul>	<ul style="list-style-type: none"> <li>• Return, Continue, Jump, Fall Thru</li> </ul>
<b>Extended Targets</b>	<ul style="list-style-type: none"> <li>• Ulog</li> <li>• log</li> </ul> <p><i>Unique to Cumulus Linux:</i></p> <ul style="list-style-type: none"> <li>• span</li> <li>• erspan</li> <li>• police</li> <li>• tricolorpolice</li> <li>• setclass</li> </ul>	

## Other Unsupported Rules

- Rules that have no matches and accept all packets in a chain are currently ignored. This probably has side effects in the sense that the rules below them do get hit, when normally they wouldn't.
- Chain default rules (which are ACCEPT) are also ignored.
- Rules that match on eth\* interfaces are assumed to be Linux management interfaces and are ignored.

## Common Examples

### Policing Control Plane and Data Plane Traffic

You can configure quality of service for traffic on both the control plane and the data plane. By using QoS policers, you can rate limit traffic so incoming packets get dropped if they exceed specified thresholds.



Counters on POLICE ACL rules in `iptables` do not currently show the packets that are dropped due to those rules.

Use the `POLICE` target with `iptables`. `POLICE` takes these arguments:

- `--set-class value`: Sets the system internal class of service queue configuration to *value*.
- `--set-rate value`: Specifies the maximum rate in kilobytes (KB) or packets.
- `--set-burst value`: Specifies the number of packets or kilobytes (KB) allowed to arrive sequentially.
- `--set-mode string`: Sets the mode in KB (kilobytes) or *pkt* (packets) for rate and burst size.

For example, to rate limit the incoming traffic on `swp1` to 400 packets/second with a burst of 100 packets /second and set the class of the queue for the policed traffic as 0, set this rule in your appropriate `.rules` file:

```
-A INPUT --in-interface swp1 -j POLICE --set-mode pkt --set-rate 400 --set-burst 100 --set-class 0
```

Here is another example of control plane ACL rules to lock down the switch. You specify them in `/etc/cumulus/acl/policy.d/00control_plane.rules`:

```
INGRESS_INTF = swp+
INGRESS_CHAIN = INPUT
INNFWD_CHAIN = INPUT, FORWARD
MARTIAN_SOURCES_4 = "240.0.0.0/5,127.0.0.0/8,224.0.0.0/8,255.255.255.255/32"
MARTIAN_SOURCES_6 = "ff00::/8,::128,::ffff:0.0.0.0/96,::1/128"

# Custom Policy Section
SSH_SOURCES_4 = "192.168.0.0/24"
NTP_SERVERS_4 = "192.168.0.1/32,192.168.0.4/32"
DNS_SERVERS_4 = "192.168.0.1/32,192.168.0.4/32"
SNMP_SERVERS_4 = "192.168.0.1/32"

[iptables]
-A $INNFWD_CHAIN --in-interface $INGRESS_INTF -s $MARTIAN_SOURCES_4 -j DROP
```

```

-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p ospf -j POLICE --set-mode
pkt --set-rate 2000 --set-burst 2000 --set-class 7
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --dport bgp -j POLICE
--set-mode pkt --set-rate 2000 --set-burst 2000 --set-class 7
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --sport bgp -j POLICE
--set-mode pkt --set-rate 2000 --set-burst 2000 --set-class 7
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p icmp -j POLICE --set-mode
pkt --set-rate 100 --set-burst 40 --set-class 2
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p udp --dport bootps:bootpc
-j POLICE --set-mode pkt --set-rate 100 --set-burst 100 --set-class 2
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --dport bootps:bootpc
-j POLICE --set-mode pkt --set-rate 100 --set-burst 100 --set-class 2
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p igmp -j POLICE --set-mode
pkt --set-rate 300 --set-burst 100 --set-class 6

# Custom policy
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --dport 22 -s
$SSH_SOURCES_4 -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p udp --sport 123 -s
$NTP_SERVERS_4 -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p udp --sport 53 -s
$DNS_SERVERS_4 -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p udp --dport 161 -s
$SNMP_SERVERS_4 -j ACCEPT

# Allow UDP traceroute when we are the current TTL expired hop
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p udp --dport 1024:65535 -m
ttl --ttl-eq 1 -j ACCEPT
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -j DROP

```

## Setting DSCP on Transit Traffic

The examples here use the *mangle* table to modify the packet as it transits the switch. DSCP is expressed in decimal notation in the examples below.

```

[iptables]

#Set SSH as high priority traffic.
-t mangle -A FORWARD -p tcp --dport 22 -j DSCP --set-dscp 46

#Set everything coming in SWP1 as AF13
-t mangle -A FORWARD --in-interface swp1 -j DSCP --set-dscp 14

#Set Packets destined for 10.0.100.27 as best effort
-t mangle -A FORWARD -d 10.0.100.27/32 -j DSCP --set-dscp 0

```

```
#Example using a range of ports for TCP traffic
-t mangle -A FORWARD -p tcp -s 10.0.0.17/32 --sport 10000:20000 -d 10.0.100.27/32 --dport 10000:20000 -j DSCP --set-dscp 34
```

## Verifying DSCP Values on Transit Traffic

The examples here use the DSCP match criteria in combination with other IP, TCP and interface matches to identify traffic and count the number of packets.

```
[iptables]

#Match and count the packets that match SSH traffic with DSCP EF
-A FORWARD -p tcp --dport 22 -m dscp --dscp 46 -j ACCEPT

#Match and count the packets coming in SWP1 as AF13
-A FORWARD --in-interface swp1 -m dscp --dscp 14 -j ACCEPT
#Match and count the packets with a destination 10.0.0.17 marked best
effort
-A FORWARD -d 10.0.100.27/32 -m dscp --dscp 0 -j ACCEPT

#Match and count the packets in a port range with DSCP AF41
-A FORWARD -p tcp -s 10.0.0.17/32 --sport 10000:20000 -d 10.0.100.27/3
2 --dport 10000:20000 -m dscp --dscp 34 -j ACCEPT
```

## Checking the Packet and Byte Counters for ACL Rules

To verify the counters, using the above example rules, first send test traffic matching the patterns through the network. The following example generates traffic with `mz`, which can be installed on host servers or even on Cumulus Linux switches. Once traffic is sent to validate the counters, they are matched on switch1 use `cl-acltool`.

```
# Send 100 TCP packets on Host1 with a DSCP value of EF with a
destination of Host2 TCP port 22:

cumulus@host1$ mz eth1 -A 10.0.0.17 -B 10.0.100.27 -c 100 -v -t tcp "d
p=22,dscp=46"
IP: ver=4, len=40, tos=184, id=0, frag=0, ttl=255, proto=6, sum=0,
SA=10.0.0.17, DA=10.0.100.27,
payload=[see next layer]
TCP: sp=0, dp=22, S=42, A=42, flags=0, win=10000, len=20, sum=0,
payload=

# Verify the 100 packets are matched on switch1
cumulus@switch1$ sudo cl-acltool -L ip
-----
Listing rules of type iptables:
-----
```

```
TABLE filter :
Chain INPUT (policy ACCEPT 9314 packets, 753K bytes)
  pkts bytes target     prot opt in      out      source
destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in      out      source
destination
    100   6400 ACCEPT      tcp  --  any     any     anywhere
anywhere          tcp dpt:ssh DSCP match 0x2e
      0     0 ACCEPT      all  --  swp1    any     anywhere
anywhere          DSCP match 0x0e
      0     0 ACCEPT      all  --  any     any      10.0.0.17
anywhere          DSCP match 0x00
      0     0 ACCEPT      tcp  --  any     any      10.0.0.17
.0.100.27          tcp spts:webmin:20000 dpts:webmin:2002      10
```

```
# Send 100 packets with a small payload on Host1 with a DSCP value of AF13 with a destination of Host2:
```

```
cumulus@host1$ mz eth1 -A 10.0.0.17 -B 10.0.100.27 -c 100 -v -t ip
  IP: ver=4, len=20, tos=0, id=0, frag=0, ttl=255, proto=0, sum=0, SA=
10.0.0.17, DA=10.0.100.27,
  payload=
```

```
# Verify the 100 packets are matched on switch1
cumulus@switch1$ sudo cl-acltool -L ip
```

```
-----
```

```
Listing rules of type iptables:
```

```
-----
```

```
TABLE filter :
Chain INPUT (policy ACCEPT 9314 packets, 753K bytes)
  pkts bytes target     prot opt in      out      source
destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in      out      source
destination
    100   6400 ACCEPT      tcp  --  any     any     anywhere
anywhere          tcp dpt:ssh DSCP match 0x2e
    100   7000 ACCEPT      all  --  swp3    any     anywhere
anywhere          DSCP match 0x0e
    100   6400 ACCEPT      all  --  any     any      10.0.0.17
anywhere          DSCP match 0x00
      0     0 ACCEPT      tcp  --  any     any      10.0.0.17
.0.100.27          tcp spts:webmin:20000 dpts:webmin:2002      10
```

```
# Send 100 packets on Host1 with a destination of Host2:
```

```
cumulus@host1$ mz eth1 -A 10.0.0.17 -B 10.0.100.27 -c 100 -v -t ip
```

```

IP: ver=4, len=20, tos=56, id=0, frag=0, ttl=255, proto=0, sum=0,
SA=10.0.0.17, DA=10.0.100.27,
payload=

# Verify the 100 packets are matched on switch1
cumulus@switch1$ sudo cl-acltool -L ip
-----
Listing rules of type iptables:
-----
TABLE filter :
Chain INPUT (policy ACCEPT 9314 packets, 753K bytes)
  pkts bytes target     prot opt in     out     source
destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source
destination
  100   6400 ACCEPT    tcp   --  any      any      anywhere
anywhere          tcp  dpt:ssh  DSCP match 0x2e
  100   7000 ACCEPT    all   --  swp3    any      anywhere
anywhere          DSCP match 0x0e
      0     0 ACCEPT    all   --  any      any      10.0.0.17
anywhere          DSCP match 0x00
      0     0 ACCEPT    tcp   --  any      any      10.0.0.17
.0.100.27        tcp  spts:webmin:20000 dpts:webmin:2002Still working
  
```

## Filtering Specific TCP Flags

The example solution below creates rules on the INPUT and FORWARD chains to drop ingress IPv4 and IPv6 TCP packets when the SYN bit is set and the RST, ACK and FIN bits are reset. The default for the INPUT and FORWARD chains allows all other packets. The ACL is applied to ports swp20 and swp21. After configuring this ACL, new TCP sessions that originate from ingress ports swp20 and swp21 will not be allowed. TCP sessions that originate from any other port are allowed.

```

INGRESS_INTF = swp20,swp21

[iptables]
-A INPUT, FORWARD --in-interface $INGRESS_INTF -p tcp --syn -j DROP
[ip6tables]
-A INPUT, FORWARD --in-interface $INGRESS_INTF -p tcp --syn -j DROP
  
```

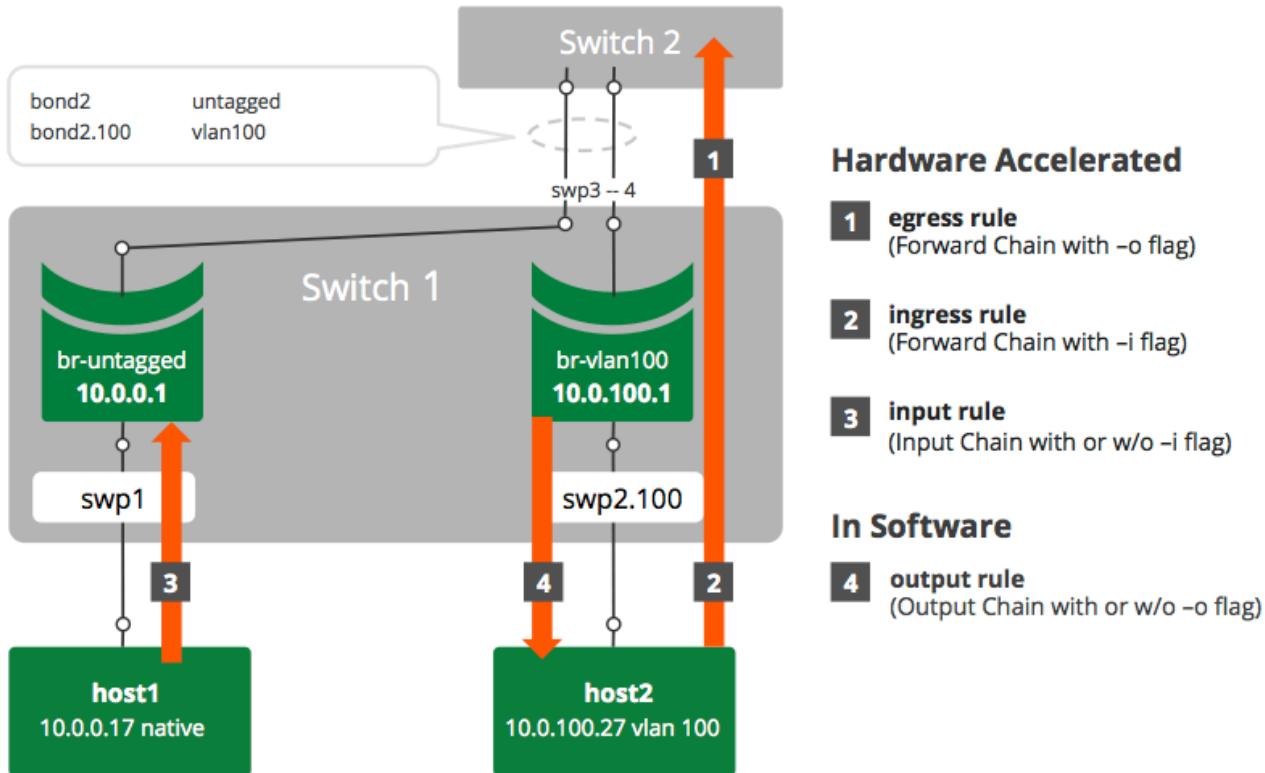
The `--syn` flag in the above rule matches packets with the SYN bit set and the ACK, RST and FIN bits are cleared. It is equivalent to using `-tcp-flags SYN,RST,ACK,FIN SYN`. For example, the above rule could be re-written as:

```

-A INPUT, FORWARD --in-interface $INGRESS_INTF -p tcp --tcp-flags SYN,
RST,ACK,FIN SYN -j DROP
  
```

## Example Scenario

The following example scenario demonstrates where several different rules are applied to show what is possible.



Following are the configurations for the two switches used in these examples. The configuration for each switch appears in `/etc/network/interfaces` on that switch.

## Switch 1 Configuration

```
auto swp1
iface swp1

auto swp2
iface swp2

auto swp3
iface swp3

auto swp4
iface swp4

auto bond2
iface bond2
    bond-slaves swp3 swp4

auto br-untagged
```

```

iface br-untagged
  address 10.0.0.1/24
  bridge_ports swp1 bond2
  bridge_stp on

auto br-tag100
iface br-tag100
  address 10.0.100.1/24
  bridge_ports swp2.100 bond2.100
  bridge_stp on

```

## Switch 2 Configuration

```

auto swp3
iface swp3

auto swp4
iface swp4

auto br-untagged
iface br-untagged
  address 10.0.0.2/24
  bridge_ports bond2
  bridge_stp on

auto br-tag100
iface br-tag100
  address 10.0.100.2/24
  bridge_ports bond2.100
  bridge_stp on

auto bond2
iface bond2
  bond-slaves swp3 swp4

```

## Egress Rule

The following rule blocks any TCP with destination port 200 traffic going from host1 or host2 through the switch (corresponding to rule 1 in the diagram above).

```
[iptables] -A FORWARD -o bond2 -p tcp --dport 200 -j DROP
```

## Ingress Rule

The following rule blocks any UDP traffic with source port 200 going from host1 through the switch (corresponding to rule 2 in the diagram above).

```
[iptables] -A FORWARD -i swp2 -p udp --sport 200 -j DROP
```

## **Input Rule**

The following rule blocks any UDP traffic with source port 200 and destination port 50 going from host1 to the switch (corresponding to rule 3 in the diagram above).

```
[iptables] -A INPUT -i swp1 -p udp --sport 200 --dport 50 -j DROP
```

## **Output Rule**

The following rule blocks any TCP traffic with source port 123 and destination port 123 going from Switch 1 to host2 (corresponding to rule 4 in the diagram above).

```
[iptables] -A OUTPUT -o br-tag100 -p tcp --sport 123 --dport 123 -j DROP
```

## **Combined Rules**

The following rule blocks any TCP traffic with source port 123 and destination port 123 going from any switch port egress or generated from Switch 1 to host1 or host2 (corresponding to rules 1 and 4 in the diagram above).

```
[iptables] -A OUTPUT, FORWARD -o swp+ -p tcp --sport 123 --dport 123 -j DROP
```

This also becomes 2 ACLs, and is effectively the same as:

```
[iptables]
-A FORWARD -o swp+ -p tcp --sport 123 --dport 123 -j DROP
-A OUTPUT -o swp+ -p tcp --sport 123 --dport 123 -j DROP
```

## **Layer 2-only Rules/ebtables**

The following rule blocks any traffic with source MAC address 00:00:00:00:00:12 and destination MAC address 08:9e:01:ce:e2:04 going from any switch port egress/ingress.

```
[ebtables] -A FORWARD -s 00:00:00:00:00:12 -d 08:9e:01:ce:e2:04 -j DROP
```

## Useful Links

- [www.netfilter.org](http://www.netfilter.org)
- Netfilter.org packet filtering how-to

## Caveats and Errata

### Not All Rules Supported

As mentioned in the [Supported Rules](#) section (see page 109) above, not all `iptables`, `ip6tables` or `ebtables` rules are supported. Refer to that section for specific rule support.

### Log Actions Cannot Be Forwarded

Logged packets cannot be forwarded. The hardware cannot both forward a packet and send the packet to the control plane (or kernel) for logging. To emphasize this, a log action must also have a drop action.

## Tomahawk Hardware Limitations

### Rate Limiting per Pipeline, Not Global

On Tomahawk switches, the field processor (FP) polices on a per-pipeline basis instead of globally, as with a Trident II switch. If packets come in to different switch ports that are on different pipelines on the ASIC, they may be rate limited differently.

For example, your switch is set so BFD is rate limited to 2000 packets per second. When the BFD packets are received on port1/pipe1 and port2/pipe2, they would each be rate limited at 2000 pps, thus the switch is rate limiting at 4000 pps overall. Since there are four pipelines on a Tomahawk switch, you could see a 4x increase of your configured rate limits.

### Ping-pong Mode Enabled by Default

In Cumulus Linux, the ping-pong atomic policy update mode is enabled by default. If you have Tomahawk switches and plan to use only SPAN and/or mangle rules, you must disable ping-pong mode.

To do so, set the value for `acl.non_atomic_update_mode` to true in `/etc/cumulus/switchd.conf`:

```
acl.non_atomic_update_mode = TRUE
```

Then restart `switchd` (see page 128).

### `iptables` Interactions with `cl-acltool`

Since Cumulus Linux is a Linux operating system, the `iptables` commands can be used directly and will work. However, you should consider using `cl-acltool` instead because:

- Without using `cl-acltool`, rules are not installed into hardware.
- Running `cl-acltool -i` (the installation command) will reset all rules and delete anything that is not stored in `/etc/cumulus/acl/policy.conf`.

For example performing:

```
cumulus@switch:~$ sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

Does work, and the rules appear when you run `cl-acltool -L`:

```
cumulus@switch:~$ sudo cl-acltool -L ip
-----
Listing rules of type iptables:
-----
TABLE filter :
Chain INPUT (policy ACCEPT 72 packets, 5236 bytes)
  pkts bytes target     prot opt in      out
source                      destination
          0     0  DROP      icmp --  any    any
anywhere                    anywhere           icmp echo-request
```

However, running `cl-acltool -i` or `reboot` will remove them. To ensure all rules that can be in hardware are hardware accelerated, place them in `/etc/cumulus/acl/policy.conf` and run `cl-acltool -i`.

## Where to Assign Rules

- If a switch port is assigned to a bond, any egress rules must be assigned to the bond.
- When using the OUTPUT chain, rules must be assigned to the source. For example, if a rule is assigned to the switch port in the direction of traffic but the source is a bridge (VLAN), the traffic won't be affected by the rule and must be applied to the bridge.
- If all transit traffic needs to have a rule applied, use the FORWARD chain, not the OUTPUT chain.

## Generic Error Message Displayed after ACL Rule Installation Failure

After an ACL rule installation failure, a generic error message like the following is displayed:

```
cumulus@switch:$ sudo cl-acltool -i -p 00control_plane.rules
Using user provided rule file 00control_plane.rules
Reading rule file 00control_plane.rules ...
Processing rules in file 00control_plane.rules ...
error: hw sync failed (sync_acl hardware installation failed)
Installing acl policy... Rolling back ..
failed.
```

## Managing Application Daemons

You manage application daemons (services) in Cumulus Linux in the following ways:

- Identifying active listener ports
- Identifying daemons currently active or stopped
- Identifying boot time state of a specific daemon
- Disabling or enabling a specific daemon

### ***Using systemd and the systemctl Command***

In general, you manage services using `systemd` via the `systemctl` command. You use it with any service on the switch to start/stop/restart/reload/enable/disable/reenable or get the status of the service.

```
systemctl start | stop | restart | status | reload | enable | disable |
reenable SERVICENAME.service
```

For example to restart networking, run the command:

```
systemctl restart networking.service
```



Unlike the `service` command in Debian Wheezy, the service name is written **after** the `systemctl` subcommand, not before it.

### ***Understanding the systemctl Subcommands***

`systemctl` has a number of subcommands that perform a specific operation on a given daemon.

- **status**: Returns the status of the specified daemon.
- **start**: Starts the daemon.
- **stop**: Stops the daemon.
- **restart**: Stops, then starts the daemon, all the while maintaining state. So if there are dependent services or services that mark the restarted service as *Required*, the other services also get restarted. For example, running `systemctl restart quagga.service` restarts any of the routing protocol daemons that are enabled and running, such as `bgpd` or `ospfd`.
- **reload**: Reloads a daemon's configuration.
- **enable**: Enables the daemon to start when the system boots, but does not start it unless you use the `systemctl start SERVICENAME.service` command or reboot the switch.

- **disable**: Disables the daemon, but does not stop it unless you use the `systemctl stop SERVICENAME.service` command or reboot the switch. A disabled daemon can still be started or stopped.
- **reenable**: Disables, then enables a daemon. You might need to do this so that any new *Wants* or *WantedBy* lines create the symlinks necessary for ordering. This has no side effects on other daemons.

## Ensuring a Service Starts after Multiple Restarts

By default, `systemd` is configured to try to restart a particular service only a certain number of times within a given interval before the service fails to start at all. The settings for this are stored in the service script. The settings are *StartLimitInterval* (which defaults to 10 seconds) and *StartBurstLimit* (which defaults to 5 attempts), but many services override these defaults, sometimes with much longer times. `switchd.service`, for example, sets *StartLimitInterval=10m* and *StartBurstLimit=3*, which means if you restart `switchd` more than 3 times in 10 minutes, it will not start.

When the restart fails for this reason, a message similar to the following appears:

```
Job for switchd.service failed. See 'systemctl status switchd.service' and  
'journalctl -xn' for details.
```

And `systemctl status switchd.service` shows output similar to:

```
Active: failed (Result: start-limit) since Thu 2016-04-07 21:55:14 UTC; 15s  
ago
```

To clear this error, run `systemctl reset-failed switchd.service`. If you know you are going to restart frequently (multiple times within the *StartLimitInterval*), you can run the same command before you issue the restart request. This also applies to stop followed by start.

## Contents

(Click to expand)

- Using `systemd` and the `systemctl` Command (see page 122)
  - Understanding the `systemctl` Subcommands (see page 122)
  - Ensuring a Service Starts after Multiple Restarts (see page 123)
- Contents (see page 123)
- Identifying Active Listener Ports for IPv4 and IPv6 (see page 123)
- Identifying Daemons Currently Active or Stopped (see page 124)
- Identifying Boot Time State of a Specific Daemon (see page 125)

## Identifying Active Listener Ports for IPv4 and IPv6

You can identify the active listener ports under both IPv4 and IPv6 using the `lsof` command:

```
cumulus@switch:~$ sudo lsof -Pnl +M -i4
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
ntpd 1882 104 16u IPv4 3954 0t0 UDP *:123
ntpd 1882 104 18u IPv4 3963 0t0 UDP 127.0.0.1:123
ntpd 1882 104 19u IPv4 3964 0t0 UDP 192.168.8.37:123
snmpd 1987 105 8u IPv4 5423 0t0 UDP *:161
zebra 1993 103 10u IPv4 5151 0t0 TCP 127.0.0.1:2601 (LISTEN)
sshd 2496 0 3u IPv4 5809 0t0 TCP *:22 (LISTEN)
sshd 31700 0 3r IPv4 187630 0t0 TCP 192.168.8.37:22->192.168.8.3:50386
(ESTABLISHED)

cumulus@switch:~$ sudo lsof -Pnl +M -i6
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
ntpd 1882 104 17u IPv6 3955 0t0 UDP *:123
ntpd 1882 104 20u IPv6 3965 0t0 UDP [::1]:123
ntpd 1882 104 21u IPv6 3966 0t0 UDP [fe80::7272:cfff:fe96:6639]:123
sshd 2496 0 4u IPv6 5811 0t0 TCP *:22 (LISTEN)
```

## ***Identifying Daemons Currently Active or Stopped***

To determine which daemons are currently active or stopped, run `cl-service-summary`:

```
cumulus@switch:~$ sudo cl-service-summary
Service cron      enabled   active
Service ssh       enabled   active
Service syslog    enabled   active
Service arp_refresh enabled   active
Service clagd     enabled   active
Service lldpd     enabled   active
Service mstpd     enabled   active
Service poed      enabled   inactive
Service portwd    enabled   inactive
Service ptmd      enabled   active
Service pwdx      enabled   active
Service smond     enabled   active
Service switchd   enabled   active
Service vxrd      disabled  inactive
Service vxsnd     disabled  inactive
Service bgpd      disabled  inactive
Service isisd     disabled  inactive
Service ospf6d    disabled  inactive
Service ospfd     disabled  inactive
```

Service rdnbrd	disabled	inactive
Service ripd	disabled	inactive
Service ripngd	disabled	inactive
Service zebra	disabled	inactive

Another way to get this information is to use the `sudo systemctl status` command, then pipe the results to `grep`, using the - or + operators:

```
cumulus@switch:~$ sudo systemctl status | grep +
[ ? ] acلينت
[ + ] arp_refresh
[ + ] auditd
...
cumulus@switch:~$ sudo systemctl status | grep -
[ - ] isc-dhcp-server
[ - ] openvswitch-vtep
[ - ] ptmd
...
```

## Identifying Boot Time State of a Specific Daemon

The `ls` command can provide the boot time state of a daemon. A file link with a name starting with **S** identifies a boot-time-enabled daemon. A file link with a name starting with **K** identifies a disabled daemon.

```
cumulus@switch:~/etc$ sudo ls -l rc*.d | grep <daemon name>
```

For example:

```
cumulus@switch:~/etc$ sudo ls -l rc*.d | grep snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 K02snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 K02snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 S01snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 S01snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 S01snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 S01snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 15 Apr 4 2014 K02snmpd -> ../init.d/snmpd
```

## Configuring switchd

**switchd** is the daemon at the heart of Cumulus Linux. It communicates between the switch and Cumulus Linux, and all the applications running on Cumulus Linux.

The **switchd** configuration is stored in `/etc/cumulus/switchd.conf`.



Versions of Cumulus Linux prior to 2.1 stored the **switchd** configuration at `/etc/default/switchd`.

### Contents

(Click to expand)

- Contents (see page 126)
- The switchd File System (see page 126)
- Configuring switchd Parameters (see page 128)
- Restarting switchd (see page 128)
- Booting a Switch without a License (see page 129)
- Commands (see page 129)
- Configuration Files (see page 129)

### The switchd File System

**switchd** also exports a file system, mounted on `/cumulus/switchd`, that presents all the **switchd** configuration options as a series of files arranged in a tree structure. You can see the contents by parsing the **switchd** tree; run `tree /cumulus/switchd`. The output below is for a switch with one switch port configured:

```
cumulus@cumulus:~# sudo tree /cumulus/switchd/
/cumulus/switchd/
|-- config
|   |-- acl
|   |   |-- non_atomic_update_mode
|   |   `-- optimize_hw
|   |-- arp
|   |   `-- next_hops
|   |-- buf_util
|   |   |-- measure_interval
|   |   `-- poll_interval
|   |-- coalesce
|   |   |-- reducer
|   |   `-- timeout
```

```

|   |-- disable_internal_restart
|   |-- ignore_non_swps
|   |-- interface
|   |   |-- swp1
|   |   |   `-- storm_control
|   |   |       |-- broadcast
|   |   |       |-- multicast
|   |   |       `-- unknown_unicast
|   |-- logging
|   |-- route
|   |   |-- host_max_percent
|   |   |-- max_routes
|   |   `-- table
|   '-- stats
|       `-- poll_interval
|-- ctrl
|   |-- acl
|   |-- hal
|   |   `-- resync
|   |-- logger
|   |-- netlink
|   |   `-- resync
|   |-- resync
|   '-- sample
|       `-- ulog_channel
|-- run
|   '-- route_info
|       |-- ecmp_nh
|       |   |-- count
|       |   |-- max
|       |   `-- max_per_route
|       |-- host
|       |   |-- count
|       |   |-- count_v4
|       |   |-- count_v6
|       |   `-- max
|       |-- mac
|       |   |-- count
|       |   `-- max
|       '-- route
|           |-- count_0
|           |-- count_1
|           |-- count_total
|           |-- count_v4
|           |-- count_v6

```

```
|           |-- mask_limit
|           |-- max_0
|           |-- max_1
|           `-- max_total
`-- version
```

## Configuring switchd Parameters

You can use `cl-cfg` to configure many `switchd` parameters at runtime (like ACLs, interfaces, and route table utilization), which minimizes disruption to your running switch. However, some options are read only and cannot be configured at runtime.

For example, to see data related to routes, run:

```
cumulus@cumulus:~$ sudo cl-cfg -a switchd | grep route
route.table = 254
route.max_routes = 32768
route.host_max_percent = 50
cumulus@cumulus:~$
```

To modify the configuration, run `cl-cfg -w`. For example, to set the buffer utilization measurement interval to 1 minute, run:

```
cumulus@cumulus:~$ sudo cl-cfg -w switchd buf_util.measure_interval=1
```

To verify that the value changed, use `grep`:

```
cumulus@cumulus:~# cl-cfg -a switchd | grep buf
buf_util.poll_interval = 0
buf_util.measure_interval = 1
```



You can get some of this information by running `cl-resource-query`; though you cannot update the `switchd` configuration with it.

## Restarting switchd

Whenever you modify any `switchd` hardware configuration file (typically changing any `*.conf` file that requires making a change to the switching hardware, like `/etc/cumulus/datapath/traffic.conf`), you must restart `switchd` for the change to take effect:

```
cumulus@switch:~$ sudo systemctl restart switchd.service
```



You do not have to restart the `switchd` service when you update a network interface configuration (that is, edit `/etc/network/interfaces`).



Restarting `switchd` causes all network ports to reset in addition to resetting the switch hardware configuration.

## Booting a Switch without a License

If a license is not installed on a Cumulus Linux switch, the `switchd` service will not start. If you install the license again, start `switchd` with:

```
cumulus@switch:~$ sudo systemctl start switchd.service
```

## Commands

- cl-cfg

## Configuration Files

- `/etc/cumulus/switchd.conf`

## Power over Ethernet - PoE

Cumulus Linux supports Power over Ethernet (PoE), so certain Cumulus Linux switches can supply power from Ethernet switch ports to enabled devices over the Ethernet cables that connect them.

The [currently supported platform](#) includes:

- Accton AS4610-54P, which supports PoE and PoE+ (an [early access feature](#)) and configuration over Ethernet layer 2 LLDP for power negotiation

## How It Works

When a powered device is connected to the switch via an Ethernet cable:

- If the available power is greater than the power required by the connected device, power is supplied to the switch port, and the device powers on
- If available power is less than the power required by the connected device and the switch port's priority is less than the port priority set on all powered ports, power is **not** supplied to the port

- If available power is less than the power required by the connected device and the switch port's priority is greater than the priority of a currently powered port, power is removed from lower priority port(s) and power is supplied to the port
- If the total consumed power exceeds the configured power limit of the power source, low priority ports are turned off. In the case of a tie, the port with the lower port number gets priority

Power is available as follows:

PSU 1	PSU 2	PoE Power Budget
920W	x	750W
x	920W	750W
920W	920W	1650W

The AS4610-54P has an LED on the front panel to indicate PoE status:

- Green: The **poed** daemon is running and no errors are detected
- Yellow: One or more errors are detected or the **poed** daemon is not running

## About Link State and PoE State

Link state and PoE state are completely independent of each other. When a link is brought down on a particular port using `ip link <port> down`, power on that port is not turned off; however, LLDP negotiation is not possible.

## Configuring PoE

You use the `poectl` command utility to configure PoE on a [switch that supports](#) the feature. You can:

- Enable or disable PoE for a given switch port
- Set a switch port's PoE priority to one of three values: *low*, *high* or *critical*

By default, PoE is enabled on all Ethernet/1G switch ports, and these ports are set with a low priority. Switch ports can have low, high or critical priority.

To change the priority for one or more switch ports, run `poectl -p swp# [low|high|critical]`. For example:

```
cumulus@switch:~$ sudo poectl -p swp1-swp5,swp7 high
```

To disable PoE for one or more ports, run `poectl -d [port_numbers]`:

```
cumulus@switch:~$ sudo poectl -d swp1-swp5,swp7
```

To display PoE information for a set of switch ports, run `poectl -i [port_numbers]`:

```
cumulus@switch:~$ sudo poectl -i swp10-swp13
Port          Status           LLDP      Priority PD type      PD
class    Voltage   Current   Power
-----  -----  -----  -----
swp10    connected        negotiating  low  high-power
4        53.5 V     25 mA    3.9 W
swp11    searching       n/a        low  none
none     0.0 V      0 mA     0.0 W
swp12    connected        n/a        low  legacy
2        53.5 V     25 mA    1.4 W
swp13    connected        51.0 W   low  high-power
4        53.6 V     72 mA    3.8 W
```

Or to see all the PoE information for a switch, run `poectl -s`:

```
cumulus@switch:~$ poectl -s
System power:
  Total:      730.0 W
  Used:       11.0 W
  Available:  719.0 W
Connected ports:
  swp11, swp24, swp27, swp48
```

The set commands (priority, enable, disable) either succeed silently or display an error message if the command fails.

## ***poectl Arguments***

The `poectl` command takes the following arguments:

Argument	Description
<code>-h, --help</code>	Show this help message and exit
<code>-i, --port-info</code> <code>PORT_LIST</code>	Returns detailed information for the specified ports. You can specify a range of ports. For example: <code>-i swp1-swp5,swp10</code>



On an Edge-Core AS4610-54P switch, the voltage reported by the `poectl -i` command and measured through a power meter connected to the device varies by 5V. The current and power readings are correct and no difference is seen for them.

Argument	Description
-a, --all	Returns PoE status and detailed information for all ports.
-p, --priority PORT_LIST PRIORITY	Sets priority for the specified ports: low, high, critical.
-d, --disable-ports PORT_LIST	Disables PoE operation on the specified ports.
-e, --enable-ports PORT_LIST	Enables PoE operation on the specified ports.
-s, --system	Returns PoE status for the entire switch.
-r, --reset PORT_LIST	Performs a hardware reset on the specified ports. Use this if one or more ports are stuck in an error state. This does not reset any configuration settings for the specified ports.
-v, --version	Displays version information.
-j, --json	Displays output in JSON format.
--save	Saves the current configuration. The saved configuration is automatically loaded on system boot.
--load	Loads and applies the saved configuration.

## Logging poed Events

The **poed** service logs the following events to syslog:

- When a switch provides power to a powered device
- When a device that was receiving power is removed
- When the power available to the switch changes
- Errors

## Configuring a Global Proxy

Global HTTP and HTTPS proxies are configured in the `/etc/profile.d/` directory of Cumulus Linux.

1. In a terminal, create a new file in the `/etc/profile.d/` directory. In the code example below, the file is called `proxy`, and is created using the text editor `vi`.

```
cumulus@switch:~$ sudo vi /etc/profile.d/proxy
```

2. Add a line to the file to configure either an HTTP or an HTTPS proxy, and save the file:

- HTTP proxy:

```
http_proxy=http://myproxy.domain.com:8080  
export http_proxy
```

- HTTPS proxy:

```
https_proxy=https://myproxy.domain.com:8080  
export https_proxy
```

3. Run the `source` command, to execute the file in the current environment:

```
cumulus@switch:~$ source /etc/profile.d/proxy
```

The proxy is now configured. The `echo` command can be used to confirm a proxy is set up correctly:

- HTTP proxy:

```
cumulus@switch:~$ echo $http_proxy  
http://myproxy.domain.com:8080
```

- HTTPS proxy:

```
cumulus@switch:~$ echo $https_proxy  
https://myproxy.domain.com:8080
```

# Configuring and Managing Network Interfaces

**ifupdown** is the network interface manager for Cumulus Linux. Cumulus Linux 2.1 and later uses an updated version of this tool, **ifupdown2**.

For more information on network interfaces, see [Layer 1 and Switch Port Attributes \(see page 149\)](#).

- i** By default, **ifupdown** is quiet; use the verbose option **-v** when you want to know what is going on when bringing an interface down or up.

## Contents

(Click to expand)

- [Contents \(see page 134\)](#)
- [Commands \(see page 134\)](#)
- [Man Pages \(see page 135\)](#)
- [Configuration Files \(see page 135\)](#)
- [Basic Commands \(see page 135\)](#)
- [ifupdown2 Interface Classes \(see page 136\)
  - \[Bringing All auto Interfaces Up or Down \\(see page 137\\)\]\(#\)](#)
- [Configuring a Loopback Interface \(see page 138\)](#)
- [ifupdown2 Interface Dependencies \(see page 139\)
  - \[ifup Handling of Upper \\(Parent\\) Interfaces \\(see page 142\\)\]\(#\)](#)
- [Configuring IP Addresses \(see page 143\)
  - \[Purging Existing IP Addresses on an Interface \\(see page 144\\)\]\(#\)](#)
- [Specifying User Commands \(see page 145\)](#)
- [Sourcing Interface File Snippets \(see page 145\)](#)
- [Using Globs for Port Lists \(see page 146\)](#)
- [Using Templates \(see page 146\)](#)
- [Adding Descriptions to Interfaces \(see page 147\)](#)
- [Caveats and Errata \(see page 148\)](#)
- [Useful Links \(see page 148\)](#)

## Commands

- [ifdown](#)

- ifquery
- ifreload
- ifup
- mako-render

## Man Pages

The following man pages have been updated for `ifupdown2`:

- man ifdown(8)
- man ifquery(8)
- man ifreload
- man ifup(8)
- man ifupdown-addons-interfaces(5)
- man interfaces(5)

## Configuration Files

- /etc/network/interfaces

## Basic Commands

To bring up an interface or apply changes to an existing interface, run:

```
cumulus@switch:~$ sudo ifup <ifname>
```

To bring down a single interface, run:

```
cumulus@switch:~$ sudo ifdown <ifname>
```

### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To administratively bring an interface up or down, run:

```
cumulus@switch:~$ sudo ip link set dev swp1 {up|down}
```



If you specified *manual* as the address family, you must bring up that interface manually using `ifconfig`. For example, if you configured a bridge like this:

```
auto bridge01
iface bridge01 inet manual
```

You can only bring it up by running `ifconfig bridge01 up`.



`ifdown` always deletes logical interfaces after bringing them down. Use the `--admin-state` option if you only want to administratively bring the interface up or down.

To see the link and administrative state, use the `ip link show` command:

```
cumulus@switch:~$ ip link show dev swp1
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP mode DEFAULT qlen 500
    link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
```

In this example, `swp1` is administratively UP and the physical link is UP (`LOWER_UP` flag). More information on interface administrative state and physical state can be found in [this knowledge base article](#).

## ifupdown2 Interface Classes

`ifupdown2` provides for the grouping of interfaces into separate classes, where a class is simply a user-defined label used to group interfaces that share a common function (like uplink, downlink or compute). You specify classes in `/etc/network/interfaces`.

The most common class users are familiar with is `auto`, which you configure like this:

```
auto swp1
iface swp1
```

You can add other classes using the `allow` prefix. For example, if you have multiple interfaces used for uplinks, you can make up a class called `uplinks`:

```
auto swp1
allow-uplink swp1
```

```
iface swp1 inet static
    address 10.1.1.1/31
```

```
auto swp2
allow-uplink swp2
iface swp2 inet static
    address 10.1.1.3/31
```

This allows you to perform operations on only these interfaces using the --allow-uplinks option, or still use the -a options since these interfaces are also in the auto class:

```
cumulus@switch:~$ sudo ifup --allow=uplinks
cumulus@switch:~$ sudo ifreload -a
```

Another example where this feature is useful is if you're using [Management VRF \(see page 489\)](#), you can use the special interface class called *mgmt*, and put the management interface into that class:

```
allow-mgmt eth0
iface eth0 inet dhcp
    vrf mgmt
allow-mgmt mgmt
iface mgmt
    address 127.0.0.1/8
    vrf-table auto
```

All *ifupdown2* commands (*ifup*, *ifdown*, *ifquery*, *ifreload*) can take a class. Include the --allow=<class> option when you run the command. For example, to reload the configuration for the management interface described above, run:

```
cumulus@switch:~$ sudo ifreload --allow=mgmt
```

## ***Bringing All auto Interfaces Up or Down***

You can easily bring up or down all interfaces marked with the common **auto** class in **/etc/network/interfaces**. Use the -a option. For further details, see individual man pages for **ifup(8)**, **ifdown(8)**, **ifreload(8)**.

To administratively bring up all interfaces marked auto, run:

```
cumulus@switch:~$ sudo ifup -a
```

To administratively bring down all interfaces marked auto, run:

```
cumulus@switch:~$ sudo ifdown -a
```

To reload all network interfaces marked `auto`, use the `ifreload` command, which is equivalent to running `ifdown` then `ifup`, the one difference being that `ifreload` skips any configurations that didn't change):

```
cumulus@switch:~$ sudo ifreload -a
```

## Configuring a Loopback Interface

Cumulus Linux has a loopback preconfigured in `/etc/network/interfaces`. When the switch boots up, it has a loopback interface, called `/o`, which is up and assigned an IP address of 127.0.0.1.

-  The loopback interface `/o` must always be specified in `/etc/network/interfaces` and must always be up.

### ifupdown Behavior with Child Interfaces

By default, `ifupdown` recognizes and uses any interface present on the system — whether a VLAN, bond or physical interface — that is listed as a dependent of an interface. You are not required to list them in the `interfaces` file unless they need a specific configuration, for MTU, link speed, and so forth (see page 149). And if you need to delete a child interface, you should delete all references to that interface from the `interfaces` file.

For this example, `swp1` and `swp2` below do not need an entry in the `interfaces` file. The following stanzas defined in `/etc/network/interfaces` provide the exact same configuration:

#### With Child Interfaces Defined

```
auto swp1
iface swp1

auto swp2
iface swp2

auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports swp1
    swp2
        bridge-vids 1-100
        bridge-pvid 1
        bridge-stp on
```

#### Without Child Interfaces Defined

```
auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports swp1
    swp2
        bridge-vids 1-100
        bridge-pvid 1
        bridge-stp on
```

**bridge-stp on**

### Bridge in Traditional Mode - Example

For this example, swp1.100 and swp2.100 below do not need an entry in the `interfaces` file. The following stanzas defined in `/etc/network/interfaces` provide the exact same configuration:

#### With Child Interfaces Defined

```
auto swp1.100
iface swp1.100
auto swp2.100
iface swp2.100
auto br-100
iface br-100
    address 10.0.12.2
/24
    address 2001:dad:
beef::3/64
    bridge-ports
swp1.100 swp2.100
bridge-stp on
```

#### Without Child Interfaces Defined

```
auto br-100
iface br-100
    address 10.0.12.2/2
4
    address 2001:dad:
beef::3/64
    bridge-ports swp1.1
00 swp2.100
bridge-stp on
```

For more information on the bridge in traditional mode vs the bridge in VLAN-aware mode, please read [this knowledge base article](#).

## ifupdown2 Interface Dependencies

`ifupdown2` understands interface dependency relationships. When `ifup` and `ifdown` are run with all interfaces, they always run with all interfaces in dependency order. When run with the interface list on the command line, the default behavior is to not run with dependents. But if there are any built-in dependents, they will be brought up or down.

To run with dependents when you specify the interface list, use the `--with-dependents` option. `--with-dependents` walks through all dependents in the dependency tree rooted at the interface you specify. Consider the following example configuration:

```
auto bond1
iface bond1
    address 100.0.0.2/16
    bond-slaves swp29 swp30

auto bond2
iface bond2
    address 100.0.0.5/16
```

```
bond-slaves swp31 swp32

auto br2001
iface br2001
    address 12.0.1.3/24
    bridge-ports bond1.2001 bond2.2001
    bridge-stp on
```

Using `ifup --with-dependents br2001` brings up all dependents of br2001: bond1.2001, bond2.2001, bond1, bond2, bond1.2001, bond2.2001, swp29, swp30, swp31, swp32.

```
cumulus@switch:~$ sudo ifup --with-dependents br2001
```

Similarly, specifying `ifdown --with-dependents br2001` brings down all dependents of br2001: bond1.2001, bond2.2001, bond1, bond2, bond1.2001, bond2.2001, swp29, swp30, swp31, swp32.

```
cumulus@switch:~$ sudo ifdown --with-dependents br2001
```

**!** As mentioned earlier, `ifdown2` always deletes logical interfaces after bringing them down. Use the `--admin-state` option if you only want to administratively bring the interface up or down. In terms of the above example, `ifdown br2001` deletes `br2001`.

To guide you through which interfaces will be brought down and up, use the `--print-dependency` option to get the list of dependents.

Use `ifquery --print-dependency=list -a` to get the dependency list of all interfaces:

```
cumulus@switch:~$ sudo ifquery --print-dependency=list -a
lo : None
eth0 : None
bond0 : ['swp25', 'swp26']
bond1 : ['swp29', 'swp30']
bond2 : ['swp31', 'swp32']
br0 : ['bond1', 'bond2']
bond1.2000 : ['bond1']
bond2.2000 : ['bond2']
br2000 : ['bond1.2000', 'bond2.2000']
bond1.2001 : ['bond1']
bond2.2001 : ['bond2']
br2001 : ['bond1.2001', 'bond2.2001']
swp40 : None
```

```
swp25 : None
swp26 : None
swp29 : None
swp30 : None
swp31 : None
swp32 : None
```

To print the dependency list of a single interface, use:

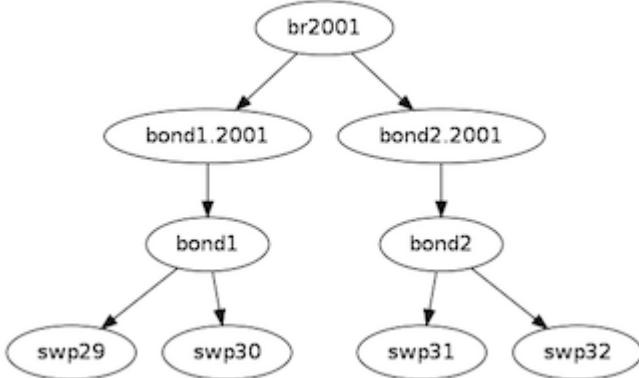
```
cumulus@switch:~$ sudo ifquery --print-dependency=list br2001
br2001 : ['bond1.2001', 'bond2.2001']
bond1.2001 : ['bond1']
bond2.2001 : ['bond2']
bond1 : ['swp29', 'swp30']
bond2 : ['swp31', 'swp32']
swp29 : None
swp30 : None
swp31 : None
swp32 : None
```

To print the dependency information of an interface in `dot` format:

```
cumulus@switch:~$ sudo ifquery --print-dependency=dot br2001
/* Generated by GvGen v.0.9 (http://software.inl.fr/trac/wiki/GvGen) */
digraph G {
    compound=true;
    node1 [label="br2001"];
    node2 [label="bond1.2001"];
    node3 [label="bond2.2001"];
    node4 [label="bond1"];
    node5 [label="bond2"];
    node6 [label="swp29"];
    node7 [label="swp30"];
    node8 [label="swp31"];
    node9 [label="swp32"];
    node1->node2;
    node1->node3;
    node2->node4;
    node3->node5;
    node4->node6;
    node4->node7;
    node5->node8;
```

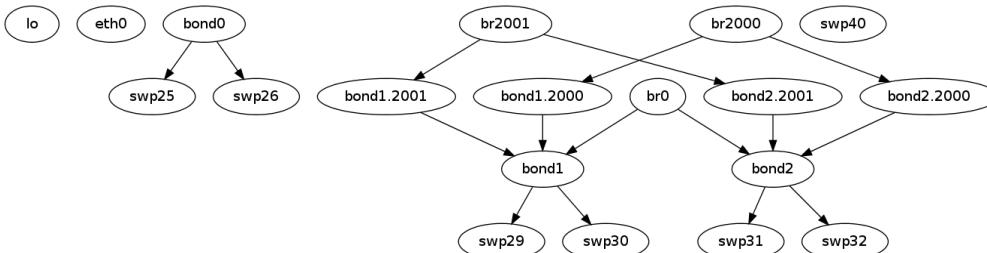
```
    node5->node9;
}
```

You can use `dot` to render the graph on an external system where `dot` is installed.



To print the dependency information of the entire `interfaces` file:

```
cumulus@switch:~$ sudo ifquery --print-dependency=dot -a >interfaces_all.dot
```



## ***ifup Handling of Upper (Parent) Interfaces***

When you run `ifup` on a logical interface (like a bridge, bond or VLAN interface), if the `ifup` resulted in the creation of the logical interface, by default it implicitly tries to execute on the interface's upper (or parent) interfaces as well. This helps in most cases, especially when a bond is brought down and up, as in the example below. This section describes the behavior of bringing up the upper interfaces.

Consider this example configuration:

```
auto br100
iface br100
    bridge-ports bond1.100 bond2.100

auto bond1
iface bond1
    bond-slaves swp1 swp2
```

If you run `ifdown bond1`, `ifdown` deletes bond1 and the VLAN interface on bond1 (bond1.100); it also removes bond1 from the bridge br100. Next, when you run `ifup bond1`, it creates bond1 and the VLAN interface on bond1 (bond1.100); it also executes `ifup br100` to add the bond VLAN interface (bond1.100) to the bridge br100.

As you can see above, implicitly bringing up the upper interface helps, but there can be cases where an upper interface (like br100) is not in the right state, which can result in warnings. The warnings are mostly harmless.

If you want to disable these warnings, you can disable the implicit upper interface handling by setting `skip_upperinterfaces=1` in `/etc/network/ifupdown2/ifupdown2.conf`.

With `skip_upperinterfaces=1`, you will have to explicitly execute `ifup` on the upper interfaces. In this case, you will have to run `ifup br100` after an `ifup bond1` to add bond1 back to bridge br100.



Although specifying a subinterface like swp1.100 and then running `ifup swp1.100` will also result in the automatic creation of the swp1 interface in the kernel, Cumulus Networks recommends you specify the parent interface swp1 as well. A parent interface is one where any physical layer configuration can reside, such as `link-speed 1000` or `link-duplex full`.

It's important to note that if you only create swp1.100 and not swp1, then you cannot run `ifup swp1` since you did not specify it.

## Configuring IP Addresses

In `/etc/network/interfaces`, list all IP addresses as shown below under the `iface` section (see `man interfaces` for more information):

```
auto swp1
iface swp1
    address 12.0.0.1/30
    address 12.0.0.2/30
```

The address method and address family are not mandatory. They default to `inet/inet6` and `static`, but `inet/inet6` **must** be specified if you need to specify `dhcp` or `loopback`:

```
auto lo
iface lo inet loopback
```

You can specify both IPv4 and IPv6 addresses in the same `iface` stanza:

```
auto swp1
iface swp1
    address 192.0.2.1/30
```

```
address 192.0.2.2/30
address 2001:DB8::1/126
```

## Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To make non-persistent changes to interfaces at runtime, use `ip addr add`:

```
cumulus@switch:~$ sudo ip addr add 192.0.2.1/30 dev swp1
cumulus@switch:~$ sudo ip addr add 2001:DB8::1/126 dev swp1
```

To remove an addresses from an interface, use `ip addr del`:

```
cumulus@switch:~$ sudo ip addr del 192.0.2.1/30 dev swp1
cumulus@switch:~$ sudo ip addr del 2001:DB8::1/126 dev swp1
```

See `man ip` for more details on the options available to manage and query interfaces.

To show the assigned address on an interface, use `ip addr show`:

```
cumulus@switch:~$ ip addr show dev swp1
3: swp1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 500
    link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/30 scope global swp1
        inet 192.0.2.2/30 scope global swp1
        inet6 2001:DB8::1/126 scope global tentative
            valid_lft forever preferred_lft forever
```

## Purging Existing IP Addresses on an Interface

By default, `ifupdown2` purges existing IP addresses on an interface. If you have other processes that manage IP addresses for an interface, you can disable this feature including the `address-purge` setting in the interface's configuration. For example, add the following to the interface configuration in `/etc/network/interfaces`:

```
auto swp1
iface swp1
    address-purge no
```



Purging existing addresses on interfaces with multiple `iface` stanzas is not supported. Doing so can result in the configuration of multiple addresses for an interface after you change an interface address and reload the configuration with `ifreload -a`. If this happens, you must shut down and restart the interface with `ifup` and `ifdown`, or manually delete superfluous addresses with `ip address delete specify.ip.address.here/mask dev DEVICE`. See also the [Caveats and Errata \(see page 148\)](#) section below for some cautions about using multiple `iface` stanzas for the same interface.

## Specifying User Commands

You can specify additional user commands in the `interfaces` file. As shown in the example below, the interface stanzas in `/etc/network/interfaces` can have a command that runs at pre-up, up, post-up, pre-down, down, and post-down:

```
auto swp1
iface swp1
    address 12.0.0.1/30
    up /sbin/foo bar
```

Any valid command can be hooked in the sequencing of bringing an interface up or down, although commands should be limited in scope to network-related commands associated with the particular interface.

For example, it wouldn't make sense to install some Debian package on `ifup` of `swp1`, even though that is technically possible. See `man interfaces` for more details.

## Sourcing Interface File Snippets

Sourcing interface files helps organize and manage the `interfaces(5)` file. For example:

```
cumulus@switch:~$ cat /etc/network/interfaces
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
```

```
auto eth0
iface eth0 inet dhcp

source /etc/network/interfaces.d/bond0
```

The contents of the sourced file used above are:

```
cumulus@switch:~$ cat /etc/network/interfaces.d/bond0
auto bond0
iface bond0
    address 14.0.0.9/30
    address 2001:ded:beef:2::1/64
    bond-slaves swp25 swp26
```

## Using Globs for Port Lists

Some modules support globs to define port lists (that is, a range of ports). You can use the `glob` keyword to specify bridge ports and bond slaves:

```
auto br0
iface br0
    bridge-ports glob swp1-6.100

auto br1
iface br1
    bridge-ports glob swp7-9.100  swp11.100 glob swp15-18.100
```

## Using Templates

`ifupdown2` supports Mako-style templates. The Mako template engine is run over the `interfaces` file before parsing.

Use the template to declare cookie-cutter bridges in the `interfaces` file:

```
%for v in [11,12]:
auto vlan${v}
iface vlan${v}
    address 10.20.${v}.3/24
    bridge-ports glob swp19-20.${v}
    bridge-stp on
%endfor
```

And use it to declare addresses in the `interfaces` file:

```
%for i in [1,12]:  
auto swp${i}  
iface swp${i}  
    address 10.20.${i}.3/24
```



Regarding Mako syntax, use square brackets (`[1,12]`) to specify a list of individual numbers (in this case, 1 and 12). Use `range(1,12)` to specify a range of interfaces.



You can test your template and confirm it evaluates correctly by running `mako-render /etc/network/interfaces`.



For more examples of configuring Mako templates, read this [knowledge base article](#).

## Adding Descriptions to Interfaces

You can add descriptions to the interfaces configured in `/etc/network/interfaces` by using the `alias` keyword. For example:

```
auto swp1  
iface swp1  
    alias swp1 hypervisor_port_1
```

You can query interface descriptions by running `ip link show`. The alias appears on the `alias` line:

```
cumulus@switch$ ip link show swp1  
3: swp1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast  
state DOWN mode DEFAULT qlen 500  
    link/ether aa:aa:aa:aa:bc brd ff:ff:ff:ff:ff:ff  
    alias hypervisor_port_1
```

Interface descriptions also appear in the SNMP OID (see page 511) IF-MIB::ifAlias.

 Aliases are limited to 256 characters.

## Caveats and Errata

While `ifupdown2` supports the inclusion of multiple `iface` stanzas for the same interface, Cumulus Networks recommends you use a single `iface` stanza for each interface, if possible.

There are cases where you must specify more than one `iface` stanza for the same interface. For example, the configuration for a single interface can come from many places, like a template or a sourced file.

If you do specify multiple `iface` stanzas for the same interface, make sure the stanzas do not specify the same interface attributes. Otherwise, unexpected behavior can result.

For example, `swp1` is configured in two places:

```
cumulus@switch:~$ cat /etc/network/interfaces

source /etc/interfaces.d/speed_settings

auto swp1
iface swp1
    address 10.0.14.2/24
```

As well as `/etc/interfaces.d/speed_settings`

```
cumulus@switch:~$ cat /etc/interfaces.d/speed_settings

auto swp1
iface swp1
    link-speed 1000
    link-duplex full
```

`ifupdown2` correctly parses a configuration like this because the same attributes are not specified in multiple `iface` stanzas.

And, as stated in the note above, you cannot purge existing addresses on interfaces with multiple `iface` stanzas.

## Useful Links

- <http://wiki.debian.org/NetworkConfiguration>
- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bonding>
- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge>

- <http://www.linuxfoundation.org/collaborate/workgroups/networking/vlan>

## Layer 1 and Switch Port Attributes

This chapter discusses the various network interfaces on a switch running Cumulus Linux.

### Contents

(Click to expand)

- Contents (see page 149)
- Commands (see page 149)
- Man Pages (see page 149)
- Configuration Files (see page 149)
- Interface Types (see page 150)
- Settings (see page 150)
  - Port Speed and Duplexing (see page 150)
  - Auto-negotiation (see page 152)
  - MTU (see page 152)
- Configuring Breakout Ports (see page 153)
  - Breaking Out a 100G Port (see page 154)
  - Breaking Out a 40G Port (see page 157)
  - Combining Four 10G Ports into One 40G Port (see page 158)
- Logical Switch Port Limitations (see page 159)
- Verification and Troubleshooting Commands (see page 160)
  - Statistics (see page 160)
  - Querying SFP Port Information (see page 160)
- Useful Links (see page 161)

### Commands

- ethtool
- ip

### Man Pages

- man ethtool
- man interfaces
- man ip
- man ip addr
- man ip link

## Configuration Files

- /etc/cumulus/ports.conf
- /etc/network/interfaces

## Interface Types

Cumulus Linux exposes network interfaces for several types of physical and logical devices:

- lo, network loopback device
- ethN, switch management port(s), for out of band management only
- swpN, switch front panel ports
- (optional) brN, bridges (IEEE 802.1Q VLANs)
- (optional) bondN, bonds (IEEE 802.3ad link aggregation trunks, or port channels)

## Settings

You can set the MTU, speed, duplex and auto-negotiation settings under a physical or logical interface stanza:

```
auto swp1
iface swp1
    address 10.1.1.1/24
    mtu 9000
    link-speed 10000
    link-duplex full
    link-autoneg off
```

To load the updated configuration, run the `ifreload -a` command:

```
cumulus@switch:~$ sudo ifreload -a
```

## Port Speed and Duplexing

Cumulus Linux supports both half- and **full-duplex** configurations. Supported port speeds include 1G, 10G and 40G. Set the speeds in terms of Mbps, where the setting for 1G is 1000, 10G is 10000 and 40G is 40000.

You can create a persistent configuration for port speeds in `/etc/network/interfaces`. Add the appropriate lines for each switch port stanza. For example:

```
auto swp1
iface swp1
    address 10.1.1.1/24
    link-speed 10000
    link-duplex full
```



If you specify the port speed in `/etc/network/interfaces`, you must also specify the duplex mode setting along with it; otherwise, `ethtool` defaults to half duplex.

You can also configure these settings at run time, using `ethtool`.

#### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

You can use `ethtool` to configure duplexing and the speed for your switch ports. You must specify both port speed and duplexing in the `ethtool` command; auto-negotiation is optional. The following examples use `swp1`.

- To set the port speed to 1G, run:

```
ethtool -s swp1 speed 1000 duplex full
```

- To set the port speed to 10G, run:

```
ethtool -s swp1 speed 10000 duplex full
```

- To enable duplexing, run:

```
ethtool -s swp1 speed 10000 duplex full|half
```

## Port Speed Limitations

Ports can be configured to one speed less than their maximum speed.

Switch port Type	Lowest Configurable Speed
1G	100 Mb
10G	1 Gigabit (1000 Mb)
40G	10G*
100G	50G (with or without breakout port), 40G*, 25G*, 10G*

\*Requires the port to be converted into a breakout port.

## Auto-negotiation

You can enable or disable **auto-negotiation** (that is, set it *on* or *off*) on a switch port.



Cumulus Linux does not support auto-negotiation for 10G or 40G interfaces.

```
auto swp1
iface swp1
    link-autoneg off
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

You can use **ethtool** to configure auto-negotiation for your switch ports. The following example use **swp1**:

- To enable or disable auto-negotiation, run:

```
ethtool -s swp1 speed 10000 duplex full autoneg on|off
```

## MTU

Interface MTU applies to the management port, front panel port, bridge, VLAN subinterfaces and bonds.

```
auto swp1
iface swp1
    mtu 9000
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To set **swp1** to Jumbo Frame MTU=9000, use **ip link set**:

```
cumulus@switch:~$ sudo ip link set dev swp1 mtu 9000
cumulus@switch:~$ ip link show dev swp1
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc pfifo_fast
    state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
```



You must take care to ensure there are no MTU mismatches in the conversation path. MTU mismatches will result in dropped or truncated packets, degrading or blocking network performance.

When you are configuring MTU for a bridge, don't set MTU on the bridge itself; set it on the individual members of the bridge. The MTU setting is the lowest MTU setting of any interface that is a member of that bridge (that is, every interface specified in `bridge-ports` in the bridge configuration in the `interfaces` file), even if another bridge member has a higher MTU value. Consider this bridge configuration:

```
auto br0
iface br0
    bridge-ports bond1 bond2 bond3 bond4 peer5
    bridge-vlan-aware yes
    bridge-vids 100-110
    bridge-stp on
```

In order for br0 to have an MTU of 9000, set the MTU for each of the member interfaces (bond1 to bond 4, and peer5), to 9000 at minimum.

```
auto peer5
iface peer5
    bond-slaves swp3 swp4
    mtu 9000
```

When configuring MTU for a bond, configure the MTU value directly under the bond interface; the configured value is inherited by member links.

To show MTU, use `ip link show`:

```
cumulus@switch:~$ ip link show dev swp1
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
```

## Configuring Breakout Ports

Cumulus Linux has the ability to:

- Break out 100G switch ports into the following with breakout cables:
  - 2x50G, 2x40G, 4x25G, 4x10G
- Break out 40G switch ports into four separate 10G ports for use with breakout cables.
- Combine (also called *aggregating* or *ganging*) four 10G switch ports into one 40G port for use with a breakout cable ([not to be confused with a bond \(see page 211\)](#)).

You configure breakout ports in the `/etc/cumulus/ports.conf` file. After you modify the configuration, restart `switchd` ([see page 128](#)) to push the new configuration; this [interrupts network services](#) ([see page 128](#)).



`/etc/cumulus/ports.conf` varies across different hardware platforms. Check the current list of supported platforms on the [hardware compatibility list](#).

### Breaking Out a 100G Port

A snippet from the `/etc/cumulus/ports.conf` looks on a Tomahawk switch like this:

```
# The Dell z9100 has:
#
#      32 QSFP28 ports numbered 1-32
#      These ports are configurable as 100G, 50G, 40G, 2x50G, 4x25G,
#      4x10G
#      or disabled.
#
#      Two SFP+ ports. These ports are configurable as 10G or disabled.
#
#      The system can only handle 128 logical ports.
#
#      This means that if all 32 QSFP28 ports are broken out into
#      4x25G or 4x10G mode, the two 10G ports (33 and 34) must be
#      set to "disabled".
# If you make changes to this file, you must restart switchd for the
# changes to take effect.
# QSFP28 ports
#
# <port label 1-32> = [4x10G|4x25G|2x50G|40G|50G|100G|disabled]
1=4x10G
2=100G
3=100G
4=100G
...
# SFP+ ports
#
# <port label 33-34> = [10G|disabled]
33=disabled
```

```
34=disabled
```

Notice that you can break out any of the 100G ports into a variety of options: four 10G ports, four 25G ports or two 50G ports. Keep in mind that you cannot have more than 128 total logical ports on the switch.

To change a 100G port to a number of logical ports, edit the `/etc/cumulus/ports.conf` file with a text editor (nano, vi, zile). Change `100G` to `4x10G`, `4x25G` or `2x50G`.

In the following example, port 3 is broken out into two logical 50G ports:

```
# QSFP28 ports
#
# <port label 1-32> = [4x10G|4x25G|2x50G|40G|50G|100G|disabled]
1=4x10G
2=100G
3=2x50G
4=100G
```

Similarly, the `ports.conf` file for a Mellanox Spectrum 100G switch looks like this:

```
# 
# mlnx,x86_MSN2700 has:
#   32 QSFP28 ports numbered 1-32
#       These ports are configurable as 40G, 50G, 2x50G, or 100G;
or a subset
#       of them can be split into 4x25G or 4x10G.
#
#       Note that a port may become disabled, i.e., unusable and
unconfigurable
#           in /etc/network/interfaces, when an adjacent port is split
into 4
#           interfaces. It is REQUIRED that the disabled port be
configured as
#           "disabled" in this file when an adjacent port is split into
4
#           interfaces.
#
# NOTE: When ports are split into 4 interfaces it is REQUIRED that
the adjacent
# disabled port be configured as "disabled" in this file. When
splitting a port
# into two interfaces, like 2x50G, it is NOT required that the
adjacent port be
# disabled. Adjacent ports only need to be disabled when a port is
split into
# 4 interfaces. For example, when splitting port 11 into 4 25G
interfaces, port
# 12 must be configued as "disabled" like this:
#
#   11=4x25G
```

```

#    12=disabled
#
#  The list of ports which can be split into 4 interfaces and the
# adjacent ports
#  which must be configured as "disabled" are:
#
#  1: 4x10G or 4x25G (would disable port 2)
#  3: 4x10G or 4x25G (would disable port 4)
#  5: 4x10G or 4x25G (would disable port 6)
#  7: 4x10G or 4x25G (would disable port 8)
#  9: 4x10G or 4x25G (would disable port 10)
# 11: 4x10G or 4x25G (would disable port 12)
# 13: 4x10G or 4x25G (would disable port 14)
# 15: 4x10G or 4x25G (would disable port 16)
# 17: 4x10G or 4x25G (would disable port 18)
# 19: 4x10G or 4x25G (would disable port 20)
# 21: 4x10G or 4x25G (would disable port 22)
# 23: 4x10G or 4x25G (would disable port 24)
# 25: 4x10G or 4x25G (would disable port 26)
# 27: 4x10G or 4x25G (would disable port 28)
# 29: 4x10G or 4x25G (would disable port 30)
# 31: 4x10G or 4x25G (would disable port 32)
#
# QSFP28 ports
#
# <port label>      = [40G|50G|100G]
#   or when split = [2x50G|4x10G|4x25G|disabled]
1=100G
2=100G
3=100G
4=100G
5=100G
...
30=100G
31=100G
32=100G

```

To change a 100G port to a number of logical ports, edit the `/etc/cumulus/ports.conf` file with a text editor (nano, vi, zile). Change 100G to 4x10G, 4x25G or 2x50G.

However, for both 100G and 40G switches using Mellanox Spectrum chipsets, there is a limit of 64 logical ports in total. You must configure the logical ports as follows:

- You can only break out odd-numbered ports into logical ports.
- You must disable the next even-numbered port.

For example, if you have a 100G Mellanox SN-2700 switch and configure port 11 as 4x25G logical ports, you must configure port 12 as disabled. In `/etc/cumulus/ports.conf`:

```
...
11=4x25G
12=disabled
...
```

In any case, when you finish editing `ports.conf`, make sure to [restart `switchd`](#) (see page 128) to reload your changes.

## ***Breaking Out a 40G Port***

A snippet from the `/etc/cumulus/ports.conf` looks on a 40G switch like this:

```
# QSFP+ ports
#
# <port label 49-52> = [4x10G|40G]
49=40G
50=40G
51=40G
52=40G
```

To change a 40G port to 4x10G ports, edit the `/etc/cumulus/ports.conf` file with a text editor (nano, vi, zile). Change `40G` to `4x10G`.

In the following example, switch port 49 is changed to a breakout port with four 10G logical ports:

```
# QSFP+ ports
#
# <port label 49-52> = [4x10G|40G]
49=4x10G
50=40G
51=40G
52=40G
```

To load the change [restart `switchd`](#) (see page 128).

Many services depend on `switchd`. It is highly recommended to restart Cumulus Linux if possible in this situation.



For both 100G and 40G switches using Mellanox Spectrum chipsets, there is a limit of 64 logical ports in total. However, the logical ports must be configured as follows:

- You can only break out odd-numbered ports into 4 logical ports.
- You must disable the next even-numbered port.

For example, if you have a 100G Mellanox SN-2700 switch and configure port 11 as 4x25G logical ports, you must configure port 12 as disabled. In `/etc/cumulus/ports.conf`:

```
...
11=4x25G
12=disabled
...
```

## Combining Four 10G Ports into One 40G Port

To gang (aggregate) four 10G ports into one 40G port for use with a breakout cable, you must edit `/etc/cumulus/ports.conf`.

-  `/etc/cumulus/ports.conf` varies across different hardware platforms. Check the current list of supported platforms on the [hardware compatibility list](#).

A snippet from the `/etc/cumulus/ports.conf` looks like this:

```
# SFP+ ports#
# <port label 1-48> = [10G|40G/4]
1=10G
2=10G
3=10G
4=10G
5=10G
```

To change four 10G ports into one 40G port, edit the `/etc/cumulus/ports.conf` file with a text editor (nano, vi, zile). Change `10G` to `40G/4` for every port being ganged.

In the following example, switch ports `swp1-4` are changed to a ganged port:

```
# SFP+ ports#
# <port label 1-48> = [10G|40G/4]
1=40G/4
2=40G/4
3=40G/4
4=40G/4
5=10G
```

To load the change, [restart `switchd`](#) (see page 128) to load your changes.

Many services depend on `switchd`. It is highly recommended to restart Cumulus Linux if possible in this situation.

-  • You must gang four 10G ports in sequential order. For example, you cannot gang `swp1`, `swp10`, `swp20` and `swp40` together.

- The ports must be in increments of four, with the starting port being swp1 (or swp5, swp9, or so forth); so you cannot gang swp2, swp3, swp4 and swp5 together.

## Logical Switch Port Limitations

100G and 40G switches with Spectrum, Tomahawk, Trident II and Trident II+ chipsets (check the [HCL](#)) can support a certain number of logical ports, depending upon the manufacturer.

Before you configure any logical/unganged ports on a switch, check the limitations listed in `/etc/cumulus/ports.conf`; this file is specific to each manufacturer.

For example, the Dell S6000 `ports.conf` file indicates the logical port limitation like this:

```
# ports.conf --
#
# This file controls port aggregation and subdivision.  For example,
QSFP+
# ports are typically configurable as either one 40G interface or four
# 10G/1000/100 interfaces.  This file sets the number of interfaces
per port
# while /etc/network/interfaces and ethtool configure the link speed f
or each
# interface.
#
# You must restart switchd for changes to take effect.
#
# The DELL S6000 has:
#     32 QSFP ports numbered 1-32
#     These ports are configurable as 40G, split into 4x10G ports or
#     disabled.
#
#     The X pipeline covers QSFP ports 1 through 16 and the Y pipeline
#     covers QSFP ports 17 through 32.
#
#     The Trident2 chip can only handle 52 logical ports per pipeline.
#
#     This means 13 is the maximum number of 40G ports you can ungang
#     per pipeline, with the remaining three 40G ports set to
#     "disabled".  The 13 40G ports become 52 unganged 10G ports, which
#     totals 52 logical ports for that pipeline.
#
```

The means the maximum number of ports for this Dell S6000 is 104.

For switches using Mellanox Spectrum chipsets, there is a limit of 64 logical ports in total. However, the logical ports must be configured in a specific way. See [the note \(see page 153\)](#) above.

## Verification and Troubleshooting Commands

### Statistics

High-level interface statistics are available with the `ip -s link` command:

```
cumulus@switch:~$ ip -s link show dev swp1
3: swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP mode DEFAULT qlen 500
    link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
      21780       242        0        0        0       242
    TX: bytes   packets   errors   dropped carrier collsns
      1145554     11325        0        0        0         0
```

Low-level interface statistics are available with `ethtool`:

```
cumulus@switch:~$ sudo ethtool -S swp1
NIC statistics:
HwIfInOctets: 21870
HwIfInUcastPkts: 0
HwIfInBcastPkts: 0
HwIfInMcastPkts: 243
HwIfOutOctets: 1148217
HwIfOutUcastPkts: 0
HwIfOutMcastPkts: 11353
HwIfOutBcastPkts: 0
HwIfInDiscards: 0
HwIfInL3Drops: 0
HwIfInBufferDrops: 0
HwIfInAclDrops: 0
HwIfInBlackholeDrops: 0
HwIfInDot3LengthErrors: 0
HwIfInErrors: 0
SoftInErrors: 0
SoftInDrops: 0
SoftInFrameErrors: 0
HwIfOutDiscards: 0
HwIfOutErrors: 0
HwIfOutQDrops: 0
HwIfOutNonQDrops: 0
SoftOutErrors: 0
SoftOutDrops: 0
SoftOutTxFifoFull: 0
HwIfOutQLen: 0
```

## Querying SFP Port Information

You can verify SFP settings using `ethtool -m`. The following example shows the output for 1G and 10G modules:

```
cumulus@switch:~# sudo ethtool -m | egrep '(swp|RXPower :|TXPower :|EthernetComplianceCode)'

swp1: SFP detected
    EthernetComplianceCodes : 1000BASE-LX
    RXPower : -10.4479dBm
    TXPower : 18.0409dBm

swp3: SFP detected
    10GEthernetComplianceCode : 10G Base-LR
    RXPower : -3.2532dBm
    TXPower : -2.0817dBm
```

## Useful Links

- <http://wiki.debian.org/NetworkConfiguration>
- <http://www.linuxfoundation.org/collaborate/workgroups/networking/vlan>
- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge>
- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bonding>

## Buffer and Queue Management

Hardware datapath configuration manages packet buffering, queueing, and scheduling in hardware. There are two configuration input files:

- `/etc/cumulus/datapath/traffic.conf`, which describes priority groups and assigns the scheduling algorithm and weights
- `/etc/bcm.d/datapath/datapath.conf`, which assigns buffer space and egress queues



Versions of these files prior to Cumulus Linux 2.1 are incompatible with Cumulus Linux 2.1 and later; using older files will cause `switchd` to fail to start and return an error that it cannot find the `/var/lib/cumulus/rc.datapath` file.

Each packet is assigned to an ASIC Class of Service (CoS) value based on the packet's priority value stored in the 802.1p (Class of Service) or DSCP (Differentiated Services Code Point) header field. The packet is assigned to a priority group based on the CoS value.

Priority groups include:

- *Control*: Highest priority traffic
- *Service*: Second-highest priority traffic
- *Lossless*: Traffic protected by priority flow control
- *Bulk*: All remaining traffic

A lossless traffic group is protected from packet drops by configuring the datapath to use priority pause. A lossless priority group requires a port group configuration, which specifies the ports configured for priority flow control and the additional buffer space assigned to each port for packets in the lossless priority group.

The scheduler is configured to use a hybrid scheduling algorithm. It applies strict priority to control traffic queues and a weighted round robin selection from the remaining queues. Unicast packets and multicast packets with the same priority value are assigned to separate queues, which are assigned equal scheduling weights.

Datapath configuration takes effect when you initialize `switchd`. Changes to the `traffic.conf` file require you to [restart `switchd` \(see page 128\)](#).

## Contents

(Click to expand)

- [Contents \(see page 162\)](#)
- [Commands \(see page 162\)](#)
- [Configuration Files \(see page 162\)](#)
- [Configuring Traffic Marking through ACL Rules \(see page 164\)](#)
- [Configuring Link Pause \(see page 165\)](#)
- [Useful Links \(see page 167\)](#)
- [Caveats and Errata \(see page 167\)](#)

## Commands

If you modify the configuration in the `/etc/cumulus/datapath/traffic.conf` file, you must [restart `switchd` \(see page 128\)](#) for the changes to take effect:

```
cumulus@switch:~$ sudo systemctl restart switchd.service
```

## Configuration Files

The following configuration applies to 10G, 40G, and 100G switches only ([any switch](#) on the Tomahawk, Trident II+ or Trident II platform).

- `/etc/cumulus/datapath/traffic.conf`: The datapath configuration file.

Sample `traffic.conf` file (Click to expand)

```
cumulus@switch:~$ cat /etc/cumulus/datapath/traffic.conf
#
# /etc/cumulus/datapath/traffic.conf
#
# packet header field used to determine the packet priority
level
# fields include {802.1p,
dscp}
```

```

traffic.packet_priority_source = 802.
1p
# remark packet priority
value
# fields include {802.1p,
none}
traffic.remark_packet_priority = none
# packet priority values assigned to each internal cos
value
# internal cos values {cos_0..
cos_7}
# (internal cos 3 has been reserved for CPU-generated
traffic)
# 802.1p values = {0..7}, dscp values = {0..63}
traffic.cos_0.packet_priorities = [0]
traffic.cos_1.packet_priorities = [1]
traffic.cos_2.packet_priorities = [2]
traffic.cos_3.packet_priorities = []
traffic.cos_4.packet_priorities = [3,4]
traffic.cos_5.packet_priorities = [5]
traffic.cos_6.packet_priorities = [6]
traffic.cos_7.packet_priorities = [7]
# priority groups
traffic.priority_group_list = [control, service, bulk]
# internal cos values assigned to each priority group
# each cos value should be assigned exactly once
# internal cos values {0..7}
priority_group.control.cos_list = [7]
priority_group.service.cos_list = [2]
priority_group.bulk.cos_list = [0,1,3,4,5,6]
# to configure a lossless priority group:
# -- uncomment the cos list config and and assign cos value(s)
# -- uncomment port_group_0 configurations and set the lossless flag,
buffer si
ze,
ports
# -- (currently only one traffic group is allowed, with port range 'al
lports')
# priority_group.lossless.cos_list = []
# lossless port group
# -- lossless flag
arranging in: tiled
arranging in: tiled
# -- buffer size in bytes for each port
# -- port group
# priority_group.lossless.lossless_flag = true
# priority_group.lossless.port_group_0.port_buffer_bytes = 4096
# priority_group.lossless.port_group_0.port_range = allports
# to configure pause on a group of ports:
# uncomment the link pause port group list
# add or replace a port group name to the list
# populate the port set, e.g.

```

```

#      swp1-swp4,swp8,swp50s0-swp50s3
# enable pause frame transmit and/or pause frame receive
# link pause
# link_pause.port_group_list = [port_group_0]
# link_pause.port_group_0.port_set = swp1-swp4,swp6
# link_pause.port_group_0.rx_enable = true
# link_pause.port_group_0.tx_enable = true
# scheduling algorithm: algorithm values = {dwrr}
scheduling.algorithm = dwrr
# traffic group scheduling weight
# weight values = {0..127}
# '0' indicates strict priority
priority_group.control.weight = 0
priority_group.service.weight = 32
priority_group.bulk.weight = 16
priority_group.lossless.weight = 16
# To turn on/off Denial of service (DOS) prevention checks
dos_enable = false
# To enable cut-through forwarding
cut_through_enable = true
# Enable resilient hashing
#resilient_hash_enable = FALSE
# Resilient hashing flowset entries per ECMP group
# Valid values - 64, 128, 256, 512, 1024
#resilient_hash_entries_ecmp = 128
# Enable symmetric hashing
#symmetric_hash_enable = TRUE
# Set sflow/sample ingress cpu packet rate and burst in packets/sec
# Values: {0..16384}
#sflow.rate = 16384
#sflow.burst = 16384
#Specify the maximum number of paths per route entry.
# Maximum paths supported is 200.
# Default value 0 takes the number of physical ports as the max path
size.
#ecmp_max_paths = 0

```

## Configuring Traffic Marking through ACL Rules

You can mark traffic for egress packets through `iptables` or `ip6tables` rule classifications. To enable these rules, you do one of the following:

- Mark DSCP values in egress packets.
- Mark 802.1p CoS values in egress packets.

To enable traffic marking, use `cl-acltool`. Add the `-p` option to specify the location of the policy file. By default, if you don't include the `-p` option, `cl-acltool` looks for the policy file in `/etc/cumulus/acl/policy.d/`.

The `iptables`-`/ip6tables`-based marking is supported via the following action extension:

```
-j SETQOS --set-dscp 10 --set-cos 5
```

You can specify one of the following targets for SETQOS:

Option	Description
-set-cos INT	Sets the datapath resource/queuing class value. Values are defined in <a href="#">IEEE_P802.1p</a> .
-set-dscp value	Sets the DSCP field in packet header to a value, which can be either a decimal or hex value.
-set-dscp-class class	Sets the DSCP field in the packet header to the value represented by the DiffServ class value. This class can be EF, BE or any of the CSxx or AFxx classes.



You can specify either `--set-dscp` or `--set-dscp-class`, but not both.

Here are two example rules:

```
[iptables]
-t mangle -A FORWARD --in-interface swp+ -p tcp --dport bgp -j SETQOS --set-
dscp 10 --set-cos 5

[ip6tables]
-t mangle -A FORWARD --in-interface swp+ -j SETQOS --set-dscp 10
```

You can put the rule in either the *mangle* table or the default *filter* table; the mangle table and filter table are put into separate TCAM slices in the hardware.

To put the rule in the mangle table, include `-t mangle`; to put the rule in the filter table, omit `-t mangle`.

## Configuring Link Pause

The PAUSE frame is a flow control mechanism that halts the transmission of the transmitter for a specified period of time. A server or other network node within the data center may be receiving traffic faster than it can handle it, thus the PAUSE frame. In Cumulus Linux, individual ports can be configured to execute link pause by:

- Transmitting pause frames when its ingress buffers become congested (TX pause enable) and/or
- Responding to received pause frames (RX pause enable).

Just like configuring buffer and queue management link pause is configured by editing `/etc/cumulus/datapath/traffic.conf`.

Here is an example configuration which turns of both types of link pause for swp2 and swp3:

```
# to configure pause on a group of ports:  
# uncomment the link pause port group list  
# add or replace a port group name to the list  
# populate the port set, e.g.  
# swp1-swp4,swp8,swp50s0-swp50s3  
# enable pause frame transmit and/or pause frame receive  
# link pause  
link_pause.port_group_list = [port_group_0]  
link_pause.port_group_0.port_set = swp2-swp3  
link_pause.port_group_0.rx_enable = true  
link_pause.port_group_0.tx_enable = true
```

A *port group* refers to one or more sequences of contiguous ports. Multiple port groups can be defined by:

- Adding a comma-separated list of port group names to the `port_group_list`.
- Adding the `port_set`, `rx_enable`, and `tx_enable` configuration lines for each port group.

You can specify the set of ports in a port group in comma-separated sequences of contiguous ports; you can see which ports are contiguous in `/var/lib/cumulus/porttab`. The syntax supports:

- A single port (`swp1s0` or `swp5`)
- A sequence of regular `swp` ports (`swp2-swp5`)
- A sequence within a breakout `swp` port (`swp6s0-swp6s3`)
- A sequence of regular and breakout ports, provided they are all in a contiguous range. For example:

```
...  
swp2  
swp3  
swp4  
swp5  
swp6s0  
swp6s1  
swp6s2  
swp6s3  
swp7  
...
```

Restart `switchd` (see page 128) to allow link pause configuration changes to take effect:

```
cumulus@switch:~$ sudo systemctl restart switchd.service
```

## ***Useful Links***

- [iptables-extensions man page](#)

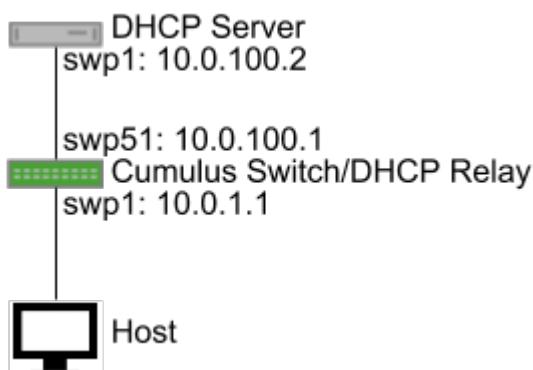
## ***Caveats and Errata***

- You can configure Quality of Service (QoS) for 10G, 40G, and 100G switches only; that is, any switch on the Tomahawk, Trident II+ or Trident II platform.

## **Configuring DHCP Relays**

You can configure an interface so it can make DHCP relay requests for IPv4 and IPv6.

To run DHCP for both IPv4 and IPv6, you need to initiate the DHCP relay and DHCP server twice: once for IPv4 using the **-4** option, and once for IPv6 using the **-6** option. Following are the configurations on the host, leaf and DHCP server using the following topology:



Contents

- [Configuring the Relays \(see page 167\)](#)

## ***Configuring the Relays***

Here is the host configuration:

```

/etc/network/interfaces
auto eth1
iface eth1 inet dhcp

auto eth1
iface eth1 inet6 dhcp
  
```

Here is the leaf configuration:

```
cumulus@switch:~$ /usr/sbin/dhcrelay -4 -i swp1 10.0.100.2 -i swp51
cumulus@switch:~$ /usr/sbin/dhcrelay -6 -i swp1 -u 2001:db8:100::2%swp51
```

You have to run two independent instances of `dhcrelay`. The `dhcrelay` feature is part of the `isc-dhcp-relay` services. The format of this command is below:

```
cumulus@switch:~$ /usr/sbin/dhcrelay -4 -i <interface_facing_host>
<ip_address_dhcp_server> -i <interface_facing_dhcp_server>
cumulus@switch:~$ /usr/sbin/dhcrelay -6 -i <interface_facing_host> -u
<ip_address_dhcp_server>%<interface_facing_dhcp_server>
```

See the `man dhcrelay` for more information.

The `systemd` launch scripts can be edited so that the `dhcrelay` service starts with the switch.

The launch scripts for `systemd` are located in `/lib/systemd/system`. Edit or create two files as described below:

```
cumulus@leaf01:/lib/systemd/system$ cat dhcrelay.service
[Unit]
Description=DHCPv4 Relay Agent Daemon
Documentation=man:dhcrelay(8)
After=network-online.target networking.service syslog.service

[Service]
Type=simple
EnvironmentFile=-/etc/default/isc-dhcp-relay
# Here, we are expecting the INTF_CMD to contain
# the -i for each interface specified,
ExecStart=/usr/sbin/dhcrelay -d -q $INTF_CMD $SERVERS $OPTIONS

[Install]
WantedBy=multi-user.target
```

```
cumulus@leaf01:/lib/systemd/system$ cat /etc/default/isc-dhcp-relay
SERVERS="10.0.100.2"

INTF_CMD="-i swp1 -i swp51"
```

```
cumulus@leaf01:/lib/systemd/system$ cat dhcrelay6.service
[Unit]
```

```

Description=DHCPv6 Relay Agent Daemon
Documentation=man:dhcrelay(8)
After=network-online.target networking.service syslog.service

[Service]
Type=simple
EnvironmentFile=/etc/default/isc-dhcp-relay6
ExecStart=/usr/sbin/dhcrelay -6 -d -q $INTF_CMD $SERVERS $OPTIONS

[Install]
WantedBy=multi-user.target

```

```

cumulus@leaf01:/lib/systemd/system$ cat /etc/default/isc-dhcp-relay6
SERVERS=" -u 2001:db8:100::2%swp51"

INTF_CMD="-l swp1"

```

Here is the DHCP server configuration:

```

/usr/sbin/dhcpd -6 -cf /etc/dhcp/dhcpd6.conf swp1
/usr/sbin/dhcpd -4 -cf /etc/dhcp/dhcpd.conf swp1

```

The configuration files for the two DHCP servers need to have two pools:

- Pool 1: Subnet overlaps interfaces
- Pool 2: Subnet that includes the addresses

Here are the sample configurations:

```

/etc/dhcp/dhcpd.conf
subnet 10.0.100.0 netmask 255.255.255.0 {
}
subnet 10.0.1.0 netmask 255.255.255.0 {
    range 10.0.1.50 10.0.1.60;
}

```

```

/etc/dhcp/dhcpd6.conf
subnet6 2001:db8:100::/64 {
}

```

```
subnet6 2001:db8:1::/64 {
    range6 2001:db8:1::100 2001:db8:1::200;
}
```

Just as you did with the DHCP relay scripts, the `systemd` initialization scripts can be used to launch the DHCP server at start. Here are sample configurations:

```
cumulus@spine01:/lib/systemd/system$ cat dhcpcd.service
[Unit]
Description=DHCPv4 Server Daemon
Documentation=man:dhcpcd(8) man:dhcpcd.conf(5)
After=network-online.target networking.service syslog.service

[Service]
Type=simple
EnvironmentFile=-/etc/default/isc-dhcp-server
ExecStart=/usr/sbin/dhcpcd -f -q $DHCPD_CONF $DHCPD_PID $INTERFACES $OPTIONS

[Install]
WantedBy=multi-user.target
```

```
cumulus@spine01:/lib/systemd/system$ cat /etc/default/isc-dhcp-server
DHCPD_CONF="-cf /etc/dhcp/dhcpcd.conf"

INTERFACES="swp1"
```

```
cumulus@spine01:/lib/systemd/system$ cat /etc/dhcp/dhcpcd.conf
ddns-update-style none;

default-lease-time 600;
max-lease-time 7200;

subnet 10.0.100.0 netmask 255.255.255.0 {
}
subnet 10.0.1.0 netmask 255.255.255.0 {
    range 10.0.1.50 10.0.1.60;
}
```

```
cumulus@spine01:/lib/systemd/system$ cat dhcpd6.service
[Unit]
Description=DHCPv6 Server Daemon
Documentation=man:dhcpd(8) man:dhcpd.conf(5)
After=network-online.target networking.service syslog.service

[Service]
Type=simple
EnvironmentFile=/etc/default/isc-dhcp-server6
ExecStart=/usr/sbin/dhcpd -f -q -6 $DHCPD_CONF $DHCPD_PID $INTERFACES
$OPTIONS

[Install]
WantedBy=multi-user.target
```

```
cumulus@spine01:/lib/systemd/system$ cat /etc/default/isc-dhcp-server6
DHCPD_CONF="-cf /etc/dhcp/dhcpd6.conf"

INTERFACES="swp1"
```

```
cumulus@spine01:/lib/systemd/system$ cat /etc/dhcp/dhcpd6.conf
ddns-update-style none;

default-lease-time 600;
max-lease-time 7200;

subnet6 2001:db8:100::/64 {
}
subnet6 2001:db8:1::/64 {
    range6 2001:db8:1::100 2001:db8:1::200;
}
```

# Layer 1 and Layer 2 Features

## Spanning Tree and Rapid Spanning Tree

Spanning tree protocol (STP) is always recommended in layer 2 topologies, as it prevents bridge loops and broadcast radiation on a bridged network.

The `mstpd` daemon is an open source project used by Cumulus Linux to implement IEEE802.1D 2004 and IEEE802.1Q 2011. `mstptcl` is the utility provided by the `mstpd` service to configure STP. STP is disabled by default on bridges in Cumulus Linux.

### Contents

This chapter covers:

- Commands (see page 172)
- Supported Modes (see page 173)
- Configuring STP within a Traditional Mode Bridge (see page 173)
  - Creating a Bridge and Configuring PVRST (see page 173)
- Configuring STP within a VLAN-aware Bridge (see page 174)
  - Creating a VLAN-aware Bridge and Configuring RSTP (see page 174)
  - RSTP Interoperation with MST (802.1s) (see page 174)
- Viewing Bridge and STP Status/Logs (see page 174)
- Customizing Spanning Tree Protocol (see page 176)
  - PortAdminEdge/PortFast Mode (see page 176)
  - BPDU Guard (see page 177)
    - Configuring BPDU Guard (see page 177)
    - Recovering a Port Disabled by BPDU Guard (see page 178)
  - Bridge Assurance (see page 180)
  - BPDU Filter (see page 180)
  - Storm Control (see page 181)
  - Example Configuration with All Possible Parameters (see page 182)
  - Configuring Other Spanning Tree Parameters (see page 183)
- Configuration Files (see page 191)
- Man Pages (see page 191)
- Useful Links (see page 191)
- Caveats and Errata (see page 191)

### Commands

- brctl
- mstptcl

## Supported Modes

The STP modes Cumulus Linux supports vary depending upon whether the [traditional or VLAN-aware bridge driver mode](#) (see page 214) is in use.

For a bridge configured in *traditional* mode, PVST and PVRST are supported, with the default set to PVRST. Each traditional bridge has its own separate STP instance.

Bridges configured in *VLAN-aware* (see page 235) mode operate **only** in RSTP mode.

## Configuring STP within a Traditional Mode Bridge

### Creating a Bridge and Configuring PVRST

Per VLAN Spanning Tree (PVST) creates a spanning tree instance for a bridge. Rapid PVST (PVRST) supports RSTP enhancements for each spanning tree instance. In order to use PVRST with a traditional bridge, a bridge corresponding to the untagged native VLAN must be created, and all the physical switch ports must be part of the same VLAN.



When connected to a switch that has a native VLAN configuration, the native VLAN **must** be configured to be VLAN 1 only for maximum interoperability.

To create a traditional mode bridge, configure the bridge stanza under `/etc/network/interfaces`. More information on configuring [bridges can be found here](#). (see page 214) To enable STP on the bridge, include the keyword `bridge-stp on`. `swp1` and `swp5` are configured for tagging VLAN 100, while `swp4` is configured to not tag a VLAN across the link.

```
auto br100
iface br100
  bridge-ports swp1.100 swp5.100
  bridge-stp on

auto br1
iface br1
  bridge-ports swp1 swp4 swp5
  bridge-stp on
```

To enable the bridge and load the new configuration from `/etc/network/interfaces`, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

You use `brctl` to create the bridge, add bridge ports in the bridge and configure STP on the bridge. `mstptcl` is used only when an admin needs to change the default configuration parameters for STP:

```
cumulus@switch:~$ sudo brctl addbr br100
cumulus@switch:~$ sudo brctl addif br100 swp1.100 swp5.100
cumulus@switch:~$ sudo brctl stp br100 on
cumulus@switch:~$ sudo ifconfig br100 up
```

## Configuring STP within a VLAN-aware Bridge

*VLAN-aware* (see page 235) bridges only operate in RSTP mode. STP BPDUs are transmitted on the native VLAN.

If a bridge running RSTP (802.1w) receives a common STP (802.1D) BPDU, it will automatically fall back to 802.1D operation.

## Creating a VLAN-aware Bridge and Configuring RSTP

To create a VLAN-aware mode bridge, configure the bridge stanza under `/etc/network/interfaces`.

```
auto br2
iface br2
  bridge-vlan-aware yes
  bridge-vids 100
  bridge-pvid  1
  bridge-ports swp1 swp4 swp5
  bridge-stp on
```

To enable the bridge and load the new configuration from `/etc/network/interfaces`, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

## RSTP Interoperation with MST (802.1s)

RSTP interoperates with MST seamlessly, creating a single instance of spanning tree which transmitts BPDUs on the native VLAN. RSTP treats the MST domain as if it were one giant switch.

## Viewing Bridge and STP Status/Logs

`mstpd` is started by default when the switch boots. `mstpd` logs and errors are located in `/var/log/syslog`.



`mstpd` is the preferred utility for interacting with STP on Cumulus Linux. `brctl` also provides certain methods for configuring STP; however, they are not as complete as the tools offered in `mstpd` and [output from brctl can be misleading](#) in some cases.

To get the bridge state, use:

```
cumulus@switch:~$ sudo brctl show
bridge name      bridge id          STP enabled    interfaces
  br2            8000.001401010100    yes           swp1
                                         swp4
                                         swp5
```

To get the `mstpd` bridge state, use:

```
cumulus@switch:~$ sudo mstptctl showbridge br2
br2 CIST info
  enabled        yes
  bridge id     F.000.00:14:01:01:01:00
  designated root F.000.00:14:01:01:01:00
  regional root F.000.00:14:01:01:01:00
  root port     none
  path cost      0      internal path cost  0
  max age        20     bridge max age       20
  forward delay  15     bridge forward delay 15
  tx hold count 6      max hops             20
  hello time     2      ageing time         200
  force protocol version   rstp
  time since topology change 90843s
  topology change count     4
  topology change          no
  topology change port      swp4
  last topology change port swp5
```

To get the `mstpd` bridge port state, use:

```
cumulus@switch:~$ sudo mstptctl showport br2
E swp1 8.001 forw F.000.00:14:01:01:01:00 F.000.00:14:01:01:01:00 8.0
01 Desg
  swp4 8.002 forw F.000.00:14:01:01:01:00 F.000.00:14:01:01:01:00 8.0
02 Desg
  E swp5 8.003 forw F.000.00:14:01:01:01:00 F.000.00:14:01:01:01:00 8.0
03 Desg

cumulus@switch:~$ sudo mstptctl showportdetail br2 swp1
br2:swp1 CIST info
  enabled        yes           role
  Designated
```

port id	8.001	state
forwarding		
external port cost	2000	admin external cost 0
internal port cost	2000	admin internal cost 0
designated root	F.000.00:14:01:01:01:00	dsgn external cost 0
dsgn regional root	F.000.00:14:01:01:01:00	dsgn internal cost 0
designated bridge	F.000.00:14:01:01:01:00	designated port 8.00
1		
admin edge port	no	auto edge port yes
oper edge port	yes	topology change ack no
point-to-point	yes	admin point-to-point auto
restricted role	no	restricted TCN no
port hello time	2	disputed no
bpdu guard port	no	bpdu guard error no
network port	no	BA inconsistent no
Num TX BPDU	45772	Num TX TCN 4
Num RX BPDU	0	Num RX TCN 0
Num Transition FWD	2	Num Transition BLK 2

## Customizing Spanning Tree Protocol

There are a number of ways you can customize STP in Cumulus Linux. You should exercise extreme caution with many of the settings below to prevent malfunctions in STP's loop avoidance.

### PortAdminEdge/PortFast Mode

`PortAdminEdge` is equivalent to the PortFast feature offered by other vendors.

All ports configured with `PortAdminEdge` bypass the listening and learning states to move immediately to forwarding.



Using `PortAdminEdge` mode has the potential to cause loops if it is not accompanied by the BPDU guard feature. All examples below include BPDU guard.

While it is common for edge ports to be configured as access ports for a simple end host, this is not mandatory. In the data center, edge ports typically connect to servers, which may pass both tagged and untagged traffic.

For the bridge, configure `PortAdminEdge` under the bridge stanza in `/etc/network/interfaces`:

```
auto br2
iface br2 inet static
  bridge-ports swp1 swp2 swp3 swp4
  bridge-stp on
  mstptctl-bpduguard swp1=yes swp2=yes swp3=yes swp4=yes
  mstptctl-portadmineedge swp1=yes swp2=yes swp3=yes swp4=yes
```

For each interface, configure `PortAdminEdge` under a switch port interface stanza in `/etc/network/interfaces`:

```
auto swp5
iface swp5
    mstpctl-bpduguard yes
    mstpctl-portadmineedge yes
```

To load the new configuration, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

```
cumulus@switch:~$ sudo mstpctl setportadmineedge br2 swp1 yes
cumulus@switch:~$ sudo mstpctl setbpduguard br2 swp1 yes
```

## **BPDU Guard**

To protect the spanning tree topology from unauthorized switches affecting the forwarding path, you can configure *BPDU guard* (Bridge Protocol Data Unit). One very common example is when someone hooks up a new switch to an access port off of a leaf switch. If this new switch is configured with a low priority, it could become the new root switch and affect the forwarding path for the entire layer 2 topology.

### **Configuring BPDU Guard**

For the bridge, configure BPDU guard under the bridge stanza in `/etc/network/interfaces`:

```
auto br2
iface br2 inet static
    bridge-ports swp1 swp2 swp3 swp4 swp5 swp6
    bridge-stp on
    mstpctl-bpduguard swp1=yes swp2=yes swp3=yes swp4=yes
```

For each interface, configure BPDU guard under an interface stanza in `/etc/network/interfaces`:

```
auto swp5
iface swp5
    mstpctl-bpduguard yes
```

To load the new configuration, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

```
cumulus@switch:~$ sudo mstpcctl setbpduGuard br2 swp1 yes
cumulus@switch:~$ sudo mstpcctl setbpduGuard br2 swp2 yes
cumulus@switch:~$ sudo mstpcctl setbpduGuard br2 swp3 yes
cumulus@switch:~$ sudo mstpcctl setbpduGuard br2 swp4 yes
```

### **Recovering a Port Disabled by BPDU Guard**

If a BPDU is received on the port, STP will bring down the port and log an error in `/var/log/syslog`. The following is a sample error:

```
mstpd: error, MSTP_IN_rx_bpdu: bridge:bond0 Recvd BPDU on BPDU Guard
Port - Port Down
```

To determine whether BPDU guard is configured, or if a BPDU has been received, run `mstpcctl showportdetail <bridge name>`:

```
cumulus@switch:~$ sudo mstpcctl showportdetail br2 swp1 | grep guard
bpdu guard port      yes                      bpdu guard error      yes
```

The only way to recover a port that has been placed in the disabled state is to manually un-shut or bring up the port with `sudo ifup [port]`, as shown in the example below:



Bringing up the disabled port does not fix the problem if the configuration on the connected end-station has not been rectified.

```
cumulus@leaf2$ mstpcctl showportdetail bridge bond0
bridge:bond0 CIST info
  enabled          no                  role
Disabled
  port id        8.001                state
discarding
  external port cost 305           admin external cost  0
  internal port cost 305          admin internal cost  0
```

```

designated root      8.000.6C:64:1A:00:4F:9C dsgn external cost      0
dsgn regional root  8.000.6C:64:1A:00:4F:9C dsgn internal cost      0
designated bridge   8.000.6C:64:1A:00:4F:9C designated port      8.00
1
admin edge port     no                         auto edge port      yes
oper edge port      no                         topology change ack no
point-to-point       yes                        admin point-to-point auto
restricted role     no                         restricted TCN      no
port hello time    10                         disputed          no
bpdu guard port    yes                        bpdu guard error   yes
network port        no                         BA inconsistent   no
Num TX BPDU         3                          Num TX TCN        2
Num RX BPDU         488                        Num RX TCN        2
Num Transition FWD 1                          Num Transition BLK 2
bpdufilter port    no                         clag ISL Oper UP  no
clag role           unknown                     clag dual conn mac 0:0:
0:0:0:0
      clag remote portID F.FFF                  clag system mac   0:0:
0:0:0:0

```

```
cumulus@leaf2$ sudo ifup bond0
```

```

cumulus@leaf2$ mstpcctl showportdetail bridge bond0
bridge:bond0 CIST info
  enabled      yes             role          Root
  port id     8.001            state
forwarding
  external port cost 305      admin external cost 0
  internal port cost 305      admin internal cost 0
  designated root     8.000.6C:64:1A:00:4F:9C dsgn external cost 0
  dsgn regional root  8.000.6C:64:1A:00:4F:9C dsgn internal cost 0
  designated bridge   8.000.6C:64:1A:00:4F:9C designated port 8.00
1
  admin edge port     no                         auto edge port      yes
  oper edge port      no                         topology change ack no
  point-to-point       yes                        admin point-to-point auto
  restricted role     no                         restricted TCN      no
  port hello time    2                          disputed          no
  bpdu guard port    no                         bpdu guard error   no
  network port        no                         BA inconsistent   no
  Num TX BPDU         3                          Num TX TCN        2
  Num RX BPDU         43                         Num RX TCN        1
  Num Transition FWD 1                          Num Transition BLK 0
  bpdufilter port    no                         clag ISL Oper UP  no
  clag role           unknown                     clag dual conn mac 0:0:
0:0:0:0
      clag remote portID F.FFF                  clag system mac   0:0:
0:0:0:0

```

## Bridge Assurance

On a point-to-point link where RSTP is running, if you want to detect unidirectional links and put the port in a discarding state (in error), you can enable bridge assurance on the port by enabling a port type network. The port would be in a bridge assurance inconsistent state until a BPDU is received from the peer. You need to configure the port type network on both the ends of the link.

A persistent configuration for bridge assurance looks like the example below:

```
mstpctl-portnetwork swp1=no
```

You can monitor logs for bridge assurance messages by doing the following:

```
cumulus@switch:~$ sudo grep -in assurance /var/log/syslog | grep mstp
1365:Jun 25 18:03:17 mstpd: br1007:swp1.1007 Bridge assurance
inconsistent
```

To load the new configuration from `/etc/network/interfaces`, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

Runtime Configuration (Advanced)

! A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To enable bridge assurance at runtime, run `mstpctl`:

```
cumulus@switch:~$ sudo mstpctl setportnetwork br1007 swp1.1007 yes
cumulus@switch:~$ sudo mstpctl showportdetail br1007 swp1.1007 | grep
network
  network port          yes           BA inconsistent      yes
```

## BPDU Filter

You can enable `bpdufilter` on a switch port, which filters BPDUs in both directions. This effectively disables STP on the port as no BPDUs are transiting.

! Using BDPU filter inappropriately can cause layer 2 loops. Use this feature deliberately and with extreme caution.

For the bridge, enable BPDU filter persistently by adding the following to `/etc/network/interfaces` under the `bridge port iface` section example:

```
auto br100
iface br100
    bridge-ports swp1.100 swp2.100
    mstpctl-portbpdufilter swp1=yes swp2=yes
```

For each interface, it is also possible to enable BPDU filter persistently for a specific port with the following configuration:

```
auto swp6
iface swp6
    mstpctl-portbpdufilter yes
```

To load the new configuration from `/etc/network/interfaces`, run `ifreload -a`:

```
cumulus@switch:~$ sudo ifreload -a
```

For more information, see `man(5) ifupdown-addons-interfaces`.

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To enable BPDU filter at runtime, run `mstpctl`:

```
cumulus@switch:~$ sudo mstpctl setportbpdufilter br100 swp1.100=yes swp2.100=yes
```

## Storm Control

*Storm control* provides protection against excessive inbound BUM (broadcast, unknown unicast, multicast) traffic on layer 2 switch port interfaces, which can cause poor network performance.

You configure storm control for each physical port in one of three ways:

- By editing `/etc/cumulus/switchd.conf`. The configuration persists across reboots and restarting `switchd`. If you change the storm control configuration in this file after rebooting the switch, you must `restart switchd` (see page 128) to activate the new configuration.
- By editing `/etc/network/interfaces`, which requires you to reload the interface configuration for the change to take effect.
- By writing directly to the `switchd` file system (see page 126).

For example, to enable broadcast and multicast storm control at 400 packets per second (pps) and 3000 pps, respectively, for swp1 editing `/etc/cumulus/switchd.conf`, configure it as follows:

```
# Storm Control setting on a port, in pps, 0 means disable
interface.swp1.storm_control.broadcast = 400
interface.swp1.storm_control.multicast = 3000
```

To configure these settings in `/etc/network/interfaces`:

```
auto swp1
iface swp1
    post-up echo 400 > /cumulus/switchd/config/interface/$IFACE
    /storm_control/broadcast
    post-up echo 3000 > /cumulus/switchd/config/interface/$IFACE
    /storm_control/multicast
    post-down echo 0 > /cumulus/switchd/config/interface/$IFACE
    /storm_control/broadcast
    post-down echo 0 > /cumulus/switchd/config/interface/$IFACE
    /storm_control/multicast
```

Then, reload the configuration:

```
cumulus@switch:$ sudo ifreload -a
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

Finally, if you are not in a position to restart `switchd`, you can change your storm control settings at runtime, which take effect immediately. For example, to set the pps on swp1 to 400 pps for broadcast traffic and 3000 for multicast traffic:

```
cumulus@switch:$ sudo cl-cfg -w switchd interface.swp1.storm_control.
broadcast=400
cumulus@switch:$ sudo cl-cfg -w switchd interface.swp1.storm_control.
multicast=3000
```

## Example Configuration with All Possible Parameters

The persistent configuration for a bridge is set in `/etc/network/interfaces`. The configuration below shows every possible option configured. There is no requirement to configure any of these options:

```

auto br2
iface br2 inet static
    bridge-ports swp1 swp2 swp3 swp4
    bridge-stp on
    mstpcctl-maxage 20
    mstpcctl-ageing 300
    mstpcctl-fdelay 15
    mstpcctl-maxhops 20
    mstpcctl-txholdcount 6
    mstpcctl-forcevers rstp
    mstpcctl-treepriority 32768
    mstpcctl-treeportpriority swp3=128
    mstpcctl-hello 2
    mstpcctl-portpathcost swp1=0 swp2=0
    mstpcctl-portadminedge swp1=no swp2=no
    mstpcctl-portautoedge swp1=yes swp2=yes
    mstpcctl-portp2p swp1=no swp2=no
    mstpcctl-portrestrole swp1=no swp2=no
    mstpcctl-portrestrcn swp1=no swp2=no
    mstpcctl-portnetwork swp1=no
    mstpcctl-bpduguard swp1=no swp2=no
    mstpcctl-bpdufilter swp4=yes

```

## Configuring Other Spanning Tree Parameters

Spanning tree parameters are defined in the IEEE 802.1D, 802.1Q specifications and in the table below.

While configuring spanning tree in a persistent configuration, as described above, is the preferred method, you can also use `mstpcctl` to configure spanning tree protocol parameters at runtime.



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

For a comparison of STP parameter configuration between `mstpcctl` and other vendors, [please read this knowledge base article](#).

Examples are included below:

Parameter	Description
<b>maxage</b>	<p>Sets the bridge's <i>maximum age</i> to &lt;max_age&gt; seconds. The default is 20.</p> <p>The maximum age must meet the condition <math>2 * (\text{Bridge Forward Delay} - 1 \text{ second}) \geq \text{Bridge Max Age}</math>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre><code>mstpcctl-maxage 24</code></pre> <p>To set this parameter at runtime, use:</p>

Parameter	Description
	<pre>mstpctl setmaxage &lt;bridge&gt; &lt;max_age&gt;</pre> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>cumulus@switch:~\$ sudo mstpctl setmaxage br2 24</pre> </div>
<b>ageing</b>	<p>Sets the Ethernet (MAC) address <i>ageing time</i> in &lt;time&gt; seconds for the bridge when the running version is STP, but not RSTP/MSTP. The default is 300.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>mstpctl-ageing 240</pre> </div> <p>To set this parameter at runtime, use:</p> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>mstpctl setageing &lt;bridge&gt; &lt;time&gt;</pre> </div> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>cumulus@switch:~\$ sudo mstpctl setageing br2 240</pre> </div>
<b>fdelay</b>	<p>Sets the bridge's <i>bridge forward delay</i> to &lt;time&gt; seconds. The default is 15.</p> <p>The bridge forward delay must meet the condition <math>2 * (\text{Bridge Forward Delay} - 1 \text{ second}) \geq \text{Bridge Max Age}</math>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>mstpctl-fdelay 15</pre> </div> <p>To set this parameter at runtime, use:</p> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>mstpctl setfdelay &lt;bridge&gt; &lt;time&gt;</pre> </div> <div style="background-color: #f0f0f0; padding: 10px;"> <pre>cumulus@switch:~\$ sudo mstpctl setfdelay br2 15</pre> </div>
<b>maxhops</b>	<p>Sets the bridge's <i>maximum hops</i> to &lt;max_hops&gt;. The default is 20.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p>

Parameter	Description
	<p><b>mstpctl-maxhops 24</b></p> <p>To set this parameter at runtime, use:</p> <pre data-bbox="437 439 1465 544"><b>mstpctl setmaxhops &lt;bridge&gt; &lt;max_hops&gt;</b></pre> <pre data-bbox="437 587 1465 692"><b>cumulus@switch:~\$ sudo mstpctl setmaxhops br2 24</b></pre>
<b>txholdcount</b>	<p>Sets the bridge's <i>bridge transmit hold count</i> to &lt;tx_hold_count&gt;. The default is 6.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="437 861 1465 967"><b>mstpctl-txholdcount 6</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="437 1009 1465 1115"><b>mstpctl settxholdcount &lt;bridge&gt; &lt;tx_hold_count&gt;</b></pre> <pre data-bbox="437 1157 1465 1205"><b>cumulus@switch:~\$ sudo mstpctl settxholdcount br2 5</b></pre>
<b>forcevers</b>	<p>Sets the bridge's <i>force STP version</i> to either RSTP/STP. MSTP is not supported currently. The default is <i>RSTP</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="437 1368 1465 1474"><b>mstpctl-forcevers rstp</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="437 1516 1465 1622"><b>mstpctl setforcevers &lt;bridge&gt; {mstp rstp stp}</b></pre> <pre data-bbox="437 1664 1465 1833"><b>cumulus@switch:~\$ sudo mstpctl setforcevers br2 rstp</b></pre>
<b>tree prio</b>	<p>Sets the bridge's <i>tree priority</i> to &lt;priority&gt; for an MSTI instance. The priority value is a number between 0 and 65535 and must be a multiple of 4096. The bridge with the lowest priority is elected the <i>root bridge</i>. The default is 32768.</p>

Parameter	Description
	<p>For <code>msti</code>, only 0 is supported currently.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="421 494 789 530"><code>mstpctl-treeprio 8192</code></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="421 677 1176 713"><code>mstpctl settreeprio &lt;bridge&gt; &lt;mstid&gt; &lt;priority&gt;</code></pre> <pre data-bbox="421 819 1356 855"><code>cumulus@switch:~\$ sudo mstpctl settreeprio br2 0 8192</code></pre>
<b>treportprio</b>	<p>Sets the <i>priority</i> of port <code>&lt;port&gt;</code> to <code>&lt;priority&gt;</code> for the MSTI instance. The priority value is a number between 0 and 240 and must be a multiple of 16. The default is 128.</p> <p><b>!</b> For <code>msti</code>, only 0 is supported currently.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="421 1296 915 1332"><code>mstpctl-treeportprio swp4 64</code></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="421 1480 1356 1516"><code>mstpctl settreeportprio &lt;bridge&gt; &lt;port&gt; &lt;mstid&gt; &lt;priority&gt;</code></pre> <pre data-bbox="421 1622 1307 1679"><code>cumulus@switch:~\$ sudo mstpctl settreeportprio br2 swp4 0 64</code></pre>
<b>hello</b>	<p>Sets the bridge's <i>bridge hello time</i> to <code>&lt;time&gt;</code> seconds. The default is 2.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="421 1930 703 1966"><code>mstpctl-hello 20</code></pre>

Parameter	Description
	<p>To set this parameter at runtime, use:</p> <pre data-bbox="453 403 975 435">mstpctl sethello &lt;bridge&gt; &lt;time&gt;</pre> <pre data-bbox="453 544 1269 576">cumulus@switch:~\$ sudo mstpctl sethello br2 20</pre>
<b>portpathcost</b>	<p>Sets the <i>port cost</i> of the port <i>&lt;port&gt;</i> in bridge <i>&lt;bridge&gt;</i> to <i>&lt;cost&gt;</i>. The default is 0. <b>mstpd</b> supports only long mode; that is, 32 bits for the path cost.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="453 868 953 899">mstpctl-portpathcost swp1=10</pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="453 1051 1197 1083">mstpctl setportpathcost &lt;bridge&gt; &lt;port&gt; &lt;cost&gt;</pre> <pre data-bbox="453 1193 1344 1256">cumulus@switch:~\$ sudo mstpctl setportpathcost br2 swp1 10</pre>
<b>portadminedge</b>	<p>Enables/disables the <i>initial edge state</i> of the port <i>&lt;port&gt;</i> in bridge <i>&lt;bridge&gt;</i>. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="453 1537 988 1569">mstpctl-portadminedge swp1=yes</pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="453 1710 1246 1742">mstpctl setportadminedge &lt;bridge&gt; &lt;port&gt; {yes no}</pre> <pre data-bbox="453 1852 1361 1915">cumulus@switch:~\$ sudo mstpctl setportadminedge br2 swp1 yes</pre>

Parameter	Description
<b>portautoedge</b>	<p>Enables/disables the <i>auto transition</i> to/from the edge state of the port &lt;port&gt; in bridge &lt;bridge&gt;. The default is <i>yes</i>.</p> <p><i>portautoedge</i> is an enhancement to the standard PortAdminEdge (PortFast) mode, which allows for the automatic detection of edge ports.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"> <p><span style="color: yellow;">⚠</span> Edge ports and access ports are not the same thing. Edge ports transition directly to the forwarding state and skip the listening and learning stages. Upstream topology change notifications are not generated when an edge port's link changes state. Access ports only forward untagged traffic; however, there is no such restriction on edge ports, which can forward both tagged and untagged traffic.</p> </div> <p>When a BPDU is received on a port configured with <i>portautoedge</i>, the port ceases to be in the edge port state and transitions into a normal STP port.</p> <p>When BPDUs are no longer received on the interface, the port becomes an edge port, and transitions through the discarding and learning states before resuming forwarding.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="414 1058 915 1089"><b>mstpctl-portautoedge swp1=no</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="414 1241 1192 1273"><b>mstpctl setportautoedge &lt;bridge&gt; &lt;port&gt; {yes no}</b></pre> <pre data-bbox="414 1385 1305 1448"><b>cumulus@switch:~\$ sudo mstpctl setportautoedge br2 swp1 no</b></pre>
<b>portp2p</b>	<p>Enables/disables the <i>point-to-point detection mode</i> of the port &lt;port&gt; in bridge &lt;bridge&gt;. The default is <i>auto</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="414 1723 829 1755"><b>mstpctl-portp2p swp1=no</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="414 1898 1192 1930"><b>mstpctl setportp2p &lt;bridge&gt; &lt;port&gt; {yes no auto}</b></pre>

Parameter	Description
	<pre data-bbox="453 312 1393 346">cumulus@switch:~\$ sudo mstpcctl setportp2p br2 swp1 no</pre>
<b>portrestrrole</b>	<p>Enables/disables the ability of the port &lt;port&gt; in bridge &lt;bridge&gt; to take the <i>root role</i>. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="453 614 975 648">mstpcctl-portrestrrole swp1=no</pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="453 804 1241 838">mstpcctl setportrestrrole &lt;bridge&gt; &lt;port&gt; {yes no}</pre> <pre data-bbox="453 946 1356 1001">cumulus@switch:~\$ sudo mstpcctl setportrestrrole br2 swp1 yes</pre>
<b>portrestrtcn</b>	<p>Enables/disables the ability of the port &lt;port&gt; in bridge &lt;bridge&gt; to propagate <i>received topology change notifications</i>. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="453 1277 975 1311">mstpcctl-portrestrtcn swp1=yes</pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="453 1467 1225 1501">mstpcctl setportrestrtcn &lt;bridge&gt; &lt;port&gt; {yes no}</pre> <pre data-bbox="453 1609 1343 1664">cumulus@switch:~\$ sudo mstpcctl setportrestrtcn br2 swp1 yes</pre>
<b>portnetwork</b>	<p>Enables/disables the <i>bridge assurance capability</i> for a network port &lt;port&gt; in bridge &lt;bridge&gt;. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p>

Parameter	Description
	<pre data-bbox="421 291 915 321"><b>mstpctl-portnetwork swp4=yes</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="421 466 1171 496"><b>mstpctl setportnetwork &lt;bridge&gt; &lt;port&gt; {yes no}</b></pre> <pre data-bbox="421 608 1367 671"><b>cumulus@switch:~\$ sudo mstpctl setportnetwork br2 swp4 yes</b></pre>
<b>bpduguard</b>	<p>Enables/disables the <i>BPDU guard configuration</i> of the port <b>&lt;port&gt;</b> in bridge <b>&lt;bridge&gt;</b>. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="421 952 861 982"><b>mstpctl-bpduguard swp1=no</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="421 1136 1139 1165"><b>mstpctl setbpduguard &lt;bridge&gt; &lt;port&gt; {yes no}</b></pre> <pre data-bbox="421 1277 1334 1341"><b>cumulus@switch:~\$ sudo mstpctl setbpduguard br2 swp1 yes</b></pre>
<b>portbpdufilter</b>	<p>Enables/disables the <i>BPDU filter</i> functionality for a port <b>&lt;port&gt;</b> in bridge <b>&lt;bridge&gt;</b>. The default is <i>no</i>.</p> <p>To set this parameter persistently, configure it under the bridge stanza:</p> <pre data-bbox="421 1617 894 1647"><b>mstpctl-bpdufilter swp4=yes</b></pre> <p>To set this parameter at runtime, use:</p> <pre data-bbox="421 1801 1220 1831"><b>mstpctl setportbpdufilter &lt;bridge&gt; &lt;port&gt; {yes no}</b></pre>

Parameter	Description
	<pre>cumulus@switch:~\$ sudo mstpcctl setportbpdufilter br2 swp4 yes</pre>

## Configuration Files

- /etc/network/interfaces
- /etc/cumulus/switchd.conf

## Man Pages

- brctl(8)
- bridge-utils-interfaces(5)
- ifupdown-addons-interfaces(5)
- mstpcctl(8)
- mstpcctl-utils-interfaces(5)

## Useful Links

The source code for `mstpd/mstpcctl` was written by [Vitalii Demianets](#) and is hosted at the sourceforge URL below.

- <https://sourceforge.net/projects/mstpd/>
- [http://en.wikipedia.org/wiki/Spanning\\_Tree\\_Protocol](http://en.wikipedia.org/wiki/Spanning_Tree_Protocol)

## Caveats and Errata

- MSTP is not supported currently. However, interoperability with MSTP networks can be accomplished using PVRSTP or PVSTP.

## Link Layer Discovery Protocol

The `lldpd` daemon implements the IEEE802.1AB (Link Layer Discovery Protocol, or LLDP) standard. LLDP allows you to know which ports are neighbors of a given port. By default, `lldpd` runs as a daemon and is started at system boot. `lldpd` command line arguments are placed in `/etc/default/lldpd`. `lldpd` configuration options are placed in `/etc/lldpd.conf` or under `/etc/lldpd.d/`.

For more details on the command line arguments and config options, please see `man lldpd(8)`.

`lldpd` supports CDP (Cisco Discovery Protocol, v1 and v2). `lldpd` logs by default into `/var/log/daemon.log` with an `lldpd` prefix.

`lldpcli` is the CLI tool to query the `lldpd` daemon for neighbors, statistics and other running configuration information. See `man lldpcli(8)` for details.

## Contents

(Click to expand)

- [Contents \(see page 192\)](#)
- [Commands \(see page 192\)](#)
- [Man Pages \(see page 192\)](#)
- [Configuring LLDP \(see page 192\)](#)
- [Example lldpcli Commands \(see page 193\)](#)
- [Enabling the SNMP Subagent in LLDP \(see page 198\)](#)
- [Configuration Files \(see page 198\)](#)
- [Useful Links \(see page 198\)](#)
- [Caveats and Errata \(see page 198\)](#)

## Commands

- [lldpd \(daemon\)](#)
- [lldpcli \(interactive CLI\)](#)

## Man Pages

- [man lldpd](#)
- [man lldpcli](#)

## Configuring LLDP

You configure lldpd settings in `/etc/lldpd.conf` or `/etc/lldpd.d/`.

Here is an example persistent configuration:

```
cumulus@switch:~$ sudo cat /etc/lldpd.conf
configure lldp tx-interval 40
configure lldp tx-hold 3
configure system interface pattern-blacklist "eth0"
```

lldpd logs to `/var/log/daemon.log` with the `lldpd` prefix:

```
cumulus@switch:~$ sudo tail -f /var/log/daemon.log | grep lldp
Aug  7 17:26:17 switch lldpd[1712]: unable to get system name
Aug  7 17:26:17 switch lldpd[1712]: unable to get system name
Aug  7 17:26:17 switch lldpcli[1711]: lldpd should resume operations
Aug  7 17:26:32 switch lldpd[1805]: NET-SNMP version 5.4.3 AgentX subagent
connected
```

## **Example lldpcli Commands**

To see all neighbors on all ports/interfaces:

```
cumulus@switch:~$ sudo lldpcli show neighbors
-----
---
LLDP neighbors:
-----
Interface: eth0, via: LLDP, RID: 1, Time: 0 day, 17:38:08
  Chassis:
    ChassisID: mac 08:9e:01:e9:66:5a
    SysName: PIONEERMS22
    SysDescr: Cumulus Linux version 2.5.4 running on quanta lb9
    MgmtIP: 192.168.0.22
    Capability: Bridge, on
    Capability: Router, on
  Port:
    PortID: ifname swp47
    PortDescr: swp47
-----
Interface: swp1, via: LLDP, RID: 10, Time: 0 day, 17:08:27
  Chassis:
    ChassisID: mac 00:01:00:00:09:00
    SysName: MSP-1
    SysDescr: Cumulus Linux version 3.0.0 running on QEMU Standard PC
(i440FX + PIIX, 1996)
    MgmtIP: 192.0.2.9
    MgmtIP: fe80::201:ff:fe00:900
    Capability: Bridge, off
    Capability: Router, on
  Port:
    PortID: ifname swp1
    PortDescr: swp1
-----
Interface: swp2, via: LLDP, RID: 10, Time: 0 day, 17:08:27
  Chassis:
    ChassisID: mac 00:01:00:00:09:00
    SysName: MSP-1
    SysDescr: Cumulus Linux version 3.0.0 running on QEMU Standard PC
```

```
(i440FX + PIIX, 1996)
    MgmtIP:      192.0.2.9
    MgmtIP:      fe80::201:ff:fe00:900
    Capability:  Bridge, off
    Capability:  Router, on
    Port:
        PortID:      ifname swp2
        PortDescr:    swp2
-----
---  
Interface:      swp3, via: LLDP, RID: 11, Time: 0 day, 17:08:27
    Chassis:
        ChassisID:   mac 00:01:00:00:0a:00
        SysName:     MSP-2
        SysDescr:    Cumulus Linux version 3.0.0 running on QEMU Standard PC
(i440FX + PIIX, 1996)
    MgmtIP:      192.0.2.10
    MgmtIP:      fe80::201:ff:fe00:a00
    Capability:  Bridge, off
    Capability:  Router, on
    Port:
        PortID:      ifname swp1
        PortDescr:    swp1
-----
---  
Interface:      swp4, via: LLDP, RID: 11, Time: 0 day, 17:08:27
    Chassis:
        ChassisID:   mac 00:01:00:00:0a:00
        SysName:     MSP-2
        SysDescr:    Cumulus Linux version 3.0.0 running on QEMU Standard PC
(i440FX + PIIX, 1996)
    MgmtIP:      192.0.2.10
    MgmtIP:      fe80::201:ff:fe00:a00
    Capability:  Bridge, off
    Capability:  Router, on
    Port:
        PortID:      ifname swp2
        PortDescr:    swp2
-----
---  
Interface:      swp49s1, via: LLDP, RID: 9, Time: 0 day, 16:55:00
    Chassis:
        ChassisID:   mac 00:01:00:00:0c:00
        SysName:     TORC-1-2
        SysDescr:    Cumulus Linux version 3.0.0 running on QEMU Standard PC
```

```
(i440FX + PIIX, 1996)
MgmtIP:      192.0.2.12
MgmtIP:      fe80::201:ff:fe00:c00
Capability:  Bridge, on
Capability:  Router, on
Port:
  PortID:      ifname swp6
  PortDescr:   swp6
-----
---
Interface:    swp49s0, via: LLDP, RID: 9, Time: 0 day, 16:55:00
Chassis:
  ChassisID:   mac 00:01:00:00:0c:00
  SysName:     TORC-1-2
  SysDescr:    Cumulus Linux version 3.0.0 running on QEMU Standard PC
(i440FX + PIIX, 1996)
MgmtIP:      192.0.2.12
MgmtIP:      fe80::201:ff:fe00:c00
Capability:  Bridge, on
Capability:  Router, on
Port:
  PortID:      ifname swp5
  PortDescr:   swp5
-----
---
```

To see `lldpd` statistics for all ports:

```
cumulus@switch:~$ sudo lldpccli show statistics
-----
LLDP statistics:
-----
Interface:    eth0
  Transmitted: 9423
  Received:    17634
  Discarded:   0
  Unrecognized: 0
  Ageout:      10
  Inserted:    20
  Deleted:     10
-----
Interface:    swp1
  Transmitted: 9423
```

```
Received:      6264
Discarded:     0
Unrecognized:  0
Ageout:        0
Inserted:      2
Deleted:       0
-----
Interface:    swp2
Transmitted:   9423
Received:      6264
Discarded:     0
Unrecognized:  0
Ageout:        0
Inserted:      2
Deleted:       0
-----
Interface:    swp3
Transmitted:   9423
Received:      6265
Discarded:     0
Unrecognized:  0
Ageout:        0
Inserted:      2
Deleted:       0
-----
... and more (output truncated to fit this document)
```

To see lldpd statistics summary for all ports:

```
cumulus@switch:~$ sudo lldpcli show statistics summary
-----
LLDP Global statistics:
-----
Summary of stats:
Transmitted:  648186
Received:     437557
Discarded:    0
Unrecognized: 0
Ageout:       10
Inserted:     38
Deleted:      10
```

To see the lldpd running configuration:

```
cumulus@switch:~$ sudo lldpcli show running-configuration
-----
Global configuration:
-----
Configuration:
  Transmit delay: 30
  Transmit hold: 4
  Receive mode: no
  Pattern for management addresses: (none)
  Interface pattern: (none)
  Interface pattern blacklist: (none)
  Interface pattern for chassis ID: (none)
  Override description with: (none)
  Override platform with: Linux
  Override system name with: (none)
  Advertise version: yes
  Update interface descriptions: no
  Promiscuous mode on managed interfaces: no
  Disable LLDP-MED inventory: yes
  LLDP-MED fast start mechanism: yes
  LLDP-MED fast start interval: 1
  Source MAC for LLDP frames on bond slaves: local
  Portid TLV Subtype for lldp frames: ifname
-----
```

## Runtime Configuration (Advanced)



A runtime configuration does not persist when you reboot the switch — all changes are lost.

To configure active interfaces:

```
lldpcli configure system interface pattern "swp*"
```

To configure inactive interfaces:

```
lldpcli configure system interface pattern-blacklist "eth0"
```



The active interface list always overrides the inactive interface list.

To reset any interface list to none:

```
lldpcli configure system interface pattern-blacklist ""
```

## ***Enabling the SNMP Subagent in LLDP***

LLDP does not enable the SNMP subagent by default. You need to edit `/etc/default/lldpd` and enable the `-x` option.

```
cumulus@switch:~$ sudo nano /etc/default/lldpd
DAEMON_OPTS=""

# Uncomment to start SNMP subagent
#DAEMON_OPTS="-x"

# Enable CDP by default
DAEMON_OPTS="$DAEMON_OPTS -c"
```

## ***Configuration Files***

- `/etc/llpd.conf`
- `/etc/llpd.d`
- `/etc/default/llpd`

## ***Useful Links***

- <http://vincentbernat.github.io/llpd/>
- [http://en.wikipedia.org/wiki/Link\\_Layer\\_Discovery\\_Protocol](http://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol)

## ***Caveats and Errata***

- Annex E (and hence Annex D) of IEEE802.1AB (lldp) is not supported.

## ***Prescriptive Topology Manager - PTM***

In data center topologies, right cabling is a time-consuming endeavor and is error prone. Prescriptive Topology Manager (PTM) is a dynamic cabling verification tool to help detect and eliminate such errors. It takes a Graphviz-DOT specified network cabling plan (something many operators already generate), stored in a `topology.dot` file, and couples it with runtime information derived from LLDP to verify that the cabling matches the specification. The check is performed on every link transition on each node in the network.

You can customize the `topology.dot` file to control `ptmd` at both the global/network level and the node /port level.

PTM runs as a daemon, named `ptmd`.

For more information, see `man ptmd(8)`.

## Contents

(Click to expand)

- [Contents \(see page 199\)](#)
- [Supported Features \(see page 199\)](#)
- [Configuring PTM \(see page 200\)](#)
- [Basic Topology Example \(see page 200\)](#)
- [Advanced PTM Configuration \(see page 201\)](#)
  - [Scripts \(see page 201\)](#)
  - [Configuration Parameters \(see page 201\)](#)
    - [Host-only Parameters \(see page 202\)](#)
    - [Global Parameters \(see page 202\)](#)
    - [Per-port Parameters \(see page 203\)](#)
    - [Templates \(see page 203\)](#)
    - [Supported BFD and LLDP Parameters \(see page 204\)](#)
  - [Bidirectional Forwarding Detection \(BFD\) \(see page 205\)](#)
    - [Configuring BFD \(see page 205\)](#)
    - [Echo Function \(see page 205\)](#)
- [Enabling Quagga to Check Link State \(see page 206\)](#)
- [Using ptmd Service Commands \(see page 207\)](#)
- [Using ptmctl Commands \(see page 208\)](#)
  - [ptmctl Examples \(see page 208\)](#)
  - [ptmctl Error Outputs \(see page 210\)](#)
- [Configuration Files \(see page 211\)](#)
- [Useful Links \(see page 211\)](#)
- [Caveats and Errata \(see page 211\)](#)

## Supported Features

- Topology verification using LLDP. `ptmd` creates a client connection to the LLDP daemon, `lldpd`, and retrieves the neighbor relationship between the nodes/ports in the network and compares them against the prescribed topology specified in the `topology.dot` file.
- Only physical interfaces, like `swp1` or `eth0`, are currently supported. Cumulus Linux does not support specifying virtual interfaces like bonds or subinterfaces like `eth0.200` in the topology file.

- Forwarding path failure detection using Bidirectional Forwarding Detection (BFD); however, demand mode is not supported. For more information on how BFD operates in Cumulus Linux, [see below \(see page 205\)](#) and see `man ptmd(8)`.
- Integration with Quagga (PTM to Quagga notification).
- Client management: `ptmd` creates an abstract named socket `/var/run/ptmd.socket` on startup. Other applications can connect to this socket to receive notifications and send commands.
- Event notifications: see Scripts below.
- User configuration via a `topology.dot` file; [see below \(see page 200\)](#).

## Configuring PTM

`ptmd` verifies the physical network topology against a DOT-specified network graph file, `/etc/ptm.d/topology.dot`.



This file must be present or else `ptmd` will not start. You can specify an alternate file using the `-c` option.

PTM supports [undirected graphs](#).

At startup, `ptmd` connects to `lldpd`, the LLDP daemon, over a Unix socket and retrieves the neighbor name and port information. It then compares the retrieved port information with the configuration information that it read from the topology file. If there is a match, then it is a PASS, else it is a FAIL.

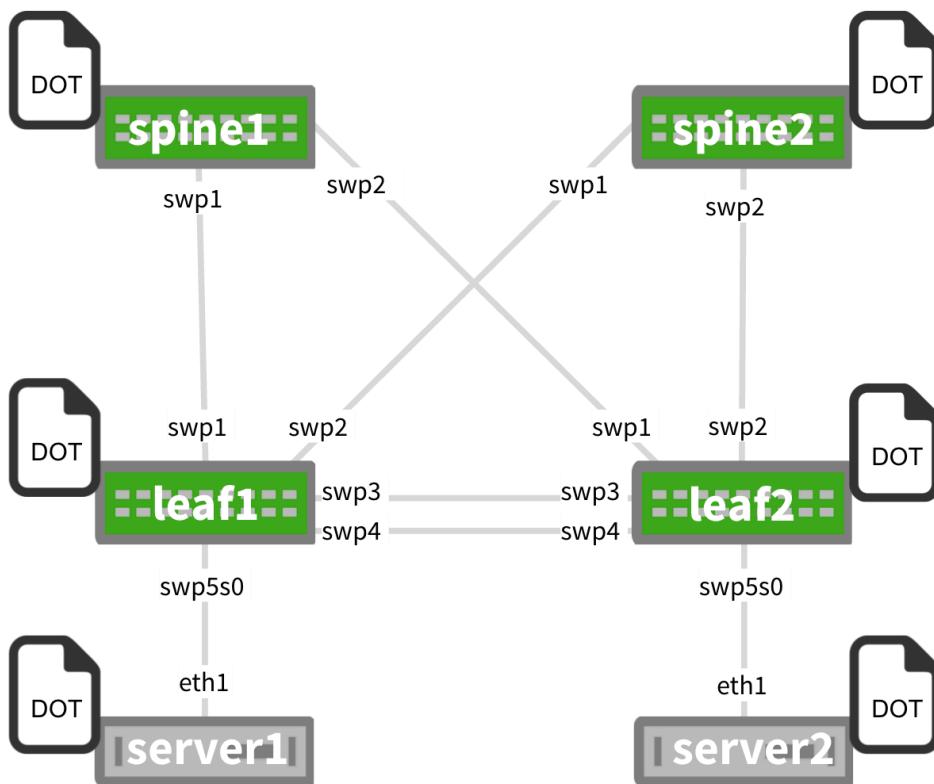


PTM performs its LLDP neighbor check using the PortID ifname TLV information. Previously, it used the PortID port description TLV information.

## Basic Topology Example

This is a basic example DOT file and its corresponding topology diagram. You should use the same `topology.dot` file on all switches, and don't split the file per device; this allows for easy automation by pushing/pulling the same exact file on each device!

```
graph G {
    "spine1": "swp1" -- "leaf1": "swp1";
    "spine1": "swp2" -- "leaf2": "swp1";
    "spine2": "swp1" -- "leaf1": "swp2";
    "spine2": "swp2" -- "leaf2": "swp2";
    "leaf1": "swp3" -- "leaf2": "swp3";
    "leaf1": "swp4" -- "leaf2": "swp4";
    "leaf1": "swp5s0" -- "server1": "eth1";
    "leaf2": "swp5s0" -- "server2": "eth1";
}
```



## Advanced PTM Configuration

PTM allows for more advanced configuration of the topology file using parameters you specify in the topology file.

### Scripts

`ptmd` executes scripts at `/etc/ptm.d/if-topo-pass` and `/etc/ptm.d/if-topo-fail` for each interface that goes through a change, running `if-topo-pass` when an LLDP or BFD check passes and running `if-topo-fails` when the check fails. The scripts receive an argument string that is the result of the `ptmctl` command, described in the `ptmd` commands section below (see page 207).

You should modify these default scripts as needed.

### Configuration Parameters

You can configure `ptmd` parameters in the topology file. The parameters are classified as host-only, global, per-port/node and templates.

## Host-only Parameters

*Host-only parameters* apply to the entire host on which PTM is running. You can include the `hostnametype` host-only parameter, which specifies whether PTM should use only the host name (`hostname`) or the fully-qualified domain name (`fqdn`) while looking for the `self-node` in the graph file. For example, in the graph file below, PTM will ignore the FQDN and only look for `switch04`, since that is the host name of the switch it's running on:

- ➊ It's a good idea to always wrap the hostname in double quotes, like "`www.example.com`". Otherwise, `ptmd` can fail if you specify a fully-qualified domain name as the hostname and do not wrap it in double quotes.
- Further, to avoid errors when starting the `ptmd` process, make sure that `/etc/hosts` and `/etc/hostname` both reflect the hostname you are using in the `topology.dot` file.

```
graph G {
    hostnametype="hostname"
    BFD="upMinTx=150,requiredMinRx=250"
    "cumulus":"swp44" -- "switch04.cumulusnetworks.com":"swp20"
    "cumulus":"swp46" -- "switch04.cumulusnetworks.com":"swp22"
}
```

However, in this next example, PTM will compare using the FQDN and look for `switch05.cumulusnetworks.com`, which is the FQDN of the switch it's running on:

```
graph G {
    hostnametype="fqdn"
    "cumulus":"swp44" -- "switch05.cumulusnetworks.com":"swp20"
    "cumulus":"swp46" -- "switch05.cumulusnetworks.com":"swp22"
}
```

## Global Parameters

*Global parameters* apply to every port listed in the topology file. There are two global parameters: LLDP and BFD. LLDP is enabled by default; if no keyword is present, default values are used for all ports. However, BFD is disabled if no keyword is present, unless there is a per-port override configured. For example:

```
graph G {
    LLDP=""
    BFD="upMinTx=150,requiredMinRx=250,afi=both"
```

```

    "cumulus": "swp44" -- "qct-ly2-04": "swp20"
    "cumulus": "swp46" -- "qct-ly2-04": "swp22"
}

```

## Per-port Parameters

*Per-port parameters* provide finer-grained control at the port level. These parameters override any global or compiled defaults. For example:

```

graph G {
    LLDP=""
    BFD="upMinTx=300,requiredMinRx=100"
    "cumulus": "swp44" -- "qct-ly2-04": "swp20" [BFD="upMinTx=150,
requiredMinRx=250,afi=both"]
    "cumulus": "swp46" -- "qct-ly2-04": "swp22"
}

```

## Templates

*Templates* provide flexibility in choosing different parameter combinations and applying them to a given port. A template instructs `ptmd` to reference a named parameter string instead of a default one. There are two parameter strings `ptmd` supports:

- `bfdtmpl`, which specifies a custom parameter tuple for BFD.
- `lldptmpl`, which specifies a custom parameter tuple for LLDP.

For example:

```

graph G {
    LLDP=""
    BFD="upMinTx=300,requiredMinRx=100"
    BFD1="upMinTx=200,requiredMinRx=200"
    BFD2="upMinTx=100,requiredMinRx=300"
    LLDP1="match_type=ifname"
    LLDP2="match_type=portdescr"
    "cumulus": "swp44" -- "qct-ly2-04": "swp20" [BFD="bfdtmpl=BFD1",
LLDP="lldptmpl=LLDP1"]
    "cumulus": "swp46" -- "qct-ly2-04": "swp22" [BFD="bfdtmpl=BFD2",
LLDP="lldptmpl=LLDP2"]
    "cumulus": "swp46" -- "qct-ly2-04": "swp22"
}

```

In this template, LLDP1 and LLDP2 are templates for LLDP parameters while BFD1 and BFD2 are templates for BFD parameters.

## Supported BFD and LLDP Parameters

`ptmd` supports the following BFD parameters:

- `upMinTx`: the minimum transmit interval, which defaults to `300ms`, specified in milliseconds.
- `requiredMinRx`: the minimum interval between received BFD packets, which defaults to `300ms`, specified in milliseconds.
- `detectMult`: the detect multiplier, which defaults to `3`, and can be any non-zero value.
- `afi`: the address family to be supported for the edge. The address family must be one of the following:
  - `v4`: BFD sessions will be built for only IPv4 connected peer. This is the default value.
  - `v6`: BFD sessions will be built for only IPv6 connected peer.
  - `both`: BFD sessions will be built for both IPv4 and IPv6 connected peers.

The following is an example of a topology with BFD applied at the port level:

```
graph G {
    "cumulus-1": "swp44" -- "cumulus-2": "swp20" [BFD="upMinTx=300,
requiredMinRx=100,afi=v6"]
    "cumulus-1": "swp46" -- "cumulus-2": "swp22" [BFD="detectMult=4"]
}
```

`ptmd` supports the following LLDP parameters:

- `match_type`, which defaults to the interface name (`ifname`), but can accept a port description (`portdescr`) instead if you want `lldpd` to compare the topology against the port description instead of the interface name. You can set this parameter globally or at the per-port level.
- `match_hostname`, which defaults to the host name (`hostname`), but enables PTM to match the topology using the fully-qualified domain name (`fqdn`) supplied by LLDP.

The following is an example of a topology with LLDP applied at the port level:

```
graph G {
    "cumulus-1": "swp44" -- "cumulus-2": "swp20" [LLDP=
match_hostname=fqdn"]
    "cumulus-1": "swp46" -- "cumulus-2": "swp22" [LLDP=
match_type=portdescr]
}
```



When you specify `match_hostname=fqdn`, `ptmd` will match the entire FQDN, like `cumulus-2.domain.com` in the example below. If you do not specify anything for `match_hostname`, `ptmd` will match based on hostname only, like `cumulus-3` below, and ignore the rest of the URL:

```

graph G {
    "cumulus-1": "swp44" -- "cumulus-2.domain.com": "swp20"
    [LLDP="match_hostname=fqdn"]
    "cumulus-1": "swp46" -- "cumulus-3": "swp22" [LLDP=
    match_type=portdescr"]
}

```

## Bidirectional Forwarding Detection (BFD)

BFD provides low overhead and rapid detection of failures in the paths between two network devices. It provides a unified mechanism for link detection over all media and protocol layers. Use BFD to detect failures for IPv4 and IPv6 single or multihop paths between any two network devices, including unidirectional path failure detection. For more information, see the [BFD chapter \(see page 446\)](#).



BFD requires an IP address for any interface on which it is configured. The neighbor IP address for a single hop BFD session must be in the ARP table before BFD can start sending control packets.



You cannot specify BFD multihop sessions in the `topology.dot` file since you cannot specify the source and destination IP address pairs in that file. Use [Quagga \(see page 393\)](#) to configure multihop sessions.

## Configuring BFD

You configure BFD one of two ways: by specifying the configuration in the `topology.dot` file, or using [Quagga \(see page 446\)](#). However, the topology file has some limitations:

- The `topology.dot` file supports creating BFD IPv4 and IPv6 single hop sessions only; you cannot specify IPv4 or IPv6 multihop sessions in the topology file.
- The topology file supports BFD sessions for only link-local IPv6 peers; BFD sessions for global IPv6 peers discovered on the link will not be created.

## Echo Function

Cumulus Linux supports the *echo function* for IPv4 single hops only, and with the a synchronous operating mode only (Cumulus Linux does not support demand mode).

You use the echo function primarily to test the forwarding path on a remote system. To enable the echo function, set `echoSupport` to 1 in the topology file.

Once the echo packets are looped by the remote system, the BFD control packets can be sent at a much lower rate. You configure this lower rate by setting the `slowMinTx` parameter in the topology file to a non-zero value of milliseconds.

You can use more aggressive detection times for echo packets since the round-trip time is reduced because they are accessing the forwarding path. You configure the detection interval by setting the `echoMinRx` parameter in the topology file to a non-zero value of milliseconds; the minimum setting is 50 milliseconds. Once configured, BFD control packets are sent out at this required minimum echo Rx interval. This indicates to the peer that the local system can loop back the echo packets. Echo packets are transmitted if the peer supports receiving echo packets.

## About the Echo Packet

BFD echo packets are encapsulated into UDP packets over destination and source UDP port number 3785. The BFD echo packet format is vendor-specific and has not been defined in the RFC. BFD echo packets that originate from Cumulus Linux are 8 bytes long and have the following format:

0	1	2	3
Version	Length	Reserved	
My Discriminator			

Where:

- **Version** is the version of the BFD echo packet.
- **Length** is the length of the BFD echo packet.
- **My Discriminator** is a non-zero value that uniquely identifies a BFD session on the transmitting side. When the originating node receives the packet after being looped back by the receiving system, this value uniquely identifies the BFD session.

## Transmitting and Receiving Echo Packets

BFD echo packets are transmitted for a BFD session only when the peer has advertised a non-zero value for the required minimum echo Rx interval (the `echoMinRx` setting) in the BFD control packet when the BFD session starts. The transmit rate of the echo packets is based on the peer advertised echo receive value in the control packet.

BFD echo packets are looped back to the originating node for a BFD session only if locally the `echoMinRx` and `echoSupport` are configured to a non-zero values.

## Using Echo Function Parameters

You configure the echo function by setting the following parameters in the topology file at the global, template and port level:

- **echoSupport:** Enables and disables echo mode. Set to 1 to enable the echo function. It defaults to 0 (disable).
- **echoMinRx:** The minimum interval between echo packets the local system is capable of receiving. This is advertised in the BFD control packet. When the echo function is enabled, it defaults to 50. If you disable the echo function, this parameter is automatically set to 0, which indicates the port or the node cannot process or receive echo packets.
- **slowMinTx:** The minimum interval between transmitting BFD control packets when the echo packets are being exchanged.

## Enabling Quagga to Check Link State

The Quagga routing suite enables additional checks to ensure that routing adjacencies are formed only on links that have connectivity conformant to the specification, as determined by `ptmd`.



You only need to do this to check link state; you don't need to enable PTM to determine BFD status.

To enable the check:

```
quagga# conf t
quagga(config)# ptm-enable
quagga(config)#

```

To disable the checks:

```
quagga# conf t
quagga(config)# no ptm-enable
quagga(config)#

```

When the `ptm-enable` flag is configured by the user, the `zebra` daemon connects to `ptmd` over a Unix socket. Any time there is a change of status for an interface, `ptmd` sends notifications to `zebra`. Zebra maintains a `ptm-status` flag per interface and evaluates routing adjacency based on this flag. To check the per-interface `ptm-status`:

```
quagga# show interface swp1
Interface swp1 is up, line protocol is up
  Link ups:      0    last: (never)
  Link downs:   0    last: (never)
  PTM status: disabled
  vrf: Default-IP-Routing-Table
  index 3 metric 0 mtu 1550
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: c4:54:44:bd:01:41
quagga#

```

## Using `ptmd` Service Commands

PTM sends client notifications in CSV format.

`cumulus@switch:~$ sudo systemctl start|restart|force-reload ptmd.service`: Starts or restarts the `ptmd` service. The `topology.dot` file must be present in order for the service to start.

cumulus@switch:~\$ sudo systemctl reload ptmd.service: Instructs `ptmd` to read the `topology.dot` file again without restarting, applying the new configuration to the running state.

cumulus@switch:~\$ sudo systemctl stop ptmd.service: Stops the `ptmd` service.

cumulus@switch:~\$ sudo systemctl status ptmd.service: Retrieves the current running state of `ptmd`.

## Using `ptmctl` Commands

`ptmctl` is a client of `ptmd`; it retrieves the daemon's operational state. It connects to `ptmd` over a Unix socket and listens for notifications. `ptmctl` parses the CSV notifications sent by `ptmd`.

See `man ptmctl` for more information.

### `ptmctl Examples`

For basic output, use `ptmctl` without any options:

```
cumulus@switch:~$ sudo ptmctl

-----
port  cb1      BFD      BFD
      status    status   peer
                         BFD      BFD
                         local    type
-----
swp1  pass     pass    11.0.0.2
      N/A      N/A
      N/A      N/A
      N/A      N/A
      N/A      N/A
```

For more detailed output, use the `-d` option:

```
cumulus@switch:~$ sudo ptmctl -d

-----
port  cb1      exp      act      sysname  portID  portDescr  match  last
BFD   BFD      BFD      BFD       det_mult tx_timeout rx_timeout
echo_tx_timeout echo_rx_timeout max_hop_cnt
      status  nbr      nbr
      Type    state   peer  DownDiag
      on      upd
-----
swp45 pass     h1:swp1 h1:swp1  h1      swp1      swp1      IfName 5m: 5s  N
      N/A      N/A      N/A      N/A      N/A      N/A      N
```

```
/A           N/A           N/A
swp46 fail  h2:swp1 h2:swp1 h2           swp1      swp1      IfName 5m: 5s  N
/A   N/A     N/A   N/A     N/A           N/A       N/A      N
/A           N/A           N/A
```

To return information on active BFD sessions `ptmd` is tracking, use the `-b` option:

```
cumulus@switch:~$ sudo ptmctl -b

-----
port  peer          state local        type    diag
-----
swp1  11.0.0.2     Up    N/A         singlehop  N/A
N/A   12.12.12.1   Up    12.12.12.4 multihop   N/A
```

To return LLDP information, use the `-l` option. It returns only the active neighbors currently being tracked by `ptmd`.

```
cumulus@switch:~$ sudo ptmctl -l

-----
port  sysname  portID  port  match  last
                descr  on    upd
-----
swp45 h1        swp1    swp1    IfName 5m:59s
swp46 h2        swp1    swp1    IfName 5m:59s
```

To return detailed information on active BFD sessions `ptmd` is tracking, use the `-b` and `-d` options (results are for an IPv6-connected peer):

```
cumulus@switch:~$ sudo ptmctl -b -d

-----
port  peer          state local type    diag  det  tx_timeout
rx_timeout echo        echo    max   rx_ctrl  tx_ctrl  rx_echo
tx_echo

mult           tx_timeout  rx_timeout
```

```
hop_cnt
-----
-----
-----
swp1  fe80::202:ff:fe00:1  Up      N/A     singlehop  N/A   3    300
900      0          0          N/A        187172  185986   0
0
swp1  3101:abc:bcad::2  Up      N/A     singlehop  N/A   3    300
900      0          0          N/A        501      533     0
0
```

## ***ptmctl Error Outputs***

If there are errors in the topology file or there isn't a session, PTM will return appropriate outputs. Typical error strings are:

```
Topology file error [/etc/ptm.d/topology.dot] [cannot find node cumulus] -
please check /var/log/ptmd.log for more info

Topology file error [/etc/ptm.d/topology.dot] [cannot open file (errno 2)] -
please check /var/log/ptmd.log for more info

No Hostname/MgmtIP found [Check LLDPD daemon status] -
please check /var/log/ptmd.log for more info

No BFD sessions . Check connections

No LLDP ports detected. Check connections

Unsupported command
```

For example:

```
cumulus@switch:~$ sudo ptmctl
-----
cmd      error
-----
get-status  Topology file error [/etc/ptm.d/topology.dot] [cannot open file
(errno 2)] - please check /var/log/ptmd.log for more info
```



If you encounter errors with the `topology.dot` file, you can use `dot` (included in the Graphviz package) to validate the syntax of the topology file.

By simply opening the topology file with Graphviz, you can ensure that it is readable and that the file format is correct.

If you edit `topology.dot` file from a Windows system, be sure to double check the file formatting; there may be extra characters that keep the graph from working correctly.

## Configuration Files

- `/etc/ptm.d/topology.dot`
- `/etc/ptm.d/if-topo-pass`
- `/etc/ptm.d/if-topo-fail`

## Useful Links

- Bidirectional Forwarding Detection (BFD)
- Graphviz
- LLDP on Wikipedia
- PTMd GitHub repo

## Caveats and Errata

- Prior to version 2.1, Cumulus Linux stored the `ptmd` configuration files in `/etc/cumulus/ptm.d`. When you upgrade to version 2.1 or later, all the existing `ptmd` files are copied from their original location to `/etc/ptm.d` with a `dpkg-old` extension, except for `topology.dot`, which gets copied to `/etc/ptm.d`.

If you customized the `if-topo-pass` and `if-topo-fail` scripts, they are also copied to `dpkg-old`, and you must modify them so they can parse the CSV output correctly.

Sample `if-topo-pass` and `if-topo-fail` scripts are available in `/etc/ptm.d`. A sample `topology.dot` file is available in `/usr/share/doc/ptmd/examples`.

## Bonding - Link Aggregation

Linux bonding provides a method for aggregating multiple network interfaces (the slaves) into a single logical bonded interface (the bond). Cumulus Linux bonding supports the IEEE 802.3ad link aggregation mode. Link aggregation allows one or more links to be aggregated together to form a *link aggregation group* (LAG), such that a media access control (MAC) client can treat the link aggregation group as if it were a single link. The benefits of link aggregation are:

- Linear scaling of bandwidth as links are added to LAG
- Load balancing
- Failover protection

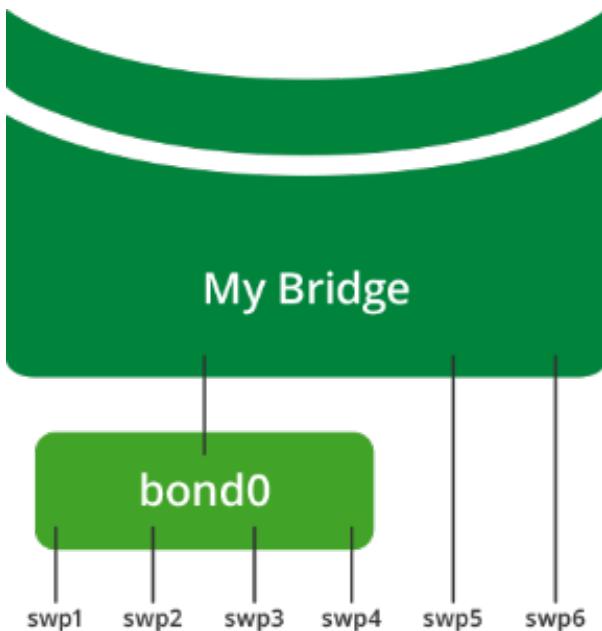
Cumulus Linux LAG control protocol is LACP version 1.

## Contents

(Click to expand)

- [Contents \(see page 211\)](#)
- [Example: Bonding 4 Slaves \(see page 212\)](#)
- [Hash Distribution \(see page 214\)](#)
- [Configuration Files \(see page 214\)](#)
- [Useful Links \(see page 214\)](#)
- [Caveats and Errata \(see page 214\)](#)

### **Example: Bonding 4 Slaves**



In this example, front panel port interfaces `swp1`-`swp4` are slaves in `bond0` (`swp5` and `swp6` are not part of `bond0`).

The name of the bond is arbitrary as long as it follows Linux interface naming guidelines, and is unique within the switch. The only bonding mode supported in Cumulus Linux is `802.3ad`. There are several `802.3ad` settings that can be applied to each bond.

All of the following settings except for `bond-slaves` are set to the recommended defaults and should only be added to a configuration in `/etc/network/interfaces` if you plan to use a different setting.

- `bond-slaves`: The list of slaves in bond.
- `bond-mode`: Is set to `802.3ad` by default and **must not** be changed.
- `bond-miimon`: How often the link state of each slave is inspected for link failures. It `100`, the recommended value.
- `bond-use-carrier`: How to determine link state. It defaults to `1`.
- `bond-xmit-hash-policy`: Hash method used to select the slave for a given packet; it defaults to `layer3+4` and **must not** be changed.
- `bond-lacp-rate`: Rate to ask link partner to transmit LACP control packets. It defaults to `1`.

- **bond-min-links**: Specifies the minimum number of links that must be active before asserting carrier on the bond. It defaults to 1, but a value greater than 1 is useful if higher level services need to ensure a minimum of aggregate bandwidth before putting the bond in service. Keeping **bond-min-links** set to 1 indicates the bond must have at least one active member for bond to assert carrier. If the number of active members drops below the **bond-min-links** setting, the bond will appear to upper-level protocols as *link-down*. When the number of active links returns to greater than or equal to **bond-min-links**, the bond will become *link-up*.

See Useful Links below for more details on settings.

To configure the bond, edit `/etc/network/interfaces` and add a stanza for bond0:

```
auto bond0
iface bond0
    address 10.0.0.1/30
    bond-slaves swp1 swp2 swp3 swp4
```

However, if you are intending that the bond become part of a bridge, you don't need to specify an IP address. The configuration would look like this:

```
auto bond0
iface bond0
    bond-slaves glob swp1-4
```

See `man interfaces` for more information on `/etc/network/interfaces`.

When networking is started on switch, bond0 is created as MASTER and interfaces swp1-swp4 come up in SLAVE mode, as seen in the `ip link show` command:

```
3: swp1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond0 state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
4: swp2: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond0 state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
5: swp3: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond0 state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
6: swp4: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond0 state UP mode DEFAULT qlen 500
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
```

And

```
55: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue
      state UP mode DEFAULT
        link/ether 44:38:39:00:03:c1 brd ff:ff:ff:ff:ff:ff
```



All slave interfaces within a bond will have the same MAC address as the bond. Typically, the first slave added to the bond donates its MAC address for the bond. The other slaves' MAC addresses are set to the bond MAC address. The bond MAC address is used as source MAC address for all traffic leaving the bond, and provides a single destination MAC address to address traffic to the bond.

## Hash Distribution

Egress traffic through a bond is distributed to a slave based on a packet hash calculation. This distribution provides load balancing over the slaves. The hash calculation uses packet header data to pick which slave to transmit the packet. For IP traffic, IP header source and destination fields are used in the calculation. For IP + TCP/UDP traffic, source and destination ports are included in the hash calculation. Traffic for a given conversation flow will always hash to the same slave. Many flows will be distributed over all the slaves to load balance the total traffic. In a failover event, the hash calculation is adjusted to steer traffic over available slaves.

## Configuration Files

- /etc/network/interfaces

## Useful Links

- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bonding>
- 802.3ad (Accessible writeup)
- Link aggregation from Wikipedia

## Caveats and Errata

- An interface cannot belong to multiple bonds.
- Slave ports within a bond should all be set to the same speed/duplex, and should match the link partner's slave ports.
- A bond cannot enslave VLAN subinterfaces. A bond can have subinterfaces, but not the other way around.

## Ethernet Bridging - VLANs

Ethernet bridges provide a means for hosts to communicate at layer 2. Bridge members can be individual physical interfaces, bonds or logical interfaces that traverse an 802.1Q VLAN trunk. Cumulus Linux does not put all ports into a bridge by default.

Cumulus Linux 2.5.0 introduced a new method for configuring bridges that are [VLAN-aware \(see page 235\)](#). The bridge driver in Cumulus Linux is capable of VLAN filtering, which allows for configurations that are similar to incumbent network devices. While Cumulus Linux supports Ethernet bridges in traditional mode Cumulus Networks recommends using [VLAN-aware \(see page 235\)](#) mode unless you are using VXLANS in your network.

For a comparison of traditional and VLAN-aware modes, read [this knowledge base article](#).



You can configure both VLAN-aware and traditional mode bridges on the same network in Cumulus Linux; however you should not have more than one VLAN-aware bridge on a given switch. If you are implementing [VXLANS \(see page 370\)](#), you **must** use traditional bridge mode.

## Contents

(Click to expand)

- [Contents \(see page 215\)](#)
- [Configuration Files \(see page 215\)](#)
- [Commands \(see page 215\)](#)
- [Creating a Bridge between Physical Interfaces \(see page 216\)](#)
  - [Creating the Bridge and Adding Interfaces \(see page 216\)](#)
  - [Showing and Verifying the Bridge Configuration \(see page 218\)](#)
- [Examining MAC Addresses \(see page 218\)](#)
- [Multiple Bridges \(see page 219\)](#)
- [Configuring an SVI \(Switch VLAN Interface\) \(see page 222\)](#)
  - [Showing and Verifying the Bridge Configuration \(see page 223\)](#)
- [Using Trunks in Traditional Bridging Mode \(see page 224\)](#)
  - [Trunk Example \(see page 225\)](#)
  - [Showing and Verifying the Trunk \(see page 226\)](#)
  - [Additional Examples \(see page 226\)](#)
- [Configuration Files \(see page 226\)](#)
- [Useful Links \(see page 227\)](#)
- [Caveats and Errata \(see page 227\)](#)

## Configuration Files

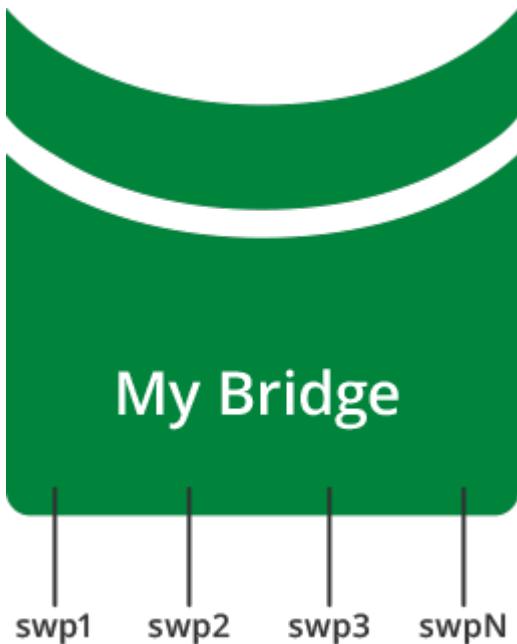
- [/etc/network/interfaces](#)

## Commands

- [brctl](#)
- [bridge](#)
- [ip addr](#)
- [ip link](#)

## ***Creating a Bridge between Physical Interfaces***

The basic use of bridging is to connect all of the physical and logical interfaces in the system into a single layer 2 domain.



## ***Creating the Bridge and Adding Interfaces***

You statically manage bridge configurations in `/etc/network/interfaces`. The following configuration snippet details an example bridge used throughout this chapter, explicitly enabling [spanning tree \(see page 172\)](#) and setting the bridge MAC address ageing timer. First, create a bridge with a descriptive name of 15 characters or fewer. Then add the logical interfaces (`bond0`) and physical interfaces (`swp5`, `swp6`) to assign to that bridge.

```
auto my_bridge
iface my_bridge
    bridge-ports bond0 swp5 swp6
    bridge-ageing 150
    bridge-stp on
```

Keyword	Explanation
bridge-ports	List of logical and physical ports belonging to the logical bridge.
bridge-ageing	Maximum amount of time before a MAC addresses learned on the bridge expires from the bridge MAC cache. The default value is 300 seconds.

Keyword	Explanation
bridge-stp	<p>Enables spanning tree protocol on this bridge. The default spanning tree mode is Per VLAN Rapid Spanning Tree Protocol (PVRST).</p> <p>For more information on spanning-tree configurations see the configuration section: <a href="#">Spanning Tree and Rapid Spanning Tree (see page 172)</a>.</p>

To bring up the bridge `my_bridge`, use the `ifreload` command:

```
cumulus@switch:~$ sudo ifreload -a
```

#### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To create the bridge and interfaces on the bridge, run:

```
cumulus@switch:~$ sudo brctl addbr my_bridge

cumulus@switch:~$ sudo brctl addif my_bridge bond0 swp5 swp6

cumulus@switch:~$ sudo brctl show
bridge name          bridge id      STP enabled    interfaces
my_bridge            8000.44383900129b  yes           bond0
                           swp5
                           swp6
```

```
cumulus@switch:~$ sudo ip link set up dev my_bridge
```

```
cumulus@switch:~$ sudo ip link set up dev bond0
```

```
cumulus@switch:~$ sudo for I in {5..6}; do ip link set up dev swp$I; done
```

## Showing and Verifying the Bridge Configuration

```
cumulus@switch:~$ ip link show my_bridge
56: my_bridge: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP mode DEFAULT
link/ether 44:38:39:00:12:9b brd ff:ff:ff:ff:ff:ff
```



Do not try to bridge the management port, eth0, with any switch ports (like swp0, swp1, and so forth). For example, if you created a bridge with eth0 and swp1, it will **not** work.

Using netshow to Display Bridge Information

`netshow` is a Cumulus Linux tool for retrieving information about your network configuration.

```
cumulus@switch$ netshow interface bridge
      Name      Speed      Mtu      Mode      Summary
      --  -----  -----  -----  -----
UP   my_bridge    N/A      1500  Bridge/L2  Untagged: bond0, swp5-6
                                         Root Port: bond0
                                         VlanID: Untagged
```

## Bridge Interface MAC Address and MTU

A bridge is a logical interface with a MAC address and an [MTU \(see page 152\)](#) (maximum transmission unit). The bridge MTU is the minimum MTU among all its members. The bridge's MAC address is inherited from the first interface that is added to the bridge as a member. The bridge MAC address remains unchanged until the member interface is removed from the bridge, at which point the bridge will inherit from the next member interface, if any. The bridge can also be assigned an IP address, as discussed later in this section.

## Examining MAC Addresses

A bridge forwards frames by looking up the destination MAC address. A bridge learns the source MAC address of a frame when the frame enters the bridge on an interface. After the MAC address is learned, the bridge maintains an age for the MAC entry in the bridge table. The age is refreshed when a frame is seen again with the same source MAC address. When a MAC is not seen for greater than the MAC ageing time, the MAC address is deleted from the bridge table.

The following shows the MAC address table of the example bridge. Notice that the `is local?` column indicates if the MAC address is the interface's own MAC address (`is local` is yes), or if it is learned on the interface from a packet's source MAC (where `is local` is no):

```
cumulus@switch:~$ sudo brctl showmacs my_bridge
  port name mac addr          is local?      ageing timer
```

swp4	06:90:70:22:a6:2e	no	19.47
swp1	12:12:36:43:6f:9d	no	40.50
bond0	2a:95:22:94:d1:f0	no	1.98
swp1	44:38:39:00:12:9b	yes	0.00
swp2	44:38:39:00:12:9c	yes	0.00
swp3	44:38:39:00:12:9d	yes	0.00
swp4	44:38:39:00:12:9e	yes	0.00
bond0	44:38:39:00:12:9f	yes	0.00
swp2	90:e2:ba:2c:b1:94	no	12.84
swp2	a2:84:fe:fc:bf:cd	no	9.43

You can use the `bridge fdb` command to display the MAC address table as well:

```
cumulus@switch:~$ bridge fdb show
44:38:39:00:12:9c dev swp2 VLAN 0 master bridge-A permanent
44:38:39:00:12:9b dev swp1 VLAN 0 master bridge-A permanent
44:38:39:00:12:9e dev swp4 VLAN 0 master bridge-B permanent
44:38:39:00:12:9d dev swp3 VLAN 0 master bridge-B permanent
```

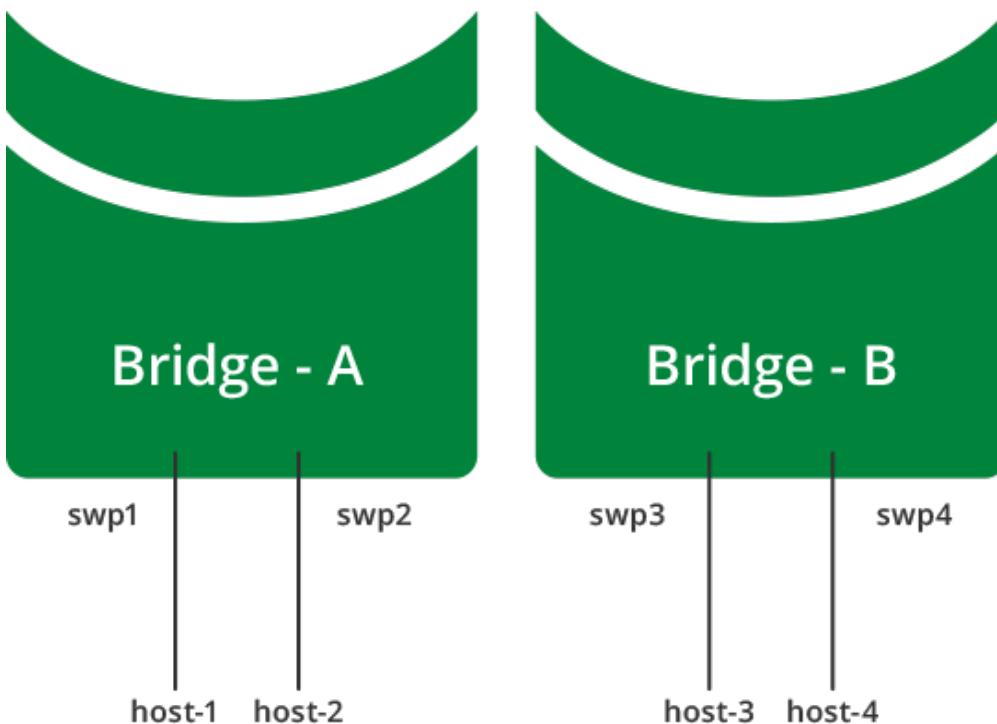


You can clear a MAC address from the table using the `bridge fdb` command:

```
cumulus@switch:~$ sudo bridge fdb del 44:38:39:00:12:9c dev swp2
```

## Multiple Bridges

Sometimes it is useful to logically divide a switch into multiple layer 2 domains, so that hosts in one domain can communicate with other hosts in the same domain but not in other domains. You can achieve this by configuring multiple bridges and putting different sets of interfaces in the different bridges. In the following example, host-1 and host-2 are connected to the same bridge (bridge-A), while host-3 and host-4 are connected to another bridge (bridge-B). host-1 and host-2 can communicate with each other, so can host-3 and host-4, but host-1 and host-2 cannot communicate with host-3 and host-4.



To configure multiple bridges, edit `/etc/network/interfaces`:

```
auto bridge-A
iface bridge-A
    bridge-ports swp1 swp2
    bridge-stp on

auto bridge-B
iface bridge-B
    bridge-ports swp3 swp4
    bridge-stp on
```

To bring up the bridges bridge-A and bridge-B, use the `ifreload` command:

```
cumulus@switch:~$ sudo ifreload -a
```

#### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

```

cumulus@switch:~$ sudo brctl addbr bridge-A

cumulus@switch:~$ sudo brctl addif bridge-A swp1 swp2

cumulus@switch:~$ sudo brctl addbr bridge-B

cumulus@switch:~$ sudo brctl addif bridge-B swp3 swp4

cumulus@switch:~$ sudo for I in {1..4}; do ip link set up dev swp$I; done

cumulus@switch:~$ sudo ip link set up dev bridge-A

cumulus@switch:~$ sudo ip link set up dev bridge-B

cumulus@switch:~$ sudo brctl show
  bridge name      bridge id          STP enabled    interfaces
  bridge-A        8000.44383900129b    yes           swp1
                                         swp2
  bridge-B        8000.44383900129d    yes           swp3
                                         swp4

cumulus@switch:~$ ip link show bridge-A
97: bridge-A: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
  state UP mode DEFAULT
    link/ether 70:72:cf:9d:4e:35 brd ff:ff:ff:ff:ff:ff
cumulus@switch:~$ ip link show bridge-B
98: bridge-B: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
  state UP mode DEFAULT
    link/ether 70:72:cf:9d:4e:37 brd ff:ff:ff:ff:ff:ff

```

### Using netshow to Display the Bridges

`netshow` is a Cumulus Linux tool for retrieving information about your network configuration.

```

cumulus@switch$ netshow interface bridge
  Name      Speed      Mtu      Mode      Summary
  --  -----
  UP  bridge-A   N/A       1500    Bridge/L2  Untagged: swp1-2
                                         Root Port: swp2
                                         VlanID: Untagged
  UP  bridge-B   N/A       1500    Bridge/L2  Untagged: swp3-4
                                         Root Port: swp3
                                         VlanID: Untagged

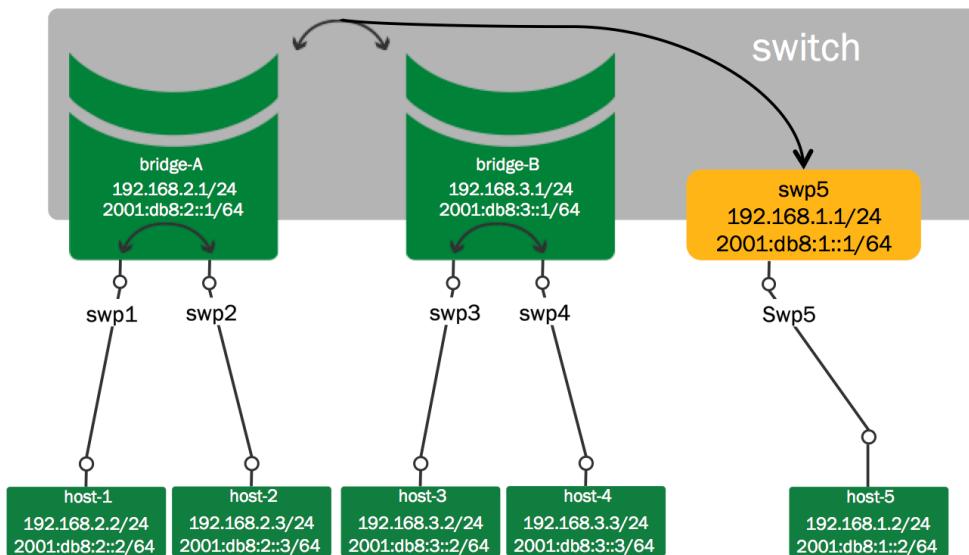
```

## Configuring an SVI (Switch VLAN Interface)

A bridge creates a layer 2 forwarding domain for hosts to communicate. A bridge can be assigned an IP address — typically of the same subnet as the hosts that are members of the bridge — and participate in routing topologies. This enables hosts within a bridge to communicate with other hosts outside the bridge through layer 3 routing.



When an interface is added to a bridge, it ceases to function as a router interface, and the IP address on the interface, if any, becomes unreachable.



The configuration for the two bridges example looks like the following:

```

auto swp5
iface swp5
  address 192.168.1.1/24
  address 2001:DB8:1::1/64
auto bridge-A
iface bridge-A
  address 192.168.2.1/24
  address 2001:DB8:2::1/64
  bridge-ports swp1 swp2
  bridge-stp on
auto bridge-B
iface bridge-B
  address 192.168.3.1/24
  address 2001:DB8:3::1/64
  bridge-ports swp3 swp4
  bridge-stp on

```

To bring up swp5 and bridges bridge-A and bridge-B, use the `ifreload` command:

```
cumulus@switch:~$ sudo ifreload -a
```

## ***Showing and Verifying the Bridge Configuration***

```
cumulus@switch$ ip addr show bridge-A
106: bridge-A: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
      state UP
        link/ether 70:72:cf:9d:4e:35 brd ff:ff:ff:ff:ff:ff
          inet 192.168.2.1/24 scope global bridge-A
            inet6 2001:db8:2::1/64 scope global
              valid_lft forever preferred_lft forever
              inet6 fe80::7272:ffff:fe9d:4e35/64 scope link
                valid_lft forever preferred_lft forever
```

```
cumulus@switch$ ip addr show bridge-B
107: bridge-B: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
      state UP
        link/ether 70:72:cf:9d:4e:37 brd ff:ff:ff:ff:ff:ff
          inet 192.168.3.1/24 scope global bridge-B
            inet6 2001:db8:3::1/64 scope global
              valid_lft forever preferred_lft forever
              inet6 fe80::7272:ffff:fe9d:4e37/64 scope link
                valid_lft forever preferred_lft forever
```

To see all the routes on the switch use the `ip route show` command:

```
cumulus@switch$ ip route show
192.168.1.0/24 dev swp5 proto kernel scope link src 192.168.1.2 dead
192.168.2.0/24 dev bridge-A proto kernel scope link src 192.168.2.1
192.168.3.0/24 dev bridge-B proto kernel scope link src 192.168.3.1
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

To add an IP address to a bridge:

```
cumulus@switch:~$ sudo ip addr add 192.0.2.101/24 dev bridge-A
cumulus@switch:~$ sudo ip addr add 192.0.2.102/24 dev bridge-B
```

Using netshow to Display the SVI

**netshow** is a Cumulus Linux tool for retrieving information about your network configuration.

```
cumulus@switch$ netshow interface bridge
  Name      Speed     Mtu     Mode      Summary
--  -----  -----  -----  -----
----- 
UP  bridge-A   N/A      1500    Bridge/L3  IP: 192.168.2.1/24, 2001:db8:2::1
/64
                                         Untagged: swp1-2
                                         Root Port: swp2
                                         VlanID: Untagged
UP  bridge-B   N/A      1500    Bridge/L3  IP: 192.168.3.1/24, 2001:db8:3::1
/64
                                         Untagged: swp3-4
                                         Root Port: swp3
                                         VlanID: Untagged
```

## Using Trunks in Traditional Bridging Mode

The [IEEE standard](#) for trunking is 802.1Q. The 802.1Q specification adds a 4 byte header within the Ethernet frame that identifies the VLAN of which the frame is a member.

802.1Q also identifies an *untagged* frame as belonging to the *native VLAN* (most network devices default their native VLAN to 1). The concept of native, non-native, tagged or untagged has generated confusion due to mixed terminology and vendor-specific implementations. Some clarification is in order:

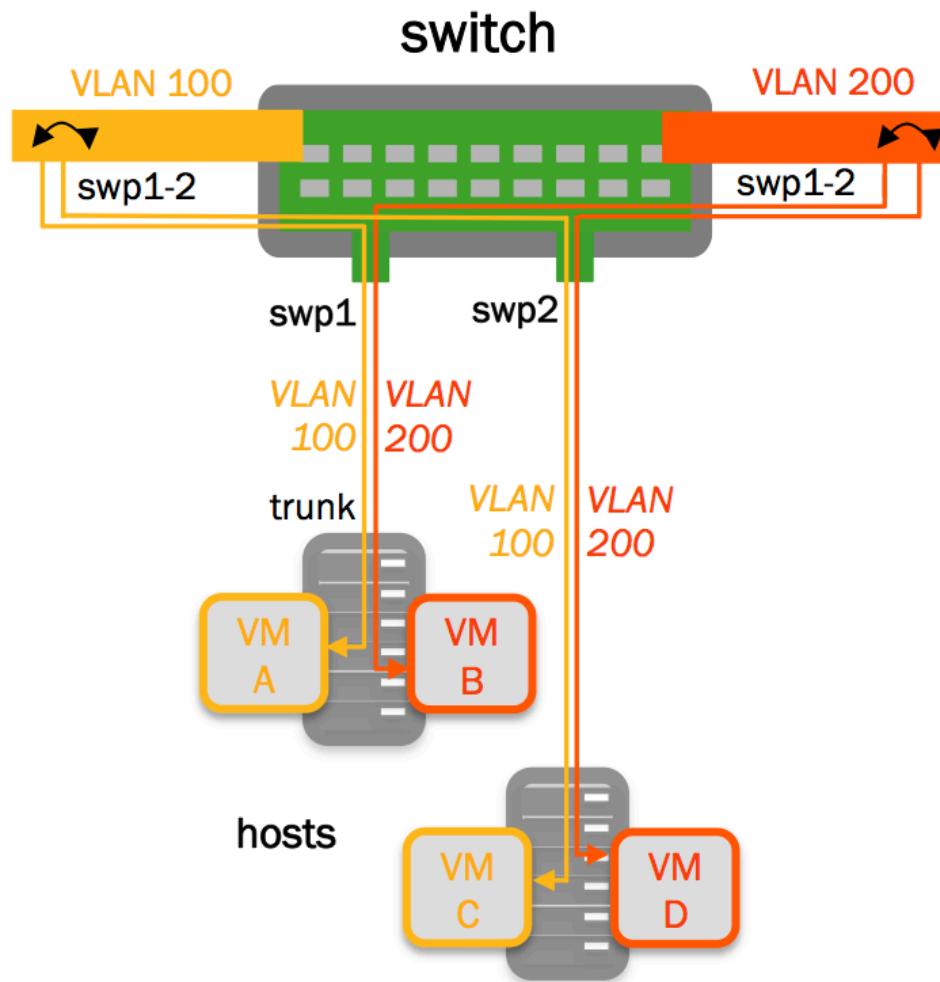
- A *trunk port* is a switch port configured to send and receive 802.1Q tagged frames.
- A switch sending an untagged (bare Ethernet) frame on a trunk port is sending from the native VLAN defined on the trunk port.
- A switch sending a tagged frame on a trunk port is sending to the VLAN identified by the 802.1Q tag.
- A switch receiving an untagged (bare Ethernet) frame on a trunk port places that frame in the native VLAN defined on the trunk port.
- A switch receiving a tagged frame on a trunk port places that frame in the VLAN identified by the 802.1Q tag.

A bridge in traditional mode has no concept of trunks, just tagged or untagged frames. With a trunk of 200 VLANs, there would need to be 199 bridges, each containing a tagged physical interface, and one bridge containing the native untagged VLAN. See the examples below for more information.



The interaction of tagged and un-tagged frames on the same trunk often leads to undesired and unexpected behavior. A switch that uses VLAN 1 for the native VLAN may send frames to a switch that uses VLAN 2 for the native VLAN, thus merging those two VLANs and their spanning tree state.

## Trunk Example



Configure the following in `/etc/network/interfaces`:

```
auto br-VLAN100
iface br-VLAN100
    bridge-ports swp1.100 swp2.100
    bridge-stp on
auto br-VLAN200
iface br-VLAN200
    bridge-ports swp1.200 swp2.200
    bridge-stp on
```

To bring up br-VLAN100 and br-VLAN200, use the `ifreload` command:

```
cumulus@switch:~$ sudo ifreload -a
```

## ***Showing and Verifying the Trunk***

```
cumulus@switch:~$ brctl show
bridge name bridge id          STP enabled interfaces
br-VLAN100  8000.7072cf9d4e35  yes      swp1.100
                                         swp2.100
br-VLAN200  8000.7072cf9d4e35  yes      swp1.200
                                         swp2.200
```

Using netshow to Display the Trunk

`netshow` is a Cumulus Linux tool for retrieving information about your network configuration.

```
cumulus@switch$ netshow interface bridge
  Name        Speed      Mtu      Mode      Summary
  --  -----
UP   br-VLAN100    N/A      1500    Bridge/L2  Tagged: swp1-2
                                         STP: rootSwitch(32768)
                                         VlanID: 100
UP   br-VLAN200    N/A      1500    Bridge/L2  Tagged: swp1-2
                                         STP: rootSwitch(32768)
                                         VlanID: 200
```

## ***Additional Examples***

You can find additional examples of VLAN tagging in [this chapter](#) (see page 227).

## ***Configuration Files***

- /etc/network/interfaces
- /etc/network/interfaces.d/
- /etc/network/if-down.d/
- /etc/network/if-post-down.d/
- /etc/network/if-pre-up.d/
- /etc/network/if-up.d/

## Useful Links

- [www.linuxfoundation.org/collaborate/workgroups/networking/bridge](http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge)
- [www.linuxfoundation.org/collaborate/workgroups/networking/vlan](http://www.linuxfoundation.org/collaborate/workgroups/networking/vlan)
- [www.linuxjournal.com/article/8172](http://www.linuxjournal.com/article/8172)

## Caveats and Errata

- The same bridge cannot contain multiple subinterfaces of the **same** port as members. Attempting to apply such a configuration will result in an error.

## VLAN Tagging

This article shows two examples of VLAN tagging, one basic and one more advanced. They both demonstrate the streamlined interface configuration from `ifupdown2`. For more information, see [Configuring and Managing Network Interfaces](#) (see page 134).

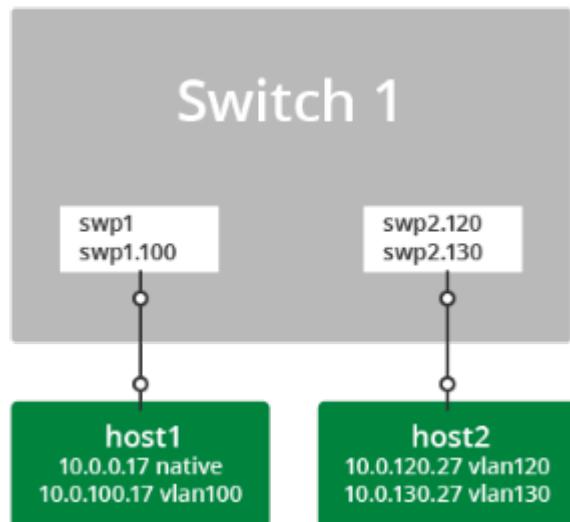
## Contents

(Click to expand)

- [VLAN Tagging, a Basic Example](#) (see page 227)
  - [Persistent Configuration](#) (see page 228)
- [VLAN Tagging, an Advanced Example](#) (see page 228)
  - [Persistent Configuration](#) (see page 229)
  - [VLAN Translation](#) (see page 233)

### ***VLAN Tagging, a Basic Example***

A simple configuration demonstrating VLAN tagging involves two hosts connected to a switch.



- *host1* connects to *swp1* with both untagged frames and with 802.1Q frames tagged for *vlan100*.
- *host2* connects to *swp2* with 802.1Q frames tagged for *vlan120* and *vlan130*.

## Persistent Configuration

To configure the above example persistently, configure `/etc/network/interfaces` like this:

```
# Config for host1

auto swp1
iface swp1

auto swp1.100
iface swp1.100

# Config for host2
# swp2 must exist to create the .1Q subinterfaces, but it is not
assigned an address

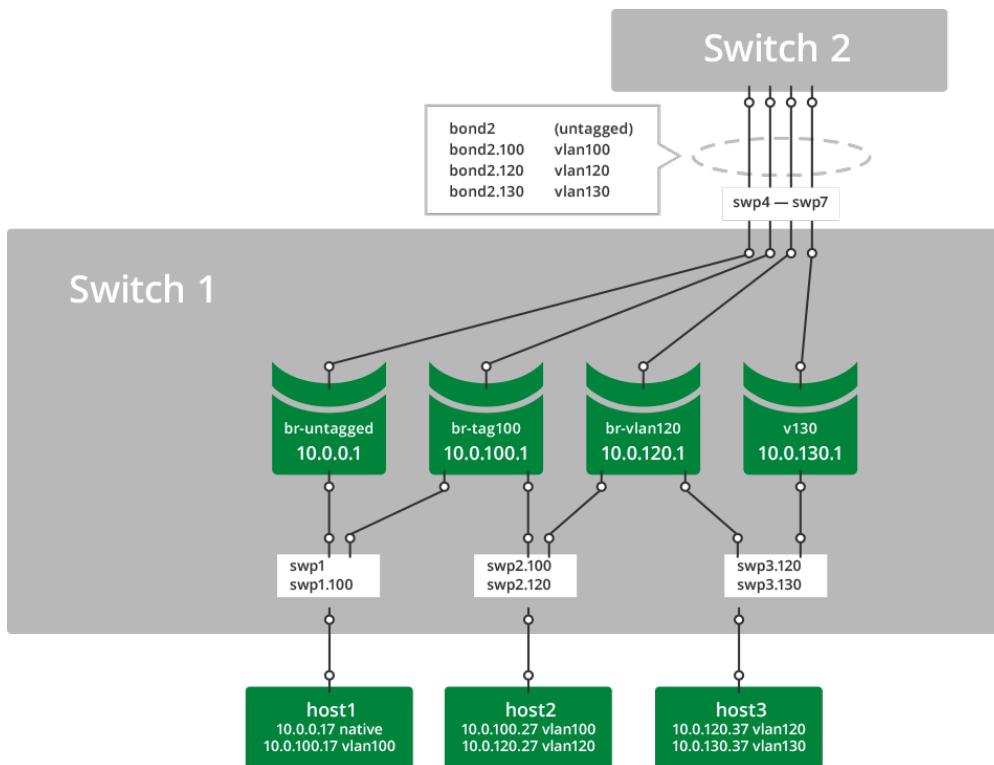
auto swp2
iface swp2

auto swp2.120
iface swp2.120

auto swp2.130
iface swp2.130
```

## VLAN Tagging, an Advanced Example

This example of VLAN tagging is more complex, involving three hosts and two switches, with a number of bridges and a bond connecting them all.



- *host1* connects to bridge *br-untagged* with bare Ethernet frames and to bridge *br-tag100* with 802.1q frames tagged for *vlan100*.
- *host2* connects to bridge *br-tag100* with 802.1q frames tagged for *vlan100* and to bridge *br-vlan120* with 802.1q frames tagged for *vlan120*.
- *host3* connects to bridge *br-vlan120* with 802.1q frames tagged for *vlan120* and to bridge *v130* with 802.1q frames tagged for *vlan130*.
- *bond2* carries tagged and untagged frames in this example.

Although not explicitly designated, the bridge member ports function as 802.1Q access ports and trunk ports. In the example above, comparing Cumulus Linux with a traditional Cisco device:

- *swp1* is equivalent to a trunk port with untagged and *vlan100*.
- *swp2* is equivalent to a trunk port with *vlan100* and *vlan120*.
- *swp3* is equivalent to a trunk port with *vlan120* and *vlan130*.
- *bond2* is equivalent to an EtherChannel in trunk mode with untagged, *vlan100*, *vlan120*, and *vlan130*.
- Bridges *br-untagged*, *br-tag100*, *br-vlan120*, and *v130* are equivalent to SVIs (switched virtual interfaces).

## Persistent Configuration

From `/etc/network/interfaces`:

```
# Config for host1 -----  
-----  
  
# swp1 does not need an iface section unless it has a specific  
setting,
```

```
# it will be picked up as a dependent of swp1.100.  
# And swp1 must exist in the system to create the .1q subinterfaces..  
# but it is not applied to any bridge..or assigned an address.  
  
auto swp1.100  
iface swp1.100  
  
# Config for host2  
# swp2 does not need an iface section unless it has a specific  
setting,  
# it will be picked up as a dependent of swp2.100 and swp2.120.  
# And swp2 must exist in the system to create the .1q subinterfaces..  
# but it is not applied to any bridge..or assigned an address.  
  
auto swp2.100  
iface swp2.100  
  
auto swp2.120  
iface swp2.120  
  
# Config for host3  
# swp3 does not need an iface section unless it has a specific  
setting,  
# it will be picked up as a dependent of swp3.120 and swp3.130.  
# And swp3 must exist in the system to create the .1q subinterfaces..  
# but it is not applied to any bridge..or assigned an address.  
  
auto swp3.120  
iface swp3.120  
  
auto swp3.130  
iface swp3.130  
  
# Configure the bond - - - - -  
- - - - -  
  
auto bond2  
iface bond2  
    bond-slaves glob swp4-7  
  
# configure the bridges - - - - -  
- - - - -  
  
auto br-untagged  
iface br-untagged  
    address 10.0.0.1/24  
    bridge-ports swp1 bond2  
    bridge-stp on  
  
auto br-tag100  
iface br-tag100  
    address 10.0.100.1/24
```

```

bridge-ports swp1.100 swp2.100 bond2.100
bridge-stp on

auto br-vlan120
iface br-vlan120
    address 10.0.120.1/24
bridge-ports swp2.120 swp3.120 bond2.120
bridge-stp on

auto v130
iface v130
    address 10.0.130.1/24
bridge-ports swp3.130 bond2.130
bridge-stp on

# -----

```

To verify:

```

cumulus@switch:~$ sudo mstptctl showbridge br-tag100
br-tag100 CIST info
  enabled      yes
  bridge id    8.000.44:38:39:00:32:8B
  designated root 8.000.44:38:39:00:32:8B
  regional root 8.000.44:38:39:00:32:8B
  root port    none
  path cost     0          internal path cost   0
  max age       20         bridge max age     20
  forward delay 15         bridge forward delay 15
  tx hold count 6          max hops        20
  hello time    2          ageing time      300
  force protocol version    rstp
  time since topology change 333040s
  topology change count      1
  topology change             no
  topology change port       swp2.100
  last topology change port None

```

```

cumulus@switch:~$ sudo mstptctl showportdetail br-tag100 | grep -B 2
state
br-tag100:bond2.100 CIST info
  enabled      yes           role
Designated
  port id     8.003         state
forwarding
--
br-tag100:swp1.100 CIST info

```

```

      enabled          yes           role
Designated
  port id        8.001         state
forwarding
--
br-tag100:swp2.100 CIST info
  enabled          yes           role
Designated
  port id        8.002         state
forwarding
  
```

```

cumulus@switch:~$ cat /proc/net/vlan/config
VLAN Dev name | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
bond2.100 | 100 | bond2
bond2.120 | 120 | bond2
bond2.130 | 130 | bond2
swp1.100 | 100 | swp1
swp2.100 | 100 | swp2
swp2.120 | 120 | swp2
swp3.120 | 120 | swp3
swp3.130 | 130 | swp3
  
```

```

cumulus@switch:~$ cat /proc/net/bonding/bond2
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer3+4 (1)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: fast
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
  Aggregator ID: 3
  Number of ports: 4
  Actor Key: 33
  Partner Key: 33
  Partner Mac Address: 44:38:39:00:32:cf

Slave Interface: swp4
MII Status: up
Speed: 10000 Mbps
Duplex: full
  
```

```

Link Failure Count: 0
Permanent HW addr: 44:38:39:00:32:8e
Aggregator ID: 3
Slave queue ID: 0

Slave Interface: swp5
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 44:38:39:00:32:8f
Aggregator ID: 3
Slave queue ID: 0

Slave Interface: swp6
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 44:38:39:00:32:90
Aggregator ID: 3
Slave queue ID: 0

Slave Interface: swp7
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 44:38:39:00:32:91
Aggregator ID: 3
Slave queue ID: 0

```



A single bridge cannot contain multiple subinterfaces of the **same** port as members. Attempting to apply such a configuration will result in an error:

```

cumulus@switch:~$ sudo brctl addbr another_bridge
cumulus@switch:~$ sudo brctl addif another_bridge swp9 swp9.100
bridge cannot contain multiple subinterfaces of the same port: swp9,
swp9.100

```

## VLAN Translation

By default, Cumulus Linux does not allow VLAN subinterfaces associated with different VLAN IDs to be part of the same bridge. Base interfaces are not explicitly associated with any VLAN IDs and are exempt from this restriction:

```
cumulus@switch:~$ sudo brctl addbr br_mix

cumulus@switch:~$ sudo ip link add link swp10 name swp10.100 type vlan id
100
cumulus@switch:~$ sudo ip link add link swp11 name swp11.200 type vlan id
200

cumulus@switch:~$ sudo brctl addif br_mix swp10.100 swp11.200
can't add swp11.200 to bridge br_mix: Invalid argument
```

In some cases, it may be useful to relax this restriction. For example, two servers may be connected to the switch using VLAN trunks, but the VLAN numbering provisioned on the two servers are not consistent. You can choose to just bridge two VLAN subinterfaces of different VLAN IDs from the servers. You do this by enabling the `sysctl net.bridge.bridge-allow-multiple-vlans`. Packets entering a bridge from a member VLAN subinterface will egress another member VLAN subinterface with the VLAN ID translated.



A bridge in [VLAN-aware mode](#) (see page 235) cannot have VLAN translation enabled for it; only bridges configured in traditional mode can utilize VLAN translation.

The following example enables the VLAN translation `sysctl`:

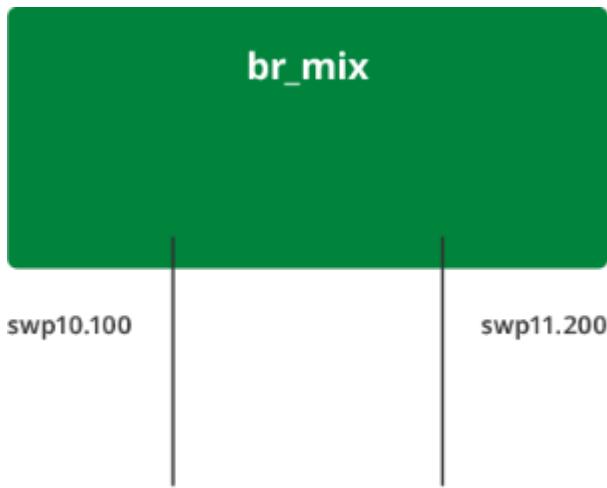
```
cumulus@switch:~$ echo net.bridge.bridge-allow-multiple-vlans = 1 | sudo
tee /etc/sysctl.d/multiple_vlans.conf
net.bridge.bridge-allow-multiple-vlans = 1
cumulus@switch:~$ sudo sysctl -p /etc/sysctl.d/multiple_vlans.conf
net.bridge.bridge-allow-multiple-vlans = 1
```

If the `sysctl` is enabled and you want to disable it, run the above example, setting the `sysctl net.bridge.bridge-allow-multiple-vlans` to 0.

Once the `sysctl` is enabled, ports with different VLAN IDs can be added to the same bridge. In the following example, packets entering the bridge `br-mix` from `swp10.100` will be bridged to `swp11.200` with the VLAN ID translated from 100 to 200:

```
cumulus@switch:~$ sudo brctl addif br_mix swp10.100 swp11.200

cumulus@switch:~$ sudo brctl show br_mix
bridge name      bridge id          STP enabled    interfaces
br_mix          8000.4438390032bd    yes           swp10.100
                                         swp11.200
```



## VLAN-aware Bridge Mode for Large-scale Layer 2 Environments

Cumulus Linux bridge driver supports two configuration modes, one that is VLAN-aware, and one that follows a more traditional Linux bridge model.

For traditional Linux bridges, the kernel supports VLANs in the form of VLAN subinterfaces. Enabling bridging on multiple VLANs means configuring a bridge for each VLAN and, for each member port on a bridge, creating one or more VLAN subinterfaces out of that port. This mode poses scalability challenges in terms of configuration size as well as boot time and run time state management, when the number of ports times the number of VLANs becomes large.

The VLAN-aware mode in Cumulus Linux implements a configuration model for large-scale L2 environments, with **one single instance** of [Spanning Tree](#) (see page 172). Each physical bridge member port is configured with the list of allowed VLANs as well as its port VLAN ID (either PVID or native VLAN — see below). MAC address learning, filtering and forwarding are *VLAN-aware*. This significantly reduces the configuration size, and eliminates the large overhead of managing the port/VLAN instances as subinterfaces, replacing them with lightweight VLAN bitmaps and state updates.



You can configure both VLAN-aware and traditional mode bridges on the same network in Cumulus Linux; however you should not have more than one VLAN-aware bridge on a given switch. If you are implementing [VXLANs](#) (see page 370), you **must** use non-aware bridges.

## Contents

(Click to expand)

- [Contents \(see page 235\)](#)
- [Defining VLAN-aware Bridge Attributes \(see page 236\)](#)
- [Basic Trunking \(see page 236\)](#)
- [VLAN Filtering/VLAN Pruning \(see page 237\)](#)
- [Untagged/Access Ports \(see page 237\)](#)
  - [Dropping Untagged Frames \(see page 238\)](#)
- [VLAN Layer 3 Addressing - Switch Virtual Interfaces and other VLAN Attributes \(see page 239\)](#)
- [Using the glob Keyword to Configure Multiple Ports in a Range \(see page 240\)](#)

- Example Configuration with Access Ports and Pruned VLANs (see page 240)
- Example Configuration with Bonds (see page 241)
- Converting a Traditional Bridge to VLAN-aware or Vice Versa (see page 243)
- Caveats and Errata (see page 243)

## Defining VLAN-aware Bridge Attributes

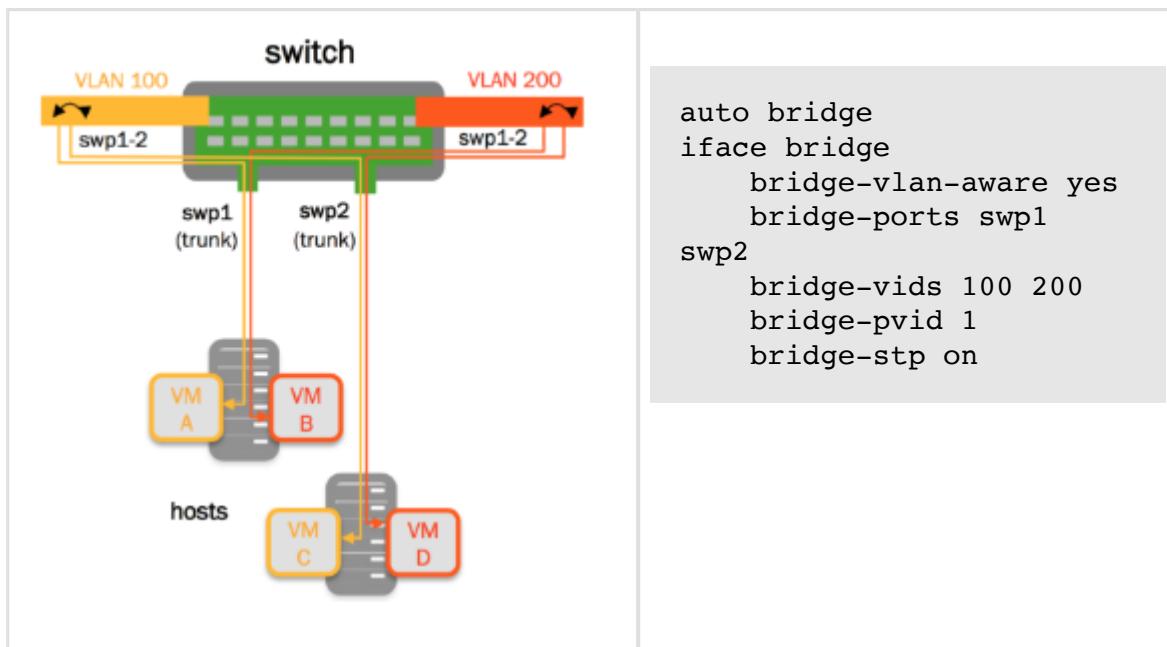
To configure a VLAN-aware bridge, include the `bridge-vlan-aware` attribute, setting it to yes. Name the bridge `bridge` to help ensure it is the only VLAN-aware bridge on the switch. The following attributes are useful for configuring VLAN-aware bridges:

- `bridge-vlan-aware`: Set to yes to indicate that the bridge is in VLAN-aware mode.
- `bridge-pvid`: A PVID is the bridge's *Primary VLAN Identifier*. The PVID defaults to 1; specifying the PVID identifies that VLAN as the native VLAN.
- `bridge-vids`: A VID is the *VLAN Identifier*, which declares the VLANs associated with this bridge.
- `bridge-access`: Declares the physical switch port as an *access port*. Access ports ignore all tagged packets; put all untagged packets into the `bridge-pvid`.
- `bridge-allow-untagged`: When set to *no*, it drops any untagged frames for a given switch port.

For a definitive list of bridge attributes, run `ifquery --syntax-help` and look for the entries under `bridge`, `bridgevlan` and `mstpcctl`.

## Basic Trunking

A basic configuration for a VLAN-aware bridge configured for STP that contains two switch ports looks like this:



The above configuration actually includes 3 VLANs: the tagged VLANs 100 and 200 and the untagged (native) VLAN of 1.



The `bridge-pvid 1` is implied by default. You do not have to specify `bridge-pvid`. And while it does not hurt the configuration, it helps other users for readability.

The following configurations are identical to each other and the configuration above:

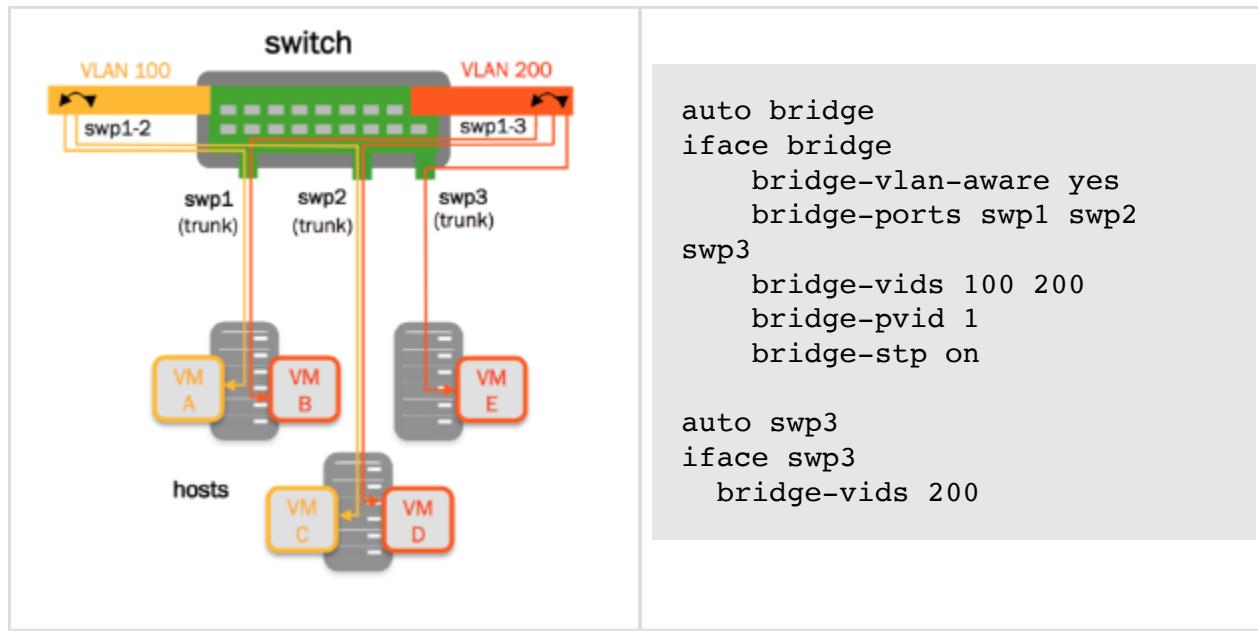
```
auto bridge
iface bridge
    bridge-vlan-
        aware yes
    bridge-ports
    swp1 swp2
    bridge-vids
    1 100 200
    bridge-stp on
```

```
auto bridge
iface bridge
    bridge-vlan-
        aware yes
    bridge-ports
    swp1 swp2
    bridge-vids
    1 100 200
    bridge-pvid 1
    bridge-stp on
```

```
auto bridge
iface bridge
    bridge-vlan-
        aware yes
    bridge-ports
    swp1 swp2
    bridge-vids
    100 200
    bridge-stp on
```

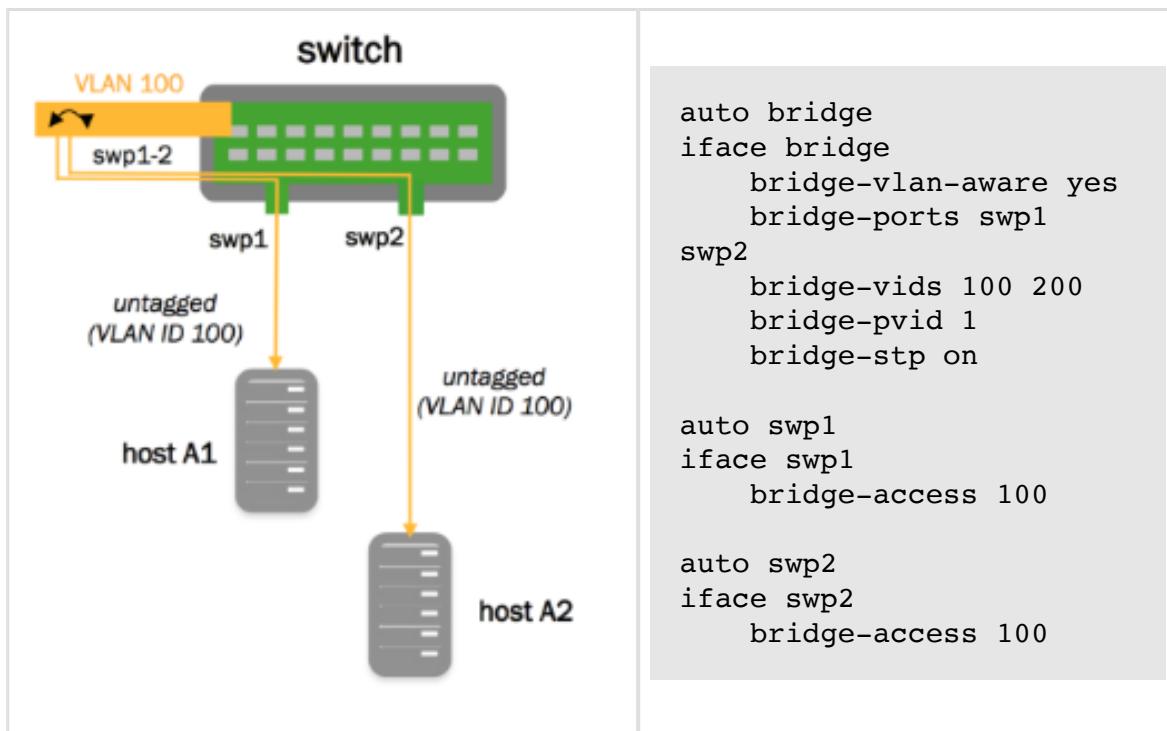
## VLAN Filtering/VLAN Pruning

By default, the bridge port inherits the bridge VIDs. A port's configuration can override the bridge VIDs. Do this by specifying port-specific VIDs using the `bridge-vids` attribute.



## Untagged/Access Ports

As described above, access ports ignore all tagged packets. In the configuration below, swp1 and swp2 are configured as access ports. All untagged traffic goes to the specified VLAN, which is VLAN 100 in the example below.



## Dropping Untagged Frames

With VLAN-aware bridge mode, it's possible to configure a switch port so it drops any untagged frames. To do this, add `bridge-allow-untagged no` under the switch port stanza in `/etc/network/interfaces`. This leaves the bridge port without a PVID and drops untagged packets.

Consider the following example bridge:

```

auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports swp1 swp9
    bridge-vids 2-100
    bridge-pvid 101
    bridge-stp on

```

Here is the VLAN membership for that configuration:

```

cumulus@switch$ bridge -c vlan show
portvlan ids
swp1 101 PVID Egress Untagged
2-100

swp9 101 PVID Egress Untagged

```

2-100

```
bridge 101
```

To configure swp9 to drop untagged frames, add `bridge-allow-untagged no`:

```
auto swp9
iface swp9
    bridge-allow-untagged no
```

When you check VLAN membership for that port, it shows that there is **no** untagged VLAN.

```
cumulus@switch$ bridge -c vlan show
portvlan ids
swp1 101 PVID Egress Untagged
      2-100

swp9 2-100

bridge 101
```

## ***VLAN Layer 3 Addressing - Switch Virtual Interfaces and other VLAN Attributes***

When configuring the VLAN attributes for the bridge, put the attributes in a separate stanza for each VLAN interface: <bridge>.<vlanid>. If you are configuring the SVI for the native VLAN, you must declare the native VLAN in its own stanza and specify its IP address. Specifying the IP address in the bridge stanza itself returns an error.

```
auto bridge.100
iface bridge.100
    address 192.168.10.1/24
    address 2001:db8::1/32
    hwaddress 44:38:39:ff:00:00

# 12 attributes
auto bridge.100
vlan bridge.100
    bridge-igmp-querier-src 172.16.101.1
```



The `vlan` object type in the L2 attributes section above is used to specify layer 2 VLAN attributes only. Currently, the only supported layer 2 VLAN attribute is `bridge-igmp-querier-src`.

However, if your switch is configured for multicast routing, then you do not need to specify `bridge-igmp-querier-src`, as there is no need for a static IGMP querier configuration on the switch. Otherwise, the static IGMP querier configuration helps to probe the hosts to refresh their IGMP reports.

You can specify a range of VLANs as well. For example:

```
auto bridge.[1-2000]
vlan bridge.[1-2000]
  ATTRIBUTE VALUE
```

## **Using the `glob` Keyword to Configure Multiple Ports in a Range**

The `glob` keyword referenced in the `bridge-ports` attribute indicates that `swp1` through `swp52` are part of the bridge, which is a short cut that saves you from enumerating each port individually:

```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-52
  bridge-stp on
  bridge-vids 310 700 707 712 850 910
```

## **Example Configuration with Access Ports and Pruned VLANs**

The following example contains an access port and a switch port that is *pruned*; that is, it only sends and receives traffic tagged to and from a specific set of VLANs declared by the `bridge-vids` attribute. It also contains other switch ports that send and receive traffic from all the defined VLANs.

```
# ports swp3-swp48 are trunk ports which inherit vlans from the
'bridge'
# ie vlans 310,700,707,712,850,910
#
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-52
  bridge-stp on
  bridge-vids 310 700 707 712 850 910

auto swp1
iface swp1
  mstpctl-portadminegedge yes
```

```

mstpctl-bpduguard yes
bridge-access 310

# The following is a trunk port that is "pruned".
# native vlan is 1, but only .1q tags of 707, 712, 850 are
# sent and received
#
auto swp2
iface swp2
    mstpctl-portadminedge yes
    mstpctl-bpduguard yes
    bridge-vids 707 712 850

# The following port is the trunk uplink and inherits all vlans
# from 'bridge'; bridge assurance is enabled using 'portnetwork'
attribute
auto swp49
iface swp49
    mstpctl-portpathcost 10
    mstpctl-portnetwork yes

# The following port is the trunk uplink and inherits all vlans
# from 'bridge'; bridge assurance is enabled using 'portnetwork'
attribute
auto swp50
iface swp50
    mstpctl-portpathcost 0
    mstpctl-portnetwork yes

```

## **Example Configuration with Bonds**

This configuration demonstrates a VLAN-aware bridge with a large set of bonds. The bond configurations are generated from a [Mako](#) template.

```

#
# vlan-aware bridge with bonds example
#
# uplink1, peerlink and downlink are bond interfaces.
# 'bridge' is a vlan aware bridge with ports uplink1, peerlink
# and downlink (swp2-20).
#
# native vlan is by default 1
#
# 'bridge-vids' attribute is used to declare vlans.
# 'bridge-pvid' attribute is used to specify native vlans if other
than 1
# 'bridge-access' attribute is used to declare access port
#
auto lo
iface lo

```

```
auto eth0
iface eth0 inet dhcp

# bond interface
auto uplink1
iface uplink1
    bond-slaves swp32
    bridge-vids 2000-2079

# bond interface
auto peerlink
iface peerlink
    bond-slaves swp30 swp31
    bridge-vids 2000-2079 4094

# bond interface
auto downlink
iface downlink
    bond-slaves swp1
    bridge-vids 2000-2079

#
# Declare vlans for all swp ports
# swp2-20 get vlans from 2004 to 2022.
# The below uses mako templates to generate iface sections
# with vlans for swp ports
#
%for port, vlanid in zip(range(2, 20), range(2004, 2022)) :
    auto swp${port}
    iface swp${port}
        bridge-vids ${vlanid}

%endfor

# svi vlan 4094
auto bridge.4094
iface bridge.4094
    address 11.100.1.252/24

# 12 attributes for vlan 4094
auto bridge.4094
vlan bridge.4094
    bridge-igmp-querier-src 172.16.101.1

#
# vlan-aware bridge
#
auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports uplink1 peerlink downlink glob swp2-20
```

```
bridge-stp on

# svi peerlink vlan
auto peerlink.4094
iface peerlink.4094
    address 192.168.10.1/30
    broadcast 192.168.10.3
```

## Converting a Traditional Bridge to VLAN-aware or Vice Versa

You cannot automatically convert a traditional bridge to/from a VLAN-aware bridge simply by changing the configuration in the `/etc/network/interfaces` file. If you need to change the mode for a bridge, do the following:

1. Delete the traditional mode bridge from the configuration and bring down all its member switch port interfaces.
2. Create a new VLAN-aware bridge, as described above.
3. Bring up the bridge.

These steps assume you are converting a traditional mode bridge to a VLAN-aware one. To do the opposite, delete the VLAN-aware bridge in step 1, and create a new traditional mode bridge in step 2.

## Caveats and Errata

- **STP:** Because Spanning Tree and Rapid Spanning Tree (see page 172) (STP) are enabled on a per-bridge basis, VLAN-aware mode essentially supports a single instance of STP across all VLANs. A common practice when using a single STP instance for all VLANs is to define all every VLAN on each switch in the spanning tree instance. `mstpd` continues to be the user space protocol daemon, and Cumulus Linux supports RSTP.
- **IGMP snooping:** IGMP snooping and group membership are supported on a per-VLAN basis, though the IGMP snooping configuration (including enable/disable, mrouter port and so forth) are defined on a per-bridge port basis.
- **VXLANs:** Use the traditional configuration mode for **VXLAN configuration** (see page 370).
- **Reserved VLAN range:** For hardware data plane internal operations, the switching silicon requires VLANs for every physical port, Linux bridge, and layer 3 subinterface. Cumulus Linux reserves a range of 1000 VLANs by default; this range is 3000-3999. In case any of your user-defined VLANs conflict with the default reserved range, you can modify the range, as long as the new range is a contiguous set of VLANs with IDs anywhere between 2 and 4094, and the minimum size of the range is 300 VLANs:

1. Edit `/etc/cumulus/switchd.conf`, uncomment `resv_vlan_range` and specify the new range.
2. Restart `switchd` (see page 128) (`sudo systemctl restart switchd.service`) for the new range to take effect.



While restarting `switchd`, all running ports will flap and forwarding will be interrupted (see page 128).

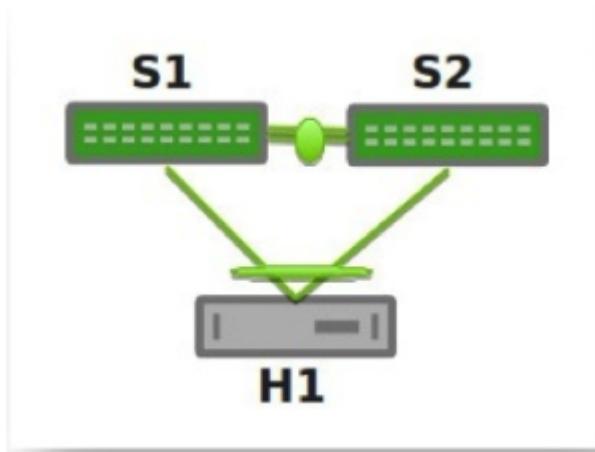
- **VLAN translation:** A bridge in VLAN-aware mode cannot have VLAN translation enabled for it; only bridges configured in [traditional mode \(see page 214\)](#) can utilize VLAN translation.

## Multi-Chassis Link Aggregation - MLAG

Multi-Chassis Link Aggregation, or MLAG, enables a server or switch with a two-port bond (such as a link aggregation group/LAG, EtherChannel, port group, or trunk) to connect those ports to different switches and operate as if they are connected to a single, logical switch. This provides greater redundancy and greater system throughput.

Dual-connected devices can create LACP bonds that contain links to each physical switch. Thus, active-active links from the dual-connected devices are supported even though they are connected to two different physical switches.

A basic setup looks like this:



The two switches, S1 and S2, known as *peer switches*, cooperate so that they appear as a single device to host H1's bond. H1 distributes traffic between the two links to S1 and S2 in any manner that you configure on the host. Similarly, traffic inbound to H1 can traverse S1 or S2 and arrive at H1.

### Contents

This chapter covers ...

- MLAG Requirements (see page 245)
- LACP and Dual-Connectedness (see page 246)
- Understanding Switch Roles (see page 247)
- Configuring MLAG (see page 247)
  - Reserved MAC Address Range (see page 248)
  - Configuring the Host or Switch (see page 248)
  - Configuring the Interfaces (see page 248)
  - Example MLAG Configuration (see page 249)
  - Configuring MLAG with a Traditional Mode Bridge (see page 253)
  - Using the clagd Command Line Interface (see page 253)
- Peer Link Interfaces and the protodown State (see page 254)

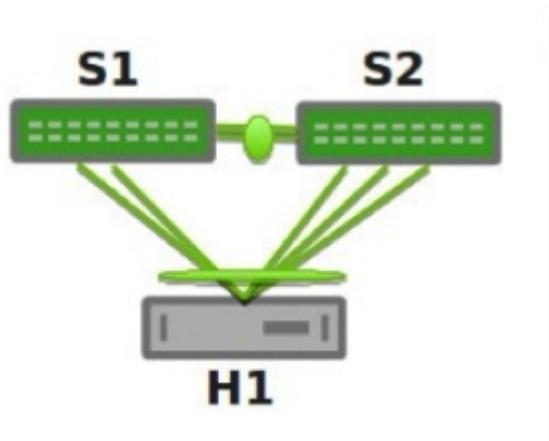
- Specifying a Backup Link (see page 254)
- Monitoring Dual-Connected Peers (see page 255)
- Configuring Layer 3 Routed Uplinks (see page 256)
- IGMP Snooping with MLAG (see page 257)
- Monitoring the Status of the clagd Service (see page 257)
- MLAG Best Practices (see page 258)
  - Understanding MTU in an MLAG Configuration (see page 258)
  - Sizing the Peerlink (see page 258)
- STP Interoperability with MLAG (see page 260)
  - Debugging STP with MLAG (see page 260)
  - Best Practices for STP with MLAG (see page 260)
- Troubleshooting MLAG (see page 261)
- Caveats and Errata (see page 261)
- Configuration Files (see page 261)

## **MLAG Requirements**

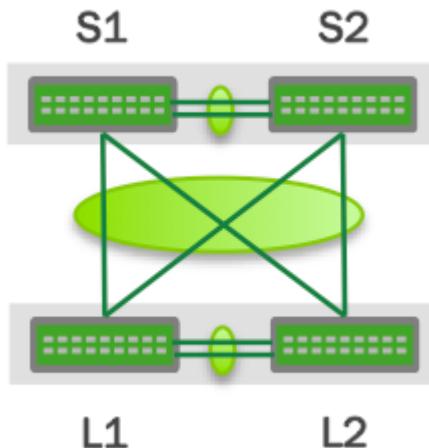
MLAG has these requirements:

- There must be a direct connection between the two peer switches implementing MLAG (S1 and S2). This is typically a bond for increased reliability and bandwidth.
- There must be only two peer switches in one MLAG configuration, but you can have multiple configurations in a network for *switch-to-switch MLAG* (see below).
- The peer switches implementing MLAG must be running Cumulus Linux version 2.5 or later.
- You must specify a unique `clag-id` for every dual-connected bond on each peer switch; the value must be between 1 and 65535 and must be the same on both peer switches in order for the bond to be considered *dual-connected*.
- The dual-connected devices (hosts or switches) must use LACP (IEEE 802.3ad/802.1ax) to form the bond. The peer switches must also use LACP.

More elaborate configurations are also possible. The number of links between the host and the switches can be greater than two, and does not have to be symmetrical:



Additionally, since S1 and S2 appear as a single switch to other bonding devices, pairs of MLAG switches can also be connected to each other in a switch-to-switch MLAG setup:

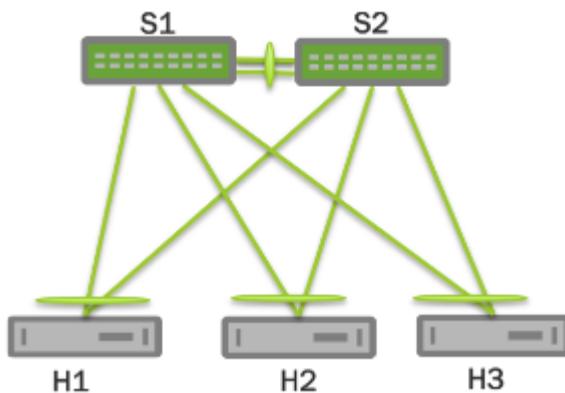


In this case, L1 and L2 are also MLAG peer switches, and thus present a two-port bond from a single logical system to S1 and S2. S1 and S2 do the same as far as L1 and L2 are concerned. For a switch-to-switch MLAG configuration, each switch pair must have a unique system MAC address. In the above example, switches L1 and L2 each have the same system MAC address configured. Switch pair S1 and S2 each have the same system MAC address configured; however, it is a different system MAC address than the one used by the switch pair L1 and L2.

## LACP and Dual-Connectedness

In order for MLAG to operate correctly, the peer switches must know which links are *dual-connected*, or are connected to the same host or switch. To do this, specify a **clag-id** for every dual-connected bond on each peer switch; the **clag-id** must be the same for the corresponding bonds on both peer switches. [Link Aggregation Control Protocol \(LACP\)](#), the IEEE standard protocol for managing bonds, is used for verifying dual-connectedness. LACP runs on the dual-connected device and on each of the peer switches. On the dual-connected device, the only configuration requirement is to create a bond that will be managed by LACP.

On each of the peer switches the links connected to the dual-connected host or switch must be placed in the bond. This is true even if the links are a single port on each peer switch, where each port is placed into a bond, as shown below:



All of the dual-connected bonds on the peer switches have their system ID set to the MLAG system ID. Therefore, from the point of view of the hosts, each of the links in its bond is connected to the same system, and so the host will use both links.

Each peer switch periodically makes a list of the LACP partner MAC addresses of all of their bonds and sends that list to its peer (using the `cldagd` service; see below). The LACP partner MAC address is the MAC address of the system at the other end of a bond, which in the figure above would be hosts H1, H2 and H3. When a switch receives this list from its peer, it compares the list to the LACP partner MAC addresses on its switch. If any matches are found and the `cldag-id` for those bonds match, then that bond is a dual-connected bond. You can also find the LACP partner MAC address in the `/sys/class/net/<bondname>/bonding/ad_partner_mac sysfs` file for each bond.

## ***Understanding Switch Roles***

Each MLAG-enabled switch in the pair has a role. When the peering relationship is established between the two switches, one switch will be in *primary* role, and the other one will be in *secondary* role. When an MLAG-enabled switch is in the secondary role, it does not send STP BPDUs on dual-connected links; it only sends BPDUs on single-connected links. The switch in the primary role sends STP BPDUs on all single- and dual-connected links.

<b>Send BPDUs</b>	<b>Primary</b>	<b>Secondary</b>
Single-connected links	Yes	Yes
Dual-connected links	Yes	No

By default, the role is determined by comparing the MAC addresses of the two sides of the peering link; the switch with the lower MAC address assumes the primary role. You can override this by setting the priority configuration, either by specifying the `cldagd-priority` option in `/etc/network/interfaces`, or by using `cldagctl`. The switch with the lower priority value is given the primary role; the default value is 32768, and the range is 0 to 65535. Read the `cldagd(8)` and `cldagctl(8)` man pages for more information.

When the `cldagd` service is exited during switch reboot or the service is stopped in the primary switch, the peer switch that is in the secondary role will become primary. If the primary switch goes down without stopping the `cldagd` service for any reason or the peer link goes down, the secondary switch will **not** change its role. In case the peer switch is determined to be not alive, the switch in the secondary role will roll back the LACP system ID to be the bond interface MAC address instead of the `cldagd-sys-mac` and the switch in primary role uses the `cldagd-sys-mac` as the LACP system ID on the bonds.

## ***Configuring MLAG***

Configuring MLAG involves:

- On the dual-connected devices, create a bond that uses LACP.
- On each peer switch, configure the interfaces, including bonds, VLANs, bridges and peer links.



MLAG synchronizes the dynamic state between the two peer switches, but it does not synchronize the switch configurations. After modifying the configuration of one peer switch, you must make the same changes to the configuration on the other peer switch. This applies to all configuration changes, including:

- Port configuration: For example, VLAN membership, [MTU \(see page 258\)](#), and bonding parameters.

- Bridge configuration: For example, spanning tree parameters or bridge properties.
- Static address entries: For example, static FDB entries and static IGMP entries.
- QoS configuration: For example, ACL entries.

You can verify the configuration of VLAN membership using the `clagctl -v verifyvlans` command.

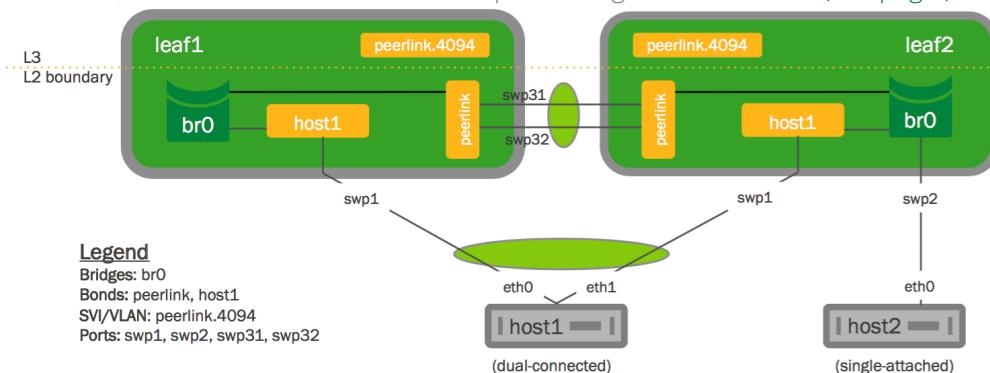
## Reserved MAC Address Range

In order to prevent MAC address conflicts with other interfaces in the same bridged network, Cumulus Networks has [reserved a range of MAC addresses](#) specifically to use with MLAG. This range of MAC addresses is 44:38:39:ff:00:00 to 44:38:39:ff:ff:ff.

Cumulus Networks recommends you use this range of MAC addresses when configuring MLAG.

## Configuring the Host or Switch

On your dual-connected device, create a bond that uses LACP. The method you use varies with the type of device you are configuring. The following image is a basic MLAG configuration, showing all the essential elements; a more detailed two-leaf/two-spine configuration is [below \(see page \)](#).



## Configuring the Interfaces

Every interface that connects to the MLAG pair from a dual-connected device should be placed into a [bond](#) ([see page 211](#)), even if the bond contains only a single link on a single physical switch (since the MLAG pair contains two or more links). Layer 2 data travels over this bond. In the examples throughout this chapter, *peerlink* is the name of the bond.

Single-attached hosts, also known as *orphan ports*, can be just a member of the bridge.

Additionally, the fast mode of LACP should be configured on the bond to allow more timely updates of the LACP state. These bonds will then be placed in a bridge, which will include the peer link between the switches.

In order to enable communication between the `clagd` services on the peer switches, you should choose an unused VLAN (also known as a *switched virtual interface* or *SVI* here) and assign an unrouteable link-local address to give the peer switches layer 3 connectivity between each other. To ensure that the VLAN is completely independent of the bridge and spanning tree forwarding decisions, configure the VLAN as a VLAN subinterface on the peer link bond rather than the VLAN-aware bridge. Cumulus Networks recommends you use 4094 for the peer link VLAN (*peerlink.4094* below) if possible. In addition, to avoid issues with STP, make sure you include untagged traffic on the peer link.

You can also specify a backup interface, which is any layer 3 backup interface for your peer links in the event that the peer link goes down. [See below \(see page 254\)](#) for more information about the backup link.

For example, if peerlink is the inter-chassis bond, and VLAN 4094 is the peer link VLAN, configure peerlink.4094 using:

```
auto peerlink.4094
iface peerlink.4094
    address 169.254.1.1/30
    clagd-peer-ip 169.254.1.2
    clagd-backup-ip 192.0.2.50
    clagd-sys-mac 44:38:39:FF:40:94
```

Then run `ifup` on the peer link VLAN interface. In this example, the command would be `sudo ifup peerlink.4094`.

There is no need to add VLAN 4094 to the bridge VLAN list, as it is unnecessary there.



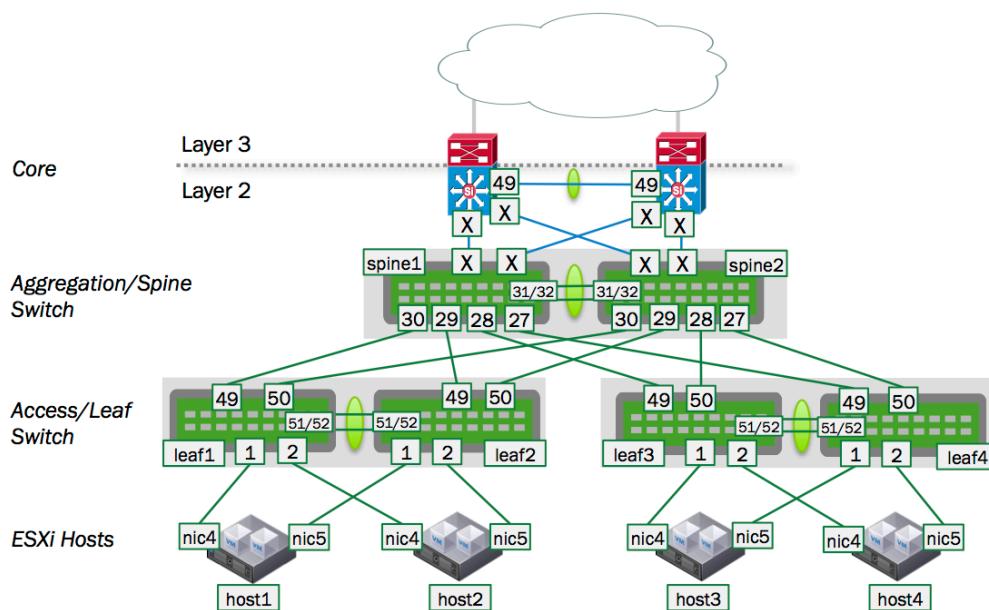
Keep in mind that when you change the MLAG configuration in the `interfaces` file, the changes take effect when you bring the peer link interface up with `ifup`. Do **not** use `systemctl restart clagd.service` to apply the new configuration.



Do not use 169.254.0.1 as the MLAG peerlink IP address, as Cumulus Linux uses this address exclusively for BGP unnumbered ([see page 425](#)) interfaces.

## Example MLAG Configuration

An example configuration is included below. It configures two bonds for MLAG, each with a single port, a peer link that is a bond with two member ports, and three VLANs on each port. You store the configuration in `/etc/network/interfaces` on each peer switch.



Configuring these interfaces uses syntax from `ifupdown2` and the [VLAN-aware bridge driver mode](#) (see page 235). The bridges use these Cumulus Linux-specific keywords:

- **bridge-vids**, which defines the allowed list of tagged 802.1q VLAN IDs for all bridge member interfaces. You can specify non-contiguous ranges with a space-separated list, like `bridge-vids 100-200 300 400-500`.
- **bridge-pvid**, which defines the untagged VLAN ID for each port. This is commonly referred to as the *native VLAN*.

The bridge configurations below indicate that each bond carries tagged frames on VLANs 1000 to 3000 but untagged frames on VLAN 1. Also, take note on how you configure the VLAN subinterface used for `clagd` communication (`peerlink.4094` in the sample configuration below).



At minimum, this VLAN subinterface should not be in your Layer 2 domain, and you should give it a very high VLAN ID (up to 4094). Read more about the [range of VLAN IDs you can use](#) (see page 243).

The configuration for the spines should look like the following (note that the `clag-id` and `clagd-sys-mac` must be the same for the corresponding bonds on spine1 and spine2):

spine1	spine2
<pre># The loopback network interface auto lo iface lo inet loopback  # The primary network interface auto eth0</pre>	<pre># The loopback network interface auto lo iface lo inet loopback  # The primary network interface auto eth0</pre>

```

iface eth0
    address 10.0.0.1
    netmask 255.255.255.0

auto peerlink
iface peerlink
    bond-slaves swp31 swp32

auto peerlink.4094
iface peerlink.4094
    address 169.254.255.1
    netmask 255.255.255.0
    clagd-priority 4096
    clagd-peer-ip 169.254.255.2
    clagd-backup-ip 10.0.0.2
    clagd-sys-mac 44:38:39:ff:00:01

# ToR pair #1
auto downlink1
iface downlink1
    bond-slaves swp29 swp30
    clag-id 1

# ToR pair #2
auto downlink2
iface downlink2
    bond-slaves swp27 swp28
    clag-id 2

auto br0
iface br0
    bridge-vlan-aware yes
    bridge-ports uplinkA
    peerlink downlink1 downlink2
    bridge-stp on
    bridge-vids 1000-2999
    bridge-pvid 1
    mstptctl-treeprio 4096

```

```

iface eth0
    address 10.0.0.2
    netmask 255.255.255.0

auto peerlink
iface peerlink
    bond-slaves swp31 swp32

auto peerlink.4094
iface peerlink.4094
    address 169.254.255.2
    netmask 255.255.255.0
    clagd-priority 8192
    clagd-peer-ip 169.254.255.1
    clagd-backup-ip 10.0.0.1
    clagd-sys-mac 44:38:39:ff:00:01

# ToR pair #1
auto downlink1
iface downlink1
    bond-slaves swp29 swp30
    clag-id 1

# ToR pair #2
auto downlink2
iface downlink2
    bond-slaves swp27 swp28
    clag-id 2

auto br0
iface br0
    bridge-vlan-aware yes
    bridge-ports uplinkA
    peerlink downlink1 downlink2
    bridge-stp on
    bridge-vids 1000-2999
    bridge-pvid 1
    mstptctl-treeprio 4096

```

Here is an example configuration file for the switches leaf1 and leaf2. Note that the `clag-id` and `clagd-sys-mac` must be the same for the corresponding bonds on leaf1 and leaf2:

leaf1

leaf2

```

# The loopback network
interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0
    address 10.0.0.3
    netmask 255.255.255.0

auto spine1-2
iface spine1-2
    bond-slaves swp49 swp50
    clag-id 1

auto peerlink
iface peerlink
    bond-slaves swp51 swp52

auto peerlink.4094
iface peerlink.4094
    address 169.254.255.3
    netmask 255.255.255.0
    clagd-priority 4096
    clagd-peer-ip 169.254.255.4
    clagd-backup-ip 10.0.0.4
    clagd-sys-mac 44:38:39:ff:
01:02

auto host1
iface host1
    bond-slaves swp1
    clag-id 2
    mstpclctl-portadminedge yes
    mstpclctl-bpduguard yes

auto host2
iface host2
    bond-slaves swp2
    clag-id 3
    mstpclctl-portadminedge yes
    mstpclctl-bpduguard yes

auto br0
iface br0
    bridge-vlan-aware yes
    bridge-ports spine1-2
    peerlink host1 host2
    bridge-stp on
  
```

```

# The loopback network
interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0
    address 10.0.0.4
    netmask 255.255.255.0

auto spine1-2
iface spine1-2
    bond-slaves swp49 swp50
    clag-id 1

auto peerlink
iface peerlink
    bond-slaves swp51 swp52

auto peerlink.4094
iface peerlink.4094
    address 169.254.255.4
    netmask 255.255.255.0
    clagd-priority 8192
    clagd-peer-ip 169.254.255.3
    clagd-backup-ip 10.0.0.3
    clagd-sys-mac 44:38:39:ff:
01:02

auto host1
iface host1
    bond-slaves swp1
    clag-id 2
    mstpclctl-portadminedge yes
    mstpclctl-bpduguard yes

auto host2
iface host2
    bond-slaves swp2
    clag-id 3
    mstpclctl-portadminedge yes
    mstpclctl-bpduguard yes

auto br0
iface br0
    bridge-vlan-aware yes
    bridge-ports spine1-2
    peerlink host1 host2
    bridge-stp on
  
```

```
bridge-vids 1000-2999
bridge-pvid 1
mstpcctl-treeprio 8192
```

```
bridge-vids 1000-2999
bridge-pvid 1
mstpcctl-treeprio 8192
```

The configuration is almost identical, except for the IP addresses used for managing the `clagd` service.



In the configurations above, the `clagd-peer-ip` and `clagd-sys-mac` parameters are mandatory, while the rest are optional. When mandatory `clagd` commands are present under a peer link subinterface, by default `clagd-enable` is set to yes and doesn't need to be specified; to disable `clagd` on the subinterface, set `clagd-enable` to no. Use `clagd-priority` to set the role of the MLAG peer switch to primary or secondary. Each peer switch in an MLAG pair must have the same `clagd-sys-mac` setting. Each `clagd-sys-mac` setting should be unique to each MLAG pair in the network. For more details refer to `man clagd`.

## Configuring MLAG with a Traditional Mode Bridge

It's possible to configure MLAG with a bridge in [traditional mode](#) (see page 214) instead of [VLAN-aware mode](#) (see page 235). In order to do so, the peer link and all dual-connected links must be configured as untagged/native (see page 224) ports on a bridge (note the absence of any VLANs in the `bridge-ports` line and the lack of the `bridge-vlan-aware` parameter below):

```
auto br0
iface br0
    bridge-ports peerlink spine1-2 host1 host2
```



For a deeper comparison of traditional versus VLAN-aware bridge modes, read this [knowledge base article](#).

## Using the `clagd` Command Line Interface

A command line utility called `clagctl` is available for interacting with a running `clagd` service to get status or alter operational behavior. For detailed explanation of the utility, please refer to the `clagctl(8)` man page. The following is a sample output of the MLAG operational status displayed by the utility:

```
cumulus@switch$ clagctl
The peer is alive
    Our Priority, ID, and Role: 8192 00:e0:ec:26:50:89 primary
    Peer Priority, ID, and Role: 8192 00:e0:ec:27:49:f6 secondary
    Peer Interface and IP: peerlink.4094 169.254.255.2
    System MAC: 44:38:39:ff:00:01
```

Dual Attached Ports

Our Interface	Peer Interface	CLAG Id
downlink1	downlink1	1
downlink2	downlink2	2

## Peer Link Interfaces and the `protodown` State

In addition to the standard UP and DOWN administrative states, an interface that is a member of an MLAG bond can also be in a `protodown` state. When MLAG detects a problem that could result in connectivity issues such as traffic black-holing or a network meltdown if the link carrier was left in an UP state, it can put that interface into `protodown` state. Such connectivity issues include:

- When the peer link goes down but the peer switch is up (that is, the backup link is active).
- When the bond is configured with an MLAG ID, but the `clagd` service is not running (whether it was deliberately stopped or simply died).
- When an MLAG-enabled node is booted or rebooted, the MLAG bonds are placed in a `protodown` state until the node establishes a connection to its peer switch, or five minutes have elapsed.

When an interface goes into a `protodown` state, it results in a local OPER DOWN (carrier down) on the interface. As of Cumulus Linux 2.5.5, the `protodown` state can be manipulated with the `ip link set` command. Given its use in preventing network meltdowns, manually manipulating `protodown` is not recommended outside the scope of interaction with the Cumulus Networks support team.

The following `ip link show` command output shows an interface in `protodown` state. Notice that the link carrier is down (NO-CARRIER):

```
cumulus@switch:~$ ip link show swp1
3: swp1: <NO-CARRIER,BROADCAST,MULTICAST,SLAVE,UP> mtu 1500 qdisc pfifo_fast master host-bond1 state DOWN mode DEFAULT qlen 500
  protodown on
    link/ether 44:38:39:00:69:84 brd ff:ff:ff:ff:ff:ff
```

## Specifying a Backup Link

You can specify a backup link for your peer links in the event that the peer link goes down. When this happens, the `clagd` service uses the backup link to check the health of the peer switch. To configure this, edit `/etc/network/interfaces` and add `clag-backup-ip <ADDRESS>` to the peer link configuration. Here's an example:

```
auto peerlink.4094
iface peerlink.4094
  address 169.254.255.1
  netmask 255.255.255.0
  clagd-priority 8192
  clagd-peer-ip 169.254.255.2
  clagd-backup-ip 192.0.2.50
  clagd-sys-mac 44:38:39:ff:00:01
  clagd-args --priority 1000
```



The backup IP address must be different than the peer link IP address (`clagd-peer-ip` above). It must be reachable by a route that doesn't use the peer link and it must be in the same network namespace as the peer link IP address.

Cumulus Networks recommends you use the switch's management IP address for this purpose.

You can also specify the backup UDP port. The port defaults to 5342, but you can configure it as an argument in `clagd-args` using `--backupPort <PORT>`.

```
auto peerlink.4094
iface peerlink.4094
    address 169.254.255.1
    netmask 255.255.255.0
    clagd-priority 8192
    clagd-peer-ip 169.254.255.2
    clagd-backup-ip 192.0.2.50
    clagd-sys-mac 44:38:39:ff:00:01
    clagd-args --backupPort 5400
```

You can see the backup IP address if you run `clagctl`:

```
cumulus@switch:~$ clagctl
The peer is alive
    Our Priority, ID, and Role: 8192 00:e0:ec:26:50:89 primary
    Peer Priority, ID, and Role: 8192 00:e0:ec:27:49:f6 secondary
        Peer Interface and IP: peerlink.4094 169.254.255.2
        Backup IP: 192.0.2.50
        System MAC: 44:38:39:ff:00:01
```

Dual Attached Ports		
Our Interface	Peer Interface	CLAG Id
downlink1	downlink1	1
downlink2	downlink2	2

## Monitoring Dual-Connected Peers

Upon receipt of a valid message from its peer, the switch knows that `clagd` is alive and executing on that peer. This causes `clagd` to change the system ID of each bond that was assigned a `clag-id` from the default value (the MAC address of the bond) to the system ID assigned to both peer switches. This makes the hosts connected to each switch act as if they are connected to the same system so that they will use all ports within their bond. Additionally, `clagd` determines which bonds are dual-connected and modifies the forwarding and learning behavior to accommodate these dual-connected bonds.

If the peer does not receive any messages for three update intervals, then that peer switch is assumed to no longer be acting as an MLAG peer. In this case, the switch reverts all configuration changes so that it operates as a standard non-MLAG switch. This includes removing all statically assigned MAC addresses, clearing the egress forwarding mask, and allowing addresses to move from any port to the peer port. Once

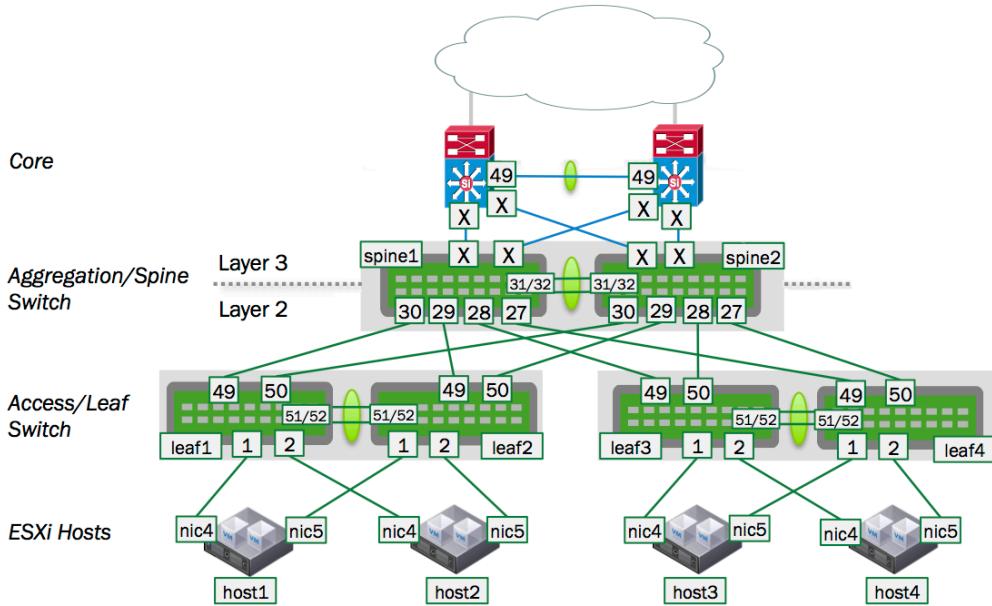
a message is again received from the peer, MLAG operation starts again as described earlier. You can configure a custom timeout setting by adding `--peerTimeout <VALUE>` to `clagd-args` in `/etc/network/interfaces`.

Once bonds are identified as dual-connected, `clagd` sends more information to the peer switch for those bonds. The MAC addresses (and VLANs) that have been dynamically learned on those ports are sent along with the LACP partner MAC address for each bond. When a switch receives MAC address information from its peer, it adds MAC address entries on the corresponding ports. As the switch learns and ages out MAC addresses, it informs the peer switch of these changes to its MAC address table so that the peer can keep its table synchronized. Periodically, at 45% of the bridge ageing time, a switch will send its entire MAC address table to the peer, so that peer switch can verify that its MAC address table is properly synchronized.

The switch sends an update frequency value in the messages to its peer, which tells `clagd` how often the peer will send these messages. You can configure a different frequency by adding `--lacpPoll <SECONDS>` to `clagd-args` in `/etc/network/interfaces`.

## Configuring Layer 3 Routed Uplinks

In this scenario, the spine switches connect at layer 3, as shown in the image below. Alternatively, the spine switches can be singly connected to each core switch at layer 3 (not shown below).



In this design, the spine switches route traffic between the server hosts in the layer 2 domains and the core. The servers (host1 - host4) each have a layer 2 connection up to the spine layer where the default gateway for the host subnets resides. However, since the spine switches act as gateway devices communicate at layer 3, you need to configure a protocol such as [VRR \(see page 264\)](#) (Virtual Router Redundancy) between the spine switch pair to support active/active forwarding.

Then, to connect the spine switches to the core switches, you need to determine whether the routing is static or dynamic. If it's dynamic, you must choose which protocol — [OSPF \(see page 406\)](#) or [BGP \(see page 419\)](#) — to use. When enabling a routing protocol in an MLAG environment it is also necessary to manage the uplinks, because by default MLAG is not aware of layer 3 uplink interfaces. In the event of a peer link failure MLAG does not remove static routes or bring down a BGP or OSPF adjacency unless a separate link state daemon such as `ifplugd` is used.

## ***IGMP Snooping with MLAG***

IGMP snooping processes IGMP reports received on a bridge port in a bridge to identify hosts that are configured to receive multicast traffic destined to that group. An IGMP query message received on a port is used to identify the port that is connected to a router and configured to receive multicast traffic.

IGMP snooping is enabled by default on the bridge. IGMP snooping multicast database entries and router port entries are synced to the peer MLAG switch. If there is no multicast router in the VLAN, the IGMP querier can be configured on the switch to generate IGMP query messages by adding a configuration like the following to `/etc/network/interfaces`:

```
auto br.100
vlan br.100
    #igmp snooping is enabled by default, but is shown here for
    completeness
    bridge-mcsnoop 1
    # If you need to specify the querier IP address
    bridge-igmp-querier-source 123.1.1.1
```

To display multicast group and router port information, use the `bridge -d mdb show` command:

```
cumulus@switch:~# sudo bridge -d mdb show
dev br port bond0 vlan 100 grp 234.1.1.1 temp
router ports on br: bond0
```

Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

```
cumulus@switch:~# sudo brctl setmcq4src br 100 192.0.2.1
cumulus@switch:~# sudo brctl setmcquerier br 1
cumulus@switch:~# sudo brctl showmcq4src br
vlan          querier address
  100           192.0.2.1
```

## ***Monitoring the Status of the clagd Service***

Due to the critical nature of the `clagd` service, `systemd` continuously monitors the status of `clagd`. `systemd` monitors the `clagd` service through the use of notify messages every 30 seconds. If the `clagd` service dies or becomes unresponsive for any reason and `systemd` receives no messages after 60 seconds, `systemd` restarts `clagd`. `systemd` logs these failures in `/var/log/syslog`, and, on the first failure, generates a `cl-support` file as well.

This monitoring is automatically configured and enabled as long as `clagd` is enabled (that is, `clagd-peer-ip` and `clagd-sys-mac` are configured in `/etc/network/interfaces`) and `clagd` been started. When `clagd` is explicitly stopped, for example with the `sudo systemctl stop clagd.service` command, monitoring of `clagd` is also stopped.

You can check the status of `clagd` monitoring by using the `cl-service-summary` command:

```
cumulus@switch:~$ sudo cl-service-summary summary
The systemctl daemon 5.4 uptime: 15m
...
Service clagd           enabled   active
...
```

## **MLAG Best Practices**

For MLAG to function properly, the dual-connected hosts' interfaces should be configured identically on the pair of peering switches. See the note above in the [Configuring MLAG \(see page 247\)](#) section.

## **Understanding MTU in an MLAG Configuration**

Note that the [MTU \(see page 152\)](#) in MLAG traffic is determined by the bridge MTU. Bridge MTU is determined by the lowest MTU setting of an interface that is a member of the bridge. If an MTU other than the default of 1500 bytes is desired, you must configure the MTU on each physical interface and bond interface that are members of the MLAG bridges in the entire bridged domain.

For example, if an MTU of 9216 is desired through the MLAG domain in the example shown above:

On the leaf switches, [configure mtu 9216 \(see page 152\)](#) for each of following interfaces, since they are members of bridge `br0`: spine1-2, peerlink, host1, host2.

```
auto br0
iface br0
  bridge-vlan-aware yes
  bridge-ports spine1-2 peerlink host1 host2    <- List of bridge
member interfaces
...
```

Likewise, to ensure the MTU 9216 path is respected through the spine switches above, also change the MTU setting for bridge `br` by configuring `mtu 9216` for each of the following members of bridge `br` on spine1 and spine2: uplinkA, peerlink, downlink1, downlink2.

```
auto br
iface br
  bridge-vlan-aware yes
  bridge-ports uplinkA peerlink downlink1 downlink2
...
```

## Sizing the Peerlink

What's the best size for a peerlink? Before we answer that, let's talk a little bit about the peerlink itself.

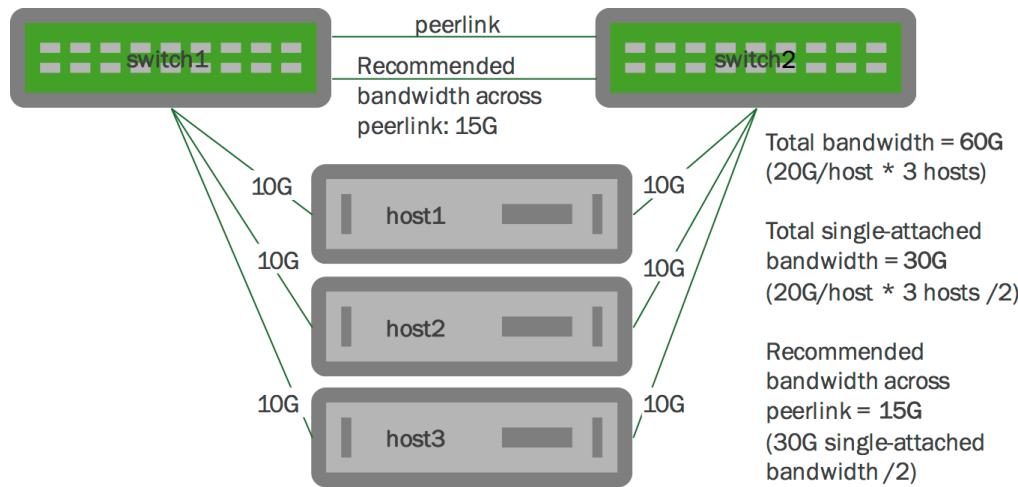
The peerlink tends to carry very little traffic when compared to the bandwidth consumed by dataplane traffic. In a typical MLAG configuration, most every connection between the two switches in the MLAG pair is dual-connected, so the only traffic going across the peerlink is traffic from the `clagd` process and some LLDP or LACP traffic. However, there are some instances where a host is connected to only one switch in the MLAG pair; these include:

- You have a hardware limitation on the host where there is only one PCIE slot, and thus, one NIC on the system, so the host is only single-connected across that interface.
- The host doesn't support 802.3ad and you can't create a bond on it.
- You are accounting for a link failure, where the host may become single connected until the failure is rectified.

So, in terms of sizing the peerlink, in general, you need to determine how much bandwidth is traveling across the single-connected interfaces, and allocate half of that bandwidth to the peerlink. We recommend half of the single-connected bandwidth because, on average, one half of the traffic destined to the single-connected host will arrive on the switch directly connected to the single-connected host and the other half will arrive on the switch that is not directly connected to the single-connected host. When this happens, only the traffic that arrives on the switch that is not directly connected to the single-connected host needs to traverse the peerlink, which is how you calculate 50% of the traffic.

In addition, you may want to add extra links to the peerlink bond to handle link failures in the peerlink bond itself.

In illustration below, each host has 2 10G links, with each 10G link going to each switch in the MLAG pair. Each host has 20G of dual-connected bandwidth, so all three hosts have a total of 60G of dual-connected bandwidth. We recommend you allocate at least 15G of bandwidth to each peerlink bond, which represents half of the single-connected bandwidth.



Scaling this example out to a full rack, when planning for link failures, you need only allocate enough bandwidth to meet your site's strategy for handling failure scenarios. Imagine a full rack with 40 servers and two switches in it. You may plan for, say, 4 to 6 servers to lose connectivity to a single switch and become single connected before you respond to the event. So expanding upon our previous example, if you have 40 hosts each with 20G of bandwidth dual-connected to the MLAG pair, you might allocate 20G to 30G of bandwidth to the peerlink — which accounts for half of the single-connected bandwidth for 4 to 6 hosts.

## STP Interoperability with MLAG

Cumulus Networks recommends that you always enable STP in your layer 2 network.

Further, with MLAG, Cumulus Networks recommends you enable BPDU guard on the host-facing bond interfaces. (For more information about BPDU guard, see [BPDU Guard and Bridge Assurance \(see page 177\)](#).)

## Debugging STP with MLAG

/var/log/daemon.log has mstpd logs.

Run `mstpctl debuglevel 3` to see MLAG-related logs in /var/log/daemon.log:

```
cumulus@switch:~$ sudo mstpctl showportdetail br peer-bond
br:peer-bond CIST info
  enabled          yes           role
Designated
  port id        8.008         state
forwarding
  .....
  bpdufilter port no
  clag ISL        yes          clag ISL Oper UP      yes
  clag role       primary      clag dual conn mac  0:0:
0:0:0:0
  clag remote portID F.FFF      clag system mac    44:3
8:39:ff:0:1
cumulus@switch:~$

cumulus@switch:~$ sudo mstpctl showportdetail br downlink-1
br:downlink-1 CIST info
  enabled          yes           role
Designated
  port id        8.006         state
forwarding
  .....
  bpdufilter port no
  clag ISL        no           clag ISL Oper UP      no
  clag role       primary      clag dual conn mac  0:0:
0:3:11:1
  clag remote portID F.FFF      clag system mac    44:3
8:39:ff:0:1
cumulus@switch:~$
```

## Best Practices for STP with MLAG

- The STP global configuration must be the same on both the switches.
- The STP configuration for dual-connected ports should be the same on both peer switches.

- Use `mstpctl` commands for all spanning tree configurations, including bridge priority, path cost and so forth. Do not use `brctl` commands for spanning tree, except for `brctl stp on/off`, as changes are not reflected to `mstpd` and can create conflicts.

## Troubleshooting MLAG

By default, when `clagd` is running, it logs its status to the `/var/log/clagd.log` file and syslog. Example log file output is below:

```
Jan 14 23:45:10 switch clagd[3704]: Beginning execution of clagd
version 1.0.0
Jan 14 23:45:10 switch clagd[3704]: Invoked with: /usr/sbin/clagd --
daemon 169.254.2.2 peer-bond.4000 44:38:39:ff:00:01 --priority 8192
Jan 14 23:45:11 switch clagd[3995]: Role is now secondary
Jan 14 23:45:31 switch clagd[3995]: Role is now primary
Jan 14 23:45:32 switch clagd[3995]: The peer switch is active.
Jan 14 23:45:35 switch clagd[3995]: downlink-1 is now dual connected.
```

## Caveats and Errata

If both the backup and peer connectivity are lost within a 30-second window, the switch in the secondary role misinterprets the event sequence, believing the peer switch is down, so it takes over as the primary.

## Configuration Files

- `/etc/network/interfaces`

## LACP Bypass

On Cumulus Linux, *LACP Bypass* is a feature that allows a **bond** (see page 211) configured in 802.3ad mode to become active and forward traffic even when there is no LACP partner. A typical use case for this feature is to enable a host, without the capability to run LACP, to PXE boot while connected to a switch on a bond configured in 802.3ad mode. Once the pre-boot process finishes and the host is capable of running LACP, the normal 802.3ad link aggregation operation takes over.

## Contents

(Click to expand)

- [Contents \(see page 261\)](#)
- [Understanding the LACP Bypass All-active Mode \(see page 261\)](#)
  - [LACP Bypass and MLAG Deployments \(see page 262\)](#)
- [Configuring LACP Bypass \(see page 262\)](#)
  - [VLAN-aware Bridge Mode Configuration \(see page 262\)](#)
  - [Traditional Bridge Mode Configuration \(see page 263\)](#)

## ***Understanding the LACP Bypass All-active Mode***

When a bond has multiple slave interfaces, each bond slave interface operates as an active link while the bond is in bypass mode. This is known as *all-active mode*. This is useful during PXE boot of a server with multiple NICs, when the user cannot determine beforehand which port needs to be active.

Keep in the mind the following caveats with all-active mode:

- All-active mode is not supported on bonds that are not specified as bridge ports on the switch.
- Spanning tree protocol (STP) does not run on the individual bond slave interfaces when the LACP bond is in all-active mode. Therefore, only use all-active mode on host-facing LACP bonds. Cumulus Networks highly recommends you configure [STP BPDU guard \(see page 177\)](#) along with all-active mode.



As of Cumulus Linux 3.0.0, priority mode, `bond-lacp-bypass-period`, `bond-lap-bypass-priority` and `bond-lap-bypass-all-active` are no longer supported.

## ***LACP Bypass and MLAG Deployments***

In an [MLAG deployment \(see page 244\)](#) where bond slaves of a host are connected to two switches and the bond is in all-active mode, all the slaves of bond are active on both the primary and secondary MLAG nodes.

## ***Configuring LACP Bypass***

You configure LACP bypass in the `/etc/network/interfaces` file.

To enable LACP bypass on the host-facing bond, under the bond interface stanza, set `bond-lacp-bypass-allow` to 1.

## ***VLAN-aware Bridge Mode Configuration***

The following configuration shows LACP bypass enabled for a bridge in [VLAN-aware mode \(see page 235\)](#):

```
auto bond1
iface bond1
    bond-slaves swp51s2 swp51s3
    clag-id 1
    bond-lacp-bypass-allow 1

...
auto br0
iface br0
    bridge-vlan-aware yes
    mstpctl-bpduguard bond1=yes
```

```
bridge_ports bond1 bond2 bond3 bond4 peer5
bridge-stp on
bridge-vids 100-105
```

You can check the status of the configuration by running `ip link show` on the bond and its slave interfaces:

```
cumulus@switch:~$ ip link show bond1
164: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue
    master br0 state UP mode DORMANT group default
        link/ether c4:54:44:f6:44:5a brd ff:ff:ff:ff:ff:ff
cumulus@switch:~$ ip link show swp51s2
55: swp51s2: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond1 state UP mode DEFAULT group default qlen 1000
        link/ether c4:54:44:f6:44:5a brd ff:ff:ff:ff:ff:ff
cumulus@switch:~$ ip link show swp52s3
56: swp51s3: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    master bond1 state UP mode DEFAULT group default qlen 1000
        link/ether c4:54:44:f6:44:5a brd ff:ff:ff:ff:ff:ff
```

Use the `cat` command to verify that LACP bypass is enabled on a bond and its slave interfaces:

```
cumulus@switch:~$ cat /sys/class/net/bond1/bonding/lacp_bypass
on 1
cumulus@switch:~$ cat /sys/class/net/bond1/bonding/slaves
swp51s2 swp51s3
cumulus@switch:~$ cat /sys/class/net/swp51s2/bonding_slave/ad_rx_bypass
1
cumulus@switch:~$ cat /sys/class/net/swp51s3/bonding_slave/ad_rx_bypass
1
```

## **Traditional Bridge Mode Configuration**

The following configuration shows LACP bypass enabled for multiple active interfaces (all-active mode) with a bridge in traditional bridge mode (see page 214):

```
auto bond1
iface bond1
    bond-slaves swp3 swp4
    bond-lacp-bypass-allow 1
```

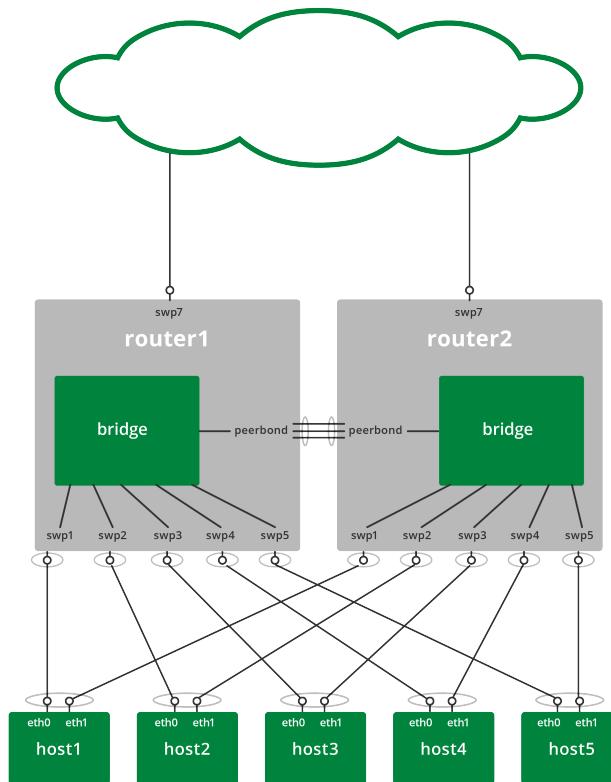
```
auto br0
iface br0
  bridge-ports bond1 bond2 bond3 bond4 peer5
  bridge-stp on
  mstpctl-bpduguard bond1=yes
```

## Virtual Router Redundancy - VRR

VRR provides virtualized router redundancy in network configurations, which enables the hosts to communicate with any redundant router without:

- Needing to be reconfigured
- Having to run dynamic router protocols
- Having to run router redundancy protocols

A basic VRR-enabled network configuration is shown below. The network consists of several hosts, two routers running Cumulus Linux and configured with [MLAG](#) (see page 244), and the rest of the network:



An actual implementation will have many more server hosts and network connections than are shown here. But this basic configuration provides a complete description of the important aspects of the VRR setup.

## Contents

(Click to expand)

- [Contents \(see page 264\)](#)
- [Configuring the Network \(see page 265\)](#)
  - [Reserved MAC Address Range \(see page 266\)](#)
  - [Configuring the Hosts \(see page 266\)](#)
  - [Configuring the Routers \(see page 266\)](#)
  - [Other Network Connections \(see page 267\)](#)
  - [Handling ARP Requests \(see page 267\)](#)
  - [Monitoring Peer Links and Uplinks \(see page 267\)](#)
- [Using ifplugd \(see page 267\)](#)
- [Notes \(see page 269\)](#)

## Configuring the Network

Configuring this network is fairly straightforward. First create the bridge subinterface, then create the secondary address for the virtual router. Configure each router with a bridge; edit each router's `/etc/network/interfaces` file and add a configuration similar to the following:

```
auto bridge.500
iface bridge.500
    address 192.168.0.252/24
    address-virtual 00:00:5e:00:01:01 192.168.0.254/24
```



Notice the simpler configuration of the bridge with `ifupdown2`. For more information, see [Configuring and Managing Network Interfaces \(see page 134\)](#).

You should always use `ifupdown2` to configure VRR, because it ensures correct ordering when bringing up the virtual and physical interfaces and it works best with [VLAN-aware bridges \(see page 235\)](#).

If you are using the `traditional mode` bridge driver, the configuration would look like this:

```
auto bridge500
iface bridge500
    address 192.168.0.252/24
    address-virtual 00:00:5e:00:01:01 192.168.0.254/24
    bridge-ports bond1.500 bond2.500 bond3.500
```

The IP address assigned to the bridge is the unique address for the bridge. The parameters of this configuration are:

- **`bridge.500`**: 500 represents a VLAN subinterface of the bridge, sometimes called a switched virtual interface, or SVI.
- **`192.168.0.252/24`**: The unique IP address assigned to this bridge. It is unique because, unlike the 192.168.0.254 address, it is assigned only to this bridge, not the bridge on the other router.
- **`00:00:5e:00:01:01`**: The MAC address of the virtual router. This must be the same on all virtual routers. Cumulus Linux has a reserved range for VRR MAC addresses. See below for details.
- **`192.168.0.254/24`**: The IP address of the virtual router, including the routing prefix. This must be the same on all the virtual routers and must match the default gateway address configured on the servers as well as the size of the subnet.
- **`address-virtual`**: This keyword enables and configures VRR.

The above bridge configuration enables VRR by creating a *MAC VLAN interface* on the SVI. This MAC VLAN interface is:

- Named bridge-500-v0, which is the name of the SVI with dots changed to dashes and "-v0" appended to the end.
- Assigned a MAC address of `00:00:5e:00:01:01`.
- Assigned an IP address of `192.168.0.254`.

## Reserved MAC Address Range

In order to prevent MAC address conflicts with other interfaces in the same bridged network, Cumulus Networks has [reserved a range of MAC addresses](#) specifically to use with VRR. This range of MAC addresses is 00:00:5E:00:01:00 to 00:00:5E:00:01:ff.

You may notice that this is the same range reserved for VRRP, since VRR serves a similar function. Cumulus Networks recommends you use this range of MAC addresses when configuring VRR.

## Configuring the Hosts

Each host should have two network interfaces. The routers configure the interfaces as bonds running LACP; the hosts should also configure its two interfaces using teaming, port aggregation, port group, or EtherChannel running LACP. Configure the hosts, either statically or via DHCP, with a gateway address that is the IP address of the virtual router; this default gateway address never changes.

Configure the links between the hosts and the routers in *active-active* mode for First Hop Redundancy Protocol.



If you are configuring VRR without [MLAG](#) (see page 244), use *active-standby* mode instead. For example, configure the bond like this:

```
auto bond0
iface bond0
    bond-mode active-backup
```

The configuration may vary, depending upon the host OS.

## Configuring the Routers

The routers implement the layer 2 network interconnecting the hosts, as well as the redundant routers. If you are using [MLAG](#) (see page 244), configure each router with a bridge interface, named *bridge* in our example, with these different types of interfaces:

- One bond interface to each host (swp1-swp5 in the image above).
- One or more interfaces to each peer router (peerbond in the image above). Multiple inter-peer links are typically bonded interfaces in order to accommodate higher bandwidth between the routers and to offer link redundancy.



If you are not using MLAG, then the bridge should have one switch port interface to each host instead of a bond.

## Other Network Connections

Other interfaces on the router can connect to other subnets and are accessed through layer 3 forwarding (swp7 in the image above).

## Handling ARP Requests

The entire purpose of this configuration is to have all the redundant routers respond to ARP requests from hosts for the virtual router IP address (192.168.0.254 in the example above) with the virtual router MAC address (00:00:5e:00:01:01 in the example above). All of the routers should respond in an identical manner, but if one router fails, the other redundant routers will continue to respond in an identical manner, leaving the hosts with the impression that nothing has changed.

Since the bridges in each of the redundant routers are connected, they will each receive and reply to ARP requests for the virtual router IP address. Each ARP request made by a host will receive multiple replies (typically two). But these replies will be identical and so the host that receives these replies will not get confused over which response is "correct" and will either ignore replies after the first, or accept them and overwrite the previous reply with identical information.

## Monitoring Peer Links and Uplinks

When an uplink on a switch in active-active mode goes down, the peer link may get congested. When this occurs, you should monitor the uplink and shut down all host-facing ports using `ifplugd` (or another script).

When the peer link goes down in a MLAG environment, one of the switches becomes secondary and all host-facing dual-connected bonds go down. The host side bond sees two different system MAC addresses, so the link to primary is active on host. If any traffic from outside this environment goes to the secondary MLAG switch, traffic will be black-holed. To avoid this, shut down all the uplinks when the peer link goes down using `ifplugd`.

## Using `ifplugd`

`ifplugd` is a link state monitoring daemon that can execute user-specified scripts on link transitions (not admin-triggered transitions, but transitions when a cable is plugged in or removed).

Run the following commands to install the `ifplugd` service:

```
cumulus@switch:$ sudo apt-get update
cumulus@switch:$ sudo apt-get install ifplugd
```

Next, configure `ifplugd`. The example below indicates that when the peerbond goes down in a MLAG environment, `ifplugd` brings down all the uplinks. Run the following `ifplugd` script on both the primary and secondary MLAG (see page 244) switches.

To configure `ifplugd`, modify `/etc/default/ifplugd` and add the appropriate peerbond interface name. `/etc/default/ifplugd` will look like this:

```
INTERFACES="peerbond"
HOTPLUG_INTERFACES=""
ARGS="-q -f -u0 -d1 -w -I"
SUSPEND_ACTION="stop"
```

Next, modify the `/etc/ifplugd/action.d/ifupdown` script.

```
#!/bin/sh
set -e
case "$2" in
up)
    clagrole=$(clagctl | grep "Our Priority" | awk '{print $8}')
    if [ "$clagrole" = "secondary" ]
    then
        #List all the interfaces below to bring up when clag
peerbond comes up.
        for interface in swp1 bond1 bond3 bond4
        do
            echo "bringing up : $interface"
            ip link set $interface up
        done
    fi
;;
down)
    clagrole=$(clagctl | grep "Our Priority" | awk '{print $8}')
    if [ "$clagrole" = "secondary" ]
    then
        #List all the interfaces below to bring down when clag
peerbond goes down.
        for interface in swp1 bond1 bond3 bond4
        do
            echo "bringing down : $interface"
            ip link set $interface down
        done
    fi
;;
esac
```

```
        done
    fi
;;
esac
```

Finally, restart `ifplugged` for your changes to take effect:

```
cumulus@switch:$ sudo systemctl restart ifplugged.service
```

## Notes

- The default shell is `/bin/sh`, which is `dash` and not `bash`. This makes for faster execution of the script since `dash` is small and quick, but consequently less featureful than `bash`. For example, it doesn't handle multiple uplinks.

## Network Virtualization

Cumulus Linux supports these forms of [network virtualization](#):

VXLAN (Virtual Extensible LAN), is a standard overlay protocol that abstracts logical virtual networks from the physical network underneath. You can deploy simple and scalable layer 3 Clos architectures while extending layer 2 segments over that layer 3 network.

VXLAN uses a VLAN-like encapsulation technique to encapsulate MAC-based layer 2 Ethernet frames within layer 3 UDP packets. Each virtual network is a VXLAN logical L2 segment. VXLAN scales to 16 million segments – a 24-bit VXLAN network identifier (VNI ID) in the VXLAN header – for multi-tenancy.

Hosts on a given virtual network are joined together through an overlay protocol that initiates and terminates tunnels at the edge of the multi-tenant network, typically the hypervisor vSwitch or top of rack. These edge points are the VXLAN tunnel end points (VTEP).

Cumulus Linux can initiate and terminate VTEPs in hardware and supports wire-rate VXLAN with Tomahawk, Trident II+ and Trident II platforms. VXLAN provides an efficient hashing scheme across IP fabric during the encapsulation process; the source UDP port is unique, with the hash based on L2-L4 information from the original frame. The UDP destination port is the standard port 4789.

Cumulus Linux includes the native Linux VXLAN kernel support and integrates with controller-based overlay solutions like VMware NSX and Midokura MidoNet.

VXLAN is supported only on switches in the [Cumulus Linux HCL](#) using Tomahawk, Trident II+ and Trident II chipsets.



VXLAN encapsulation over layer 3 subinterfaces is not supported. Therefore, VXLAN uplinks should be only configured as layer 3 interfaces without any subinterfaces.

Furthermore the VXLAN tunnel endpoints cannot share a common subnet; there must be at least one layer 3 hop between the VXLAN source and destination.

## Commands

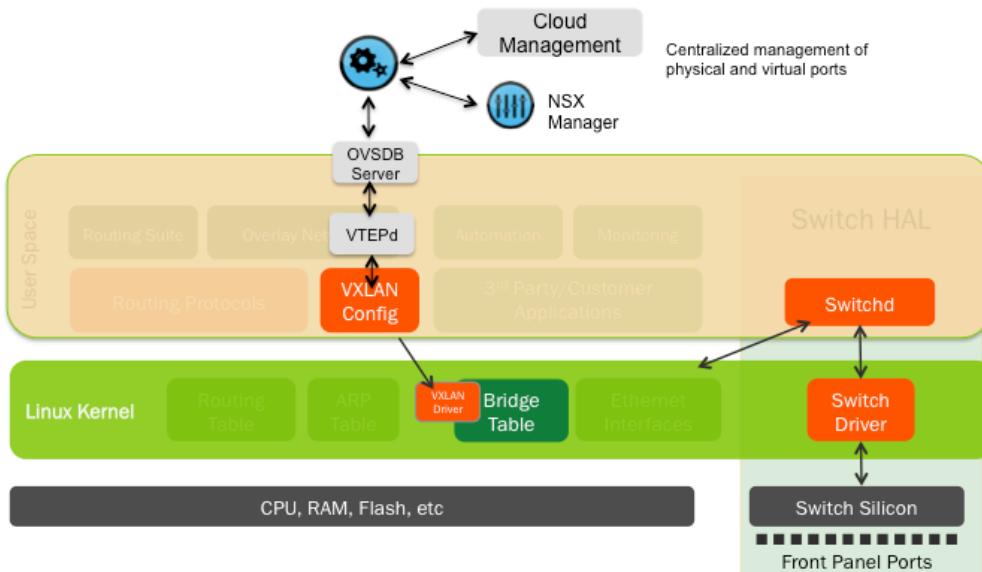
- brctl
- bridge fdb
- ip link
- ovs-pki
- ovsdb-client
- vtep-ctl

## Useful Links

- VXLAN IETF draft
- ovsdb-server

## Integrating with VMware NSX

Switches running Cumulus Linux can integrate with VMware NSX to act as VTEP gateways. The VMware NSX controller provides consistent provisioning across virtual and physical server infrastructures.



## Contents

(Click to expand)

- Contents (see page 270)
- Getting Started (see page 271)
  - Caveats and Errata (see page 271)
- Bootstrapping the NSX Integration (see page 271)
  - Enabling the openvswitch-vtep Package (see page 271)

- Using the Bootstrapping Script (see page 272)
- Manually Bootstrapping the NSX Integration (see page 273)
- Generating the Credentials Certificate (see page 273)
- Configuring the Switch as a VTEP Gateway (see page 275)
- Configuring the Transport Layer (see page 278)
- Configuring the Logical Layer (see page 279)
  - Defining Logical Switches (see page 279)
  - Defining Logical Switch Ports (see page 281)
- Verifying the VXLAN Configuration (see page 283)
- Troubleshooting VXLANs in NSX (see page 284)

## Getting Started

Before you integrate VXLANs with NSX, make sure you have the following components:

- A switch (L2 gateway) with a Tomahawk, Trident II+ or Trident II chipset running Cumulus Linux 2.0 or later;
- OVSDB server (ovsdb-server), included in Cumulus Linux 2.0 and later
- VTEPd (ovs-vtep), included in Cumulus Linux 2.0 and later

Integrating a VXLAN with NSX involves:

- Bootstrapping the NSX Integration
- Configuring the Transport Layer
- Configuring the Logical Layer
- Verifying the VXLAN Configuration

## Caveats and Errata

- As mentioned in [Network Virtualization](#) (see page 269), the switches with the source and destination VTEPs cannot reside on the same subnet; there must be at least one layer 3 hop between the VXLAN source and destination.
- There is no support for VXLAN routing in the Tomahawk, Trident II+ and Trident II chips; use a loopback interface or external router.
- Do not use 0 or 16777215 as the VNI ID, as they are reserved values under Cumulus Linux.
- For more information about NSX, see the VMware NSX User Guide, version 4.0.0 or later.

## Bootstrapping the NSX Integration

Before you start configuring the gateway service and logical switches and ports that comprise the VXLAN, you need to complete some steps to bootstrap the process. You need to do the bootstrapping just once, before you begin the integration.

## Enabling the `openvswitch-vtep` Package

Before you start bootstrapping the integration, you need to enable the `openvswitch-vtep` package, as it is disabled by default in Cumulus Linux.

1. In `/etc/default/openvswitch-vtep`, change the `START` option from `no` to `yes`:

```
cumulus@switch$ cat /etc/default/openvswitch-vtep
# This is a POSIX shell fragment                                -*- sh -*-

# Start openvswitch at boot ? yes/no
START=yes

# FORCE_COREFILES: If 'yes' then core files will be enabled.
# FORCE_COREFILES=yes

# BRCOMPAT: If 'yes' and the openvswitch-brcompat package is
installed, then
# Linux bridge compatibility will be enabled.
# BRCOMPAT=no
```

2. Start the daemon:

```
cumulus@switch$ sudo systemctl start openvswitch-vtep.service
```

## ***Using the Bootstrapping Script***

A script is available so you can do the bootstrapping automatically. For information, read `man vtep-bootstrap`. The output of the script is displayed here:

```
cumulus@vtep7: ~
cumulus@vtep7$ sudo vtep-bootstrap --credentials-path /var/lib/openvswitch vtep7 192.168.100.17 172.1
6.20.157 192.168.100.157
Executed:
    create certificate on a switch, to be used for authentication with controller
    ().
Executed:
    sign certificate
    (vtep7-req.pem      Fri Jan 17 18:04:33 UTC 2014
     fingerprint 6f443eb8445317b545d8564c2ae9638ea0a9184a).
Executed:
    define physical switch
    ().
Executed:
    define NSX controller IP address in OVSDB
    ().
Executed:
    define local tunnel IP address on the switch
    ().
Executed:
    define management IP address on the switch
    ().
Executed:
    restart a service
    (Killing ovs-vtep7d (4973).
Killing ovsdb-server (4969).
Starting ovsdb-server.
Starting ovs-vtep7d.).
cumulus@vtep7$ 
cumulus@vtep7$ ps xa | grep ovsdb-server
5184 pts/0    SK    0:00 ovsdb-server: monitoring pid 5185 (healthy)

5185 ?      S<    0:00 ovsdb-server /var/lib/openvswitch/conf.db -vANY:CONSOLE:EMER -vANY:SYSLOG:
ERR -vANY:FILE:INFO --remote=punix:/var/run/openvswitch/db.sock --remote=db:Global_managers --remote=
ptcp:6633: --private-key=/var/lib/openvswitch/vtep7-privkey.pem --certificate=/var/lib/openvswitch/vt
ep7-cert.pem --bootstrap-ca-cert=/var/lib/openvswitch/controller.cacert --no-chdir --log-file=/var/lo
g/openvswitch/ovsdb-server.log --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach --monitor
5436 pts/0    S+    0:00 grep ovsdb-server
cumulus@vtep7$
```

In the above example, the following information was passed to the `vtep-bootstrap` script:

- `--credentials-path /var/lib/openvswitch`: Is the path to where the certificate and key pairs for authenticating with the NSX controller are stored.
- `vtep7`: is the ID for the VTEP.
- `192.168.100.17`: is the IP address of the NSX controller.
- `172.16.20.157`: is the datapath IP address of the VTEP.
- `192.168.100.157`: is the IP address of the management interface on the switch.

These IP addresses will be used throughout the rest of the examples below.

## ***Manually Bootstrapping the NSX Integration***

If you don't use the script, then you must:

- Initialize the OVS database instance
- Generate a certificate and key pair for authentication by NSX
- Configure a switch as a VTEP gateway

These steps are described next.

## Generating the Credentials Certificate

First, in Cumulus Linux, you must generate a certificate that the NSX controller uses for authentication.

1. In a terminal session connected to the switch, run the following commands:

```
cumulus@switch:~$ sudo ovs-pki init
Creating controllerca...
Creating switchca...
cumulus@switch:~$ sudo ovs-pki req+sign cumulus

cumulus-req.pem Wed Oct 23 05:32:49 UTC 2013
    fingerprint b587c9fe36f09fb371750ab50c430485d33a174a
cumulus@switch:~$
cumulus@switch:~$ ls -l
total 12
-rw-r--r-- 1 root root 4028 Oct 23 05:32 cumulus-cert.pem
-rw----- 1 root root 1679 Oct 23 05:32 cumulus-privkey.pem
-rw-r--r-- 1 root root 3585 Oct 23 05:32 cumulus-req.pem
```

2. In `/usr/share/openvswitch/scripts/ovs-ctl-vtep`, make sure the lines containing **private-key**, **certificate** and **bootstrap-ca-cert** point to the correct files; **bootstrap-ca-cert** is obtained dynamically the first time the switch talks to the controller:

```
# Start ovsdb-server.
set ovsdb-server "$DB_FILE"
set "$@" -vANY:CONSOLE:EMER -vANY:SYSLOG:ERR -vANY:FILE:INFO
set "$@" --remote=unix:"$DB_SOCK"
set "$@" --remote=db:Global,managers
set "$@" --remote=ptcp:6633:$LOCALIP
set "$@" --private-key=/root/cumulus-privkey.pem
set "$@" --certificate=/root/cumulus-cert.pem
set "$@" --bootstrap-ca-cert=/root/controller.cacert
```

If files have been moved or regenerated, restart the OVSDB server and `vtep`:

```
cumulus@switch:~$ sudo systemctl restart openvswitch-vtep.service
```

3. Define the NSX controller cluster IP address in OVSDB. This causes the OVSDB server to start contacting the NSX controller:

```
cumulus@switch:~$ sudo vtep-ctl set-manager ssl:192.168.100.17:6632
```

4. Define the local IP address on the VTEP for VXLAN tunnel termination. First, find the physical switch name as recorded in OVSDB:

```
cumulus@switch:~$ sudo vtep-ctl list-ps  
vtep7
```

Then set the tunnel source IP address of the VTEP. This is the datapath address of the VTEP, which is typically an address on a loopback interface on the switch that is reachable from the underlying L3 network:

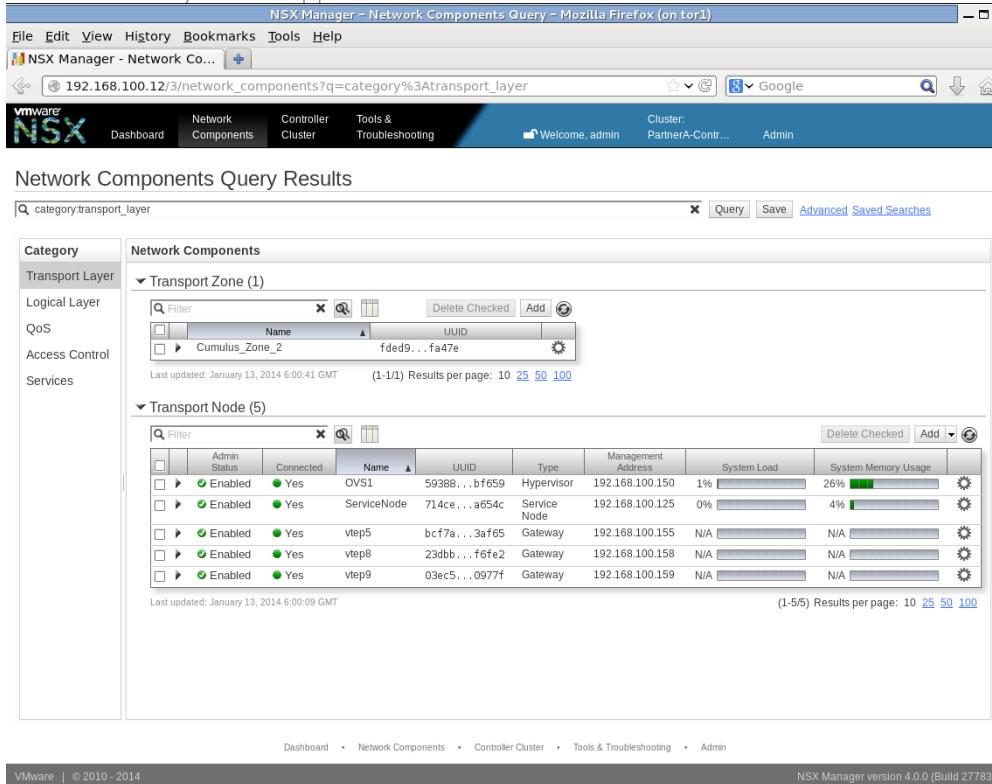
```
cumulus@switch:~$ sudo vtep-ctl set Physical_Switch vtep7  
tunnel_ip=172.16.20.157
```

Once you finish generating the certificate, keep the terminal session active, as you need to paste the certificate into NSX Manager when you configure the VTEP gateway.

## ***Configuring the Switch as a VTEP Gateway***

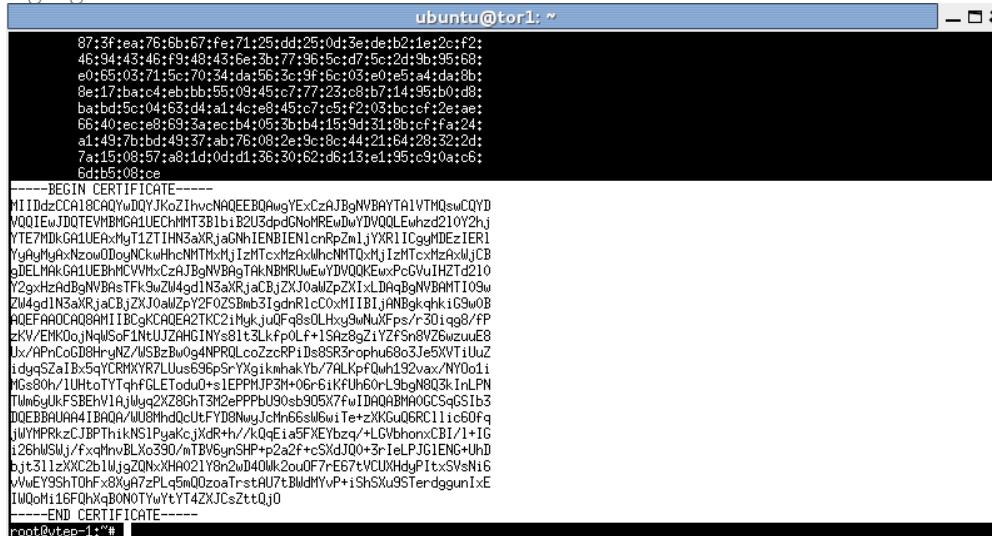
After you create a certificate, connect to NSX Manager in a browser to configure a Cumulus Linux switch as a VTEP gateway. In this example, the IP address of the NSX manager is 192.168.100.12.

- In NSX Manager, add a new gateway. Click the **Network Components** tab, then the **Transport Layer** category. Under **Transport Node**, click **Add**, then select **Manually Enter All Fields**. The Create Gateway wizard appears.



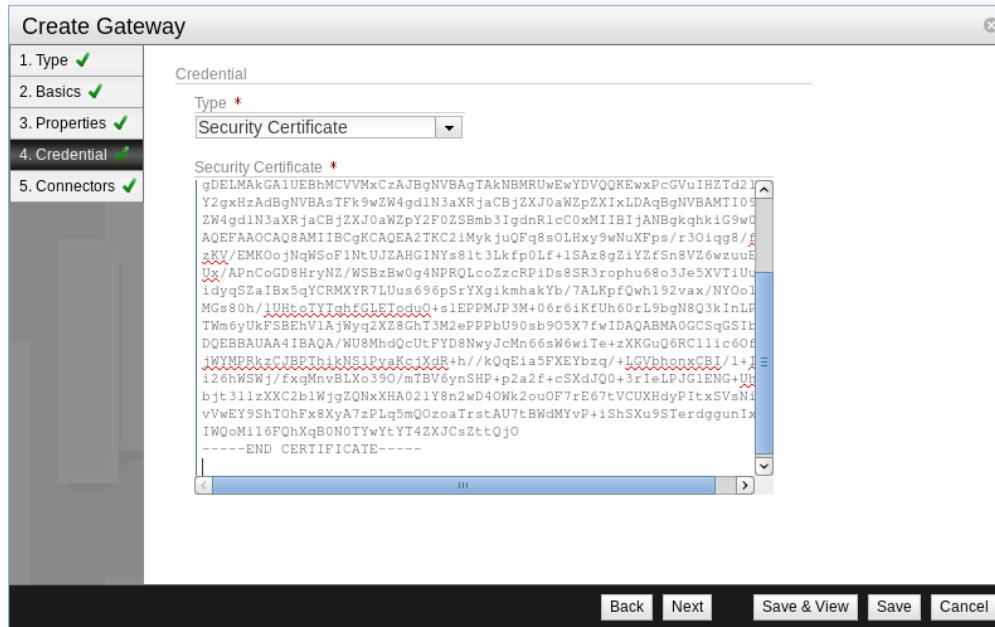
The screenshot shows the NSX Manager interface for Network Components Query. The top navigation bar includes File, Edit, View, History, Bookmarks, Tools, and Help. Below the bar, there's a search bar with the URL 192.168.100.12/3/network\_components?q=category%3Atransport\_layer. The main content area has tabs for Dashboard, Network Components, Controller Cluster, Tools & Troubleshooting, and Admin. The Network Components tab is selected. Under the Network Components tab, the Transport Layer category is selected. The Transport Zone table shows one entry: Cumulus\_Zone\_2 (fded9...fa47e). The Transport Node table shows five entries: OVS1 (5938e...bf659), ServiceNode (714ce...a654c), vtep5 (bcf7a...3af65), vtep8 (23dbb...f1fe2), and vtep9 (03ec5...0977f). Both tables have filters, delete checked, and add buttons. A status bar at the bottom indicates the interface version is NSX Manager version 4.0.0 (Build 27783).

- In the Create Gateway dialog, select **Gateway** for the **Transport Node Type**, then click **Next**.
- In the **Display Name** field, give the gateway a name, then click **Next**.
- Enable the VTEP service. Select the **VTEP Enabled** checkbox, then click **Next**.
- From the terminal session connected to the switch where you generated the certificate, copy the certificate and paste it into the **Security Certificate** text field. Copy only the bottom portion, including the **BEGIN CERTIFICATE** and **END CERTIFICATE** lines. For example, copy all the highlighted text in the terminal:



```
ubuntu@tor1: ~
87:3f:ea;76:8b:67:fe:71:25:dd:25:0d:3e:de:b2:1e:2c:f2;
46:94:43:46:f9:48:43:6e:3b:77:96:5e:d7:5c:2d:9b:95:68;
e0:65:03:71:5c:70:34:da:56:3c:9f:6e:03:e0:f4:da:8b;
8e:17:ba:c4:eb:bb:59:09:45:c7:77:23:c8:b7:14:95:b0:d8;
ba:bd:5c:04:63:41:a1:c4:e8:45:c7:c5:f2:03:bc:c7:2e:ae;
66:40:ec:e8:69:3a:ec:b4:05:3b:b4:15:9d:31:b8:cfc:faf:24;
a1:49:7b:bd:49:37:ab:76:08:2e:9c:8c:44:21:64:28:32:2d;
7a:19:08:57:a8:1d:0d:0d:36:30:02:db:13:e1:95:c9:0:a:c6;
6d:b5:08:ce
-----BEGIN CERTIFICATE-----
MIIBdCCA18CAQYDQJKzQIwvNA0QEEB0AwgYExCzATBgNVBAYTA1VTM0swCQD
001EwJQTEVMBIGA1UECHMT3B1iB2U3dpdGNoMRExDwYDVQQLEuhzd210Y2hj
YTE7MDkgA1UEAxMjT2TlHN3aXRjaGhNIEBEN1cnRpZmlyXR11CsgyIDEzER1
YgAuMyAxNzowOBYNCkuUhNMTHxkjIzTcxMzAxJhJhMTQ-MjIzMtCxJxzAxJjCB
gDELMAgCA1UEBhMCVWhCcxBJBgNVBAsTkhkNBMRUwEYDVQKExwPcGVwH2TD210
Y2gxHzAdBgNVBAsTFKw9uZl44gdIN3aXpJaC8ZXJ0alZpZXIxLxDqBgwVBANT109w
Zl44gdIN3aXpJaC8ZXJ0alZpZ2Rmb31gdR1cOwM1B1JAnBgkqhkiGw0B
9QEFAAOCAQ8M1IBCaKCAQEA2TKC2iMyjQFq8s0LHxy3wNuXfps/r30iq98/FP
zKV/EMK0oJNdsf1htUJZAHGINy81t3LkfplF+1Sa28g21YzfSh8VZ6zuuE8
Jx/APnCoGdBhNZ/USBzBw0g4NPnQlco2zcRP1Ds5R3rphu683jeXVtIUuZ
jdqgS2a1Bx5qYCRMXYR7LUsos96p5YXgikmhakYb/7HLkpQ0uh192vax/YNy0li
MG80h1UHtoTYTghGLETodu0+sLEPPNJP3M+06+61kFuhs6L9bgwB3kInLPN
Tlw6UkFSEhV1AjJyqg2XZ8GhT3M2ePPBbJ9Osbs05X7fuID9QABMA0GGSgGS1b3
DOEBBaU4A1BaQa/WU8MhdclcUtFYD8WwJcm6sdlWjTe+zXGw06RC11ic60fq
jUyYMPRkzCJBPThiKNSIPyaKcjUxRdr//hKQeia5FXYb2q/+LGvbhnxCB1/1+1G
i26hWsuJ/fxmnvBLko390/wTBw6unSP+2a2f+c5xdJ00+3r1eLPJG1ENG+UnD
bjc311zXc2b1wJg2UNxHA021Y8w2u040lk2uoF7rB7t+VCUxHdyItxvSvN16
vUyEY95hTOHfx3uA7zPLq5m02ozaTrstAU7tBldMyvP+ShSXu95STerDggunIxE
lWQoM16FGhkaBONOTyYtYT4ZJC0s2ztQj0
-----END CERTIFICATE-----
root@vtep-1:~#
```

And paste it into NSX Manager:



Then click **Next**.

6. In the Connectors dialog, click **Add Connector** to add a transport connector. This defines the tunnel endpoint that terminates the VXLAN tunnel and connects NSX to the physical gateway. You must choose a tunnel **Transport Type** of **VXLAN**. Choose an existing transport zone for the connector, or click **Create** to create a new transport zone.
7. Define the connector's IP address (that is, the underlay IP address on the switch for tunnel termination).
8. Click **OK** to save the connector, then click **Save** to save the gateway.

Once communication is established between the switch and the controller, a `controller.cacert` file will be downloaded onto the switch.

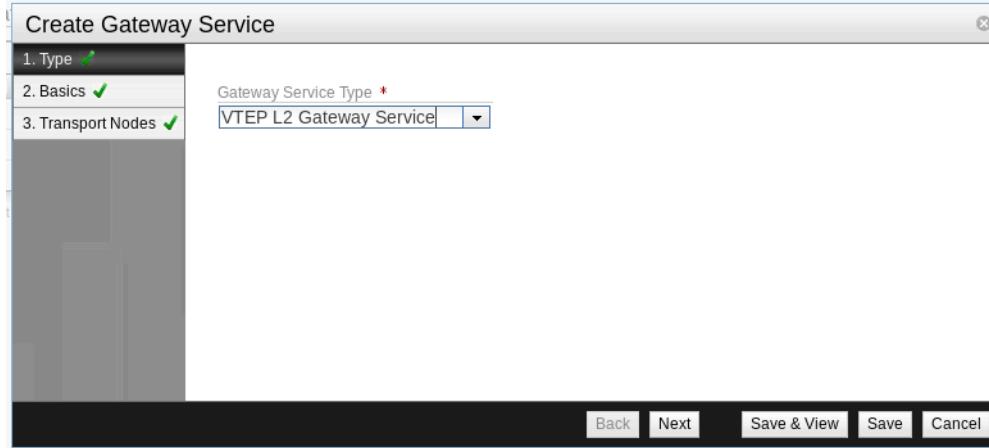
Verify the controller and switch handshake is successful. In a terminal connected to the switch, run this command:

```
cumulus@switch:~$ sudo ovsdb-client dump -f list | grep -A 7 "Manager"
Manager table
  _uuid          : 505f32af-9acb-4182-a315-022e405aa479
  inactivity_probe   : 30000
  is_connected      : true
  max_backoff       : []
  other_config      : {}
  status            : {sec_since_connect="18223", sec_since_disconnect="
  18225", state=ACTIVE}
  target            : "ssl:192.168.100.17:6632"
```

## Configuring the Transport Layer

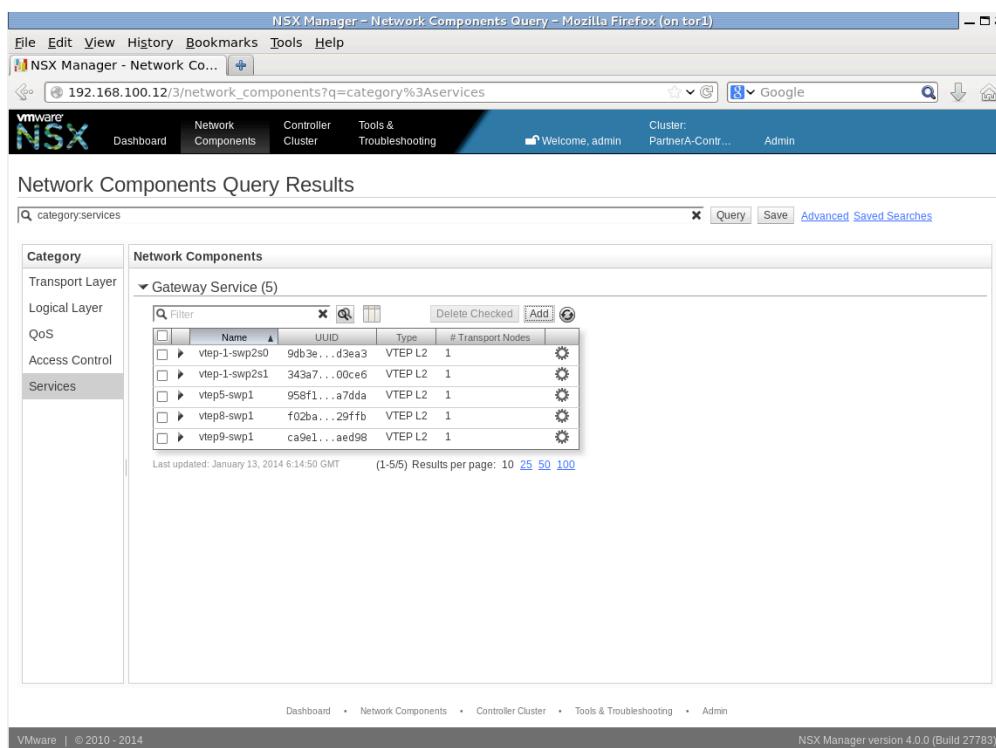
After you finish bootstrapping the NSX integration, you need to configure the transport layer. For each host-facing switch port that is to be associated with a VXLAN instance, define a **Gateway Service** for the port.

1. In NSX Manager, add a new gateway service. Click the **Network Components** tab, then the **Services** category. Under **Gateway Service**, click **Add**. The Create Gateway Service wizard appears.
2. In the Create Gateway Service dialog, select *VTEP L2 Gateway Service* as the **Gateway Service Type**.



3. Give the service a **Display Name** to represent the VTEP in NSX.
4. Click **Add Gateway** to associate the service with the gateway you created earlier.
5. In the **Transport Node** field, choose the name of the gateway you created earlier.
6. In the **Port ID** field, choose the physical port on the gateway (for example, swp10) that will connect to a logical L2 segment and carry data traffic.
7. Click **OK** to save this gateway in the service, then click **Save** to save the gateway service.

The gateway service shows up as type *VTEP L2* in NSX.



The screenshot shows the NSX Manager interface for querying network components. The left sidebar has categories: Transport Layer, Logical Layer, QoS, Access Control, and Services (which is selected). The main area displays a table titled 'Network Components' under the 'Gateway Service (5)' section. The table columns are Name, UUID, Type, and # Transport Nodes. The data shows five entries, each with a delete icon and a gear icon for more options. The table includes a filter bar at the top and pagination links (1-5/5) at the bottom.

Category	Network Components			
<b>Gateway Service (5)</b>				
	Name	UUID	Type	# Transport Nodes
	vtep-1-swp2s0	9db3e...d3ea3	VTEP L2	1
	vtep-1-swp2s1	343a7...00ce6	VTEP L2	1
	vtep5-swp1	958f1...a7dda	VTEP L2	1
	vtep8-swp1	f02ba...29ff8	VTEP L2	1
	vtep9-swp1	ca9e1...aed98	VTEP L2	1

Last updated: January 13, 2014 6:14:50 GMT      (1-5/5) Results per page: 10 [25](#) [50](#) [100](#)

Dashboard • Network Components • Controller Cluster • Tools & Troubleshooting • Admin

VMware | © 2010 - 2014      NSX Manager version 4.0.0 (Build 27783)

Next, you will configure the logical layer on NSX.

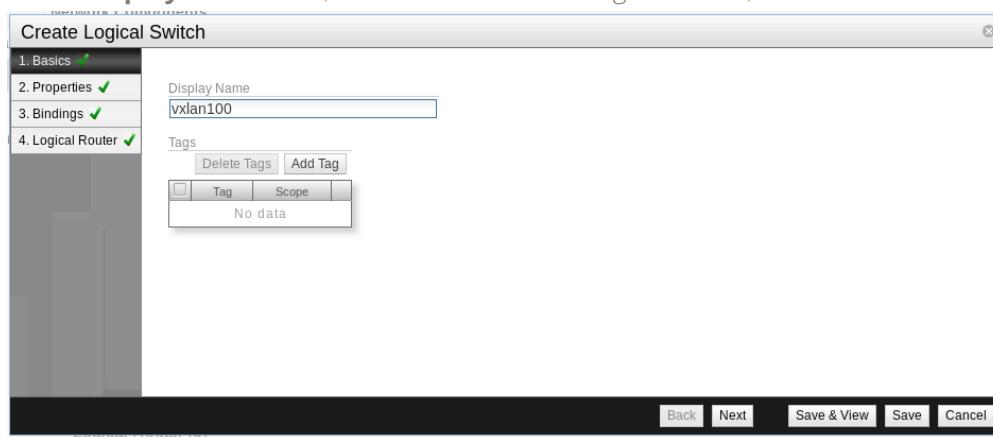
## Configuring the Logical Layer

To complete the integration with NSX, you need to configure the logical layer, which requires defining a logical switch (the VXLAN instance) and all the logical ports needed.

### Defining Logical Switches

To define the logical switch, do the following:

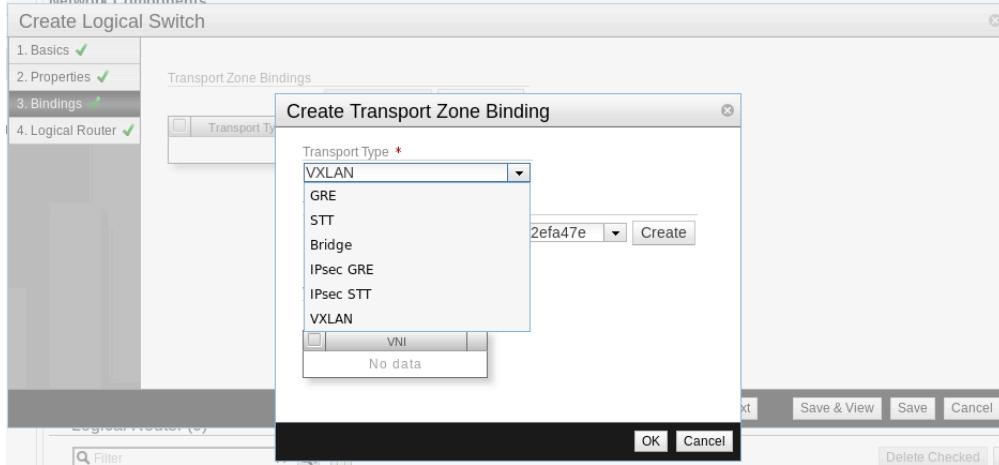
1. In NSX Manager, add a new logical switch. Click the **Network Components** tab, then the **Logical Layer** category. Under **Logical Switch**, click **Add**. The Create Logical Switch wizard appears.
2. In the **Display Name** field, enter a name for the logical switch, then click **Next**.



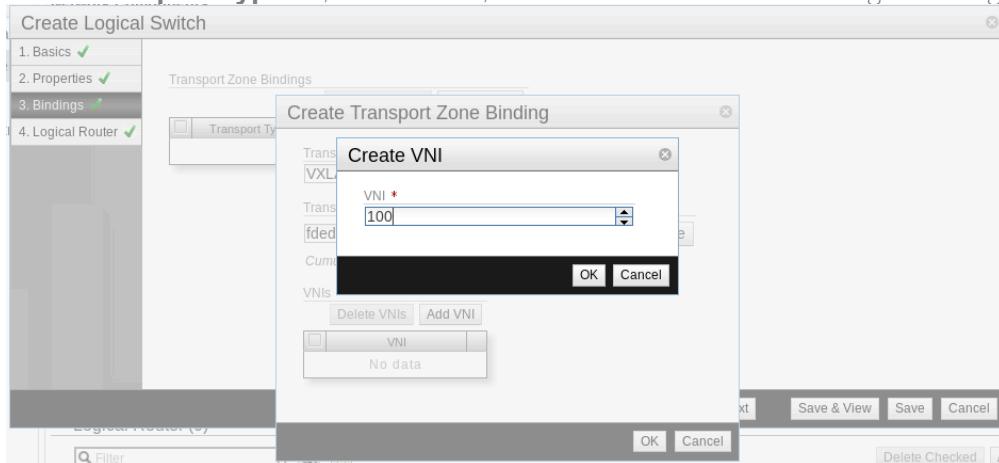
The screenshot shows the 'Create Logical Switch' wizard, step 1: Basics. The 'Display Name' field is populated with 'vxlan100'. Below it is a 'Tags' section with a 'Delete Tags' button and an 'Add Tag' button. A table below shows 'No data'. At the bottom are 'Back', 'Next', 'Save & View', 'Save', and 'Cancel' buttons.

3. Under **Replication Mode**, select **Service Nodes**, then click **Next**.

4. Specify the transport zone bindings for the logical switch. Click **Add Binding**. The Create Transport Zone Binding dialog appears.



5. In the **Transport Type** list, select **VXLAN**, then click **OK** to add the binding to the logical switch.

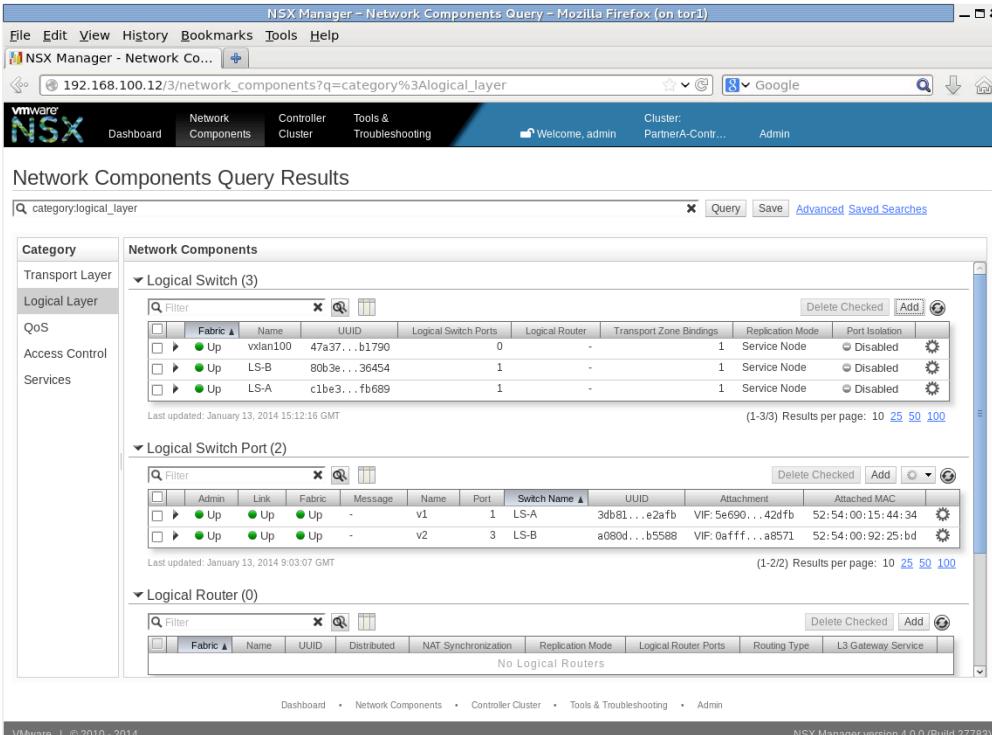


6. In the **VNI** field, assign the switch a VNI ID, then click **OK**.



Do not use 0 or 16777215 as the VNI ID, as they are reserved values under Cumulus Linux.

7. Click **Save** to save the logical switch configuration.



The screenshot shows the NSX Manager interface for querying network components. The main title is "NSX Manager – Network Components Query – Mozilla Firefox (on tor1)". The URL is "192.168.100.12:3/network\_components?q=category%3Alogical\_layer". The left sidebar has categories: Transport Layer, Logical Layer (selected), QoS, Access Control, and Services. The main content area displays two tables:

- Logical Switch (3)**: Shows three entries:
 

Fabric	Name	UUID	Logical Switch Ports	Logical Router	Transport Zone Bindings	Replication Mode	Port Isolation
Up	vxlan100	47a37...b1790	0	-	1	Service Node	Disabled
Up	LS-B	80b3e...36454	1	-	1	Service Node	Disabled
Up	LS-A	c1be3...fb689	1	-	1	Service Node	Disabled

 Last updated: January 13, 2014 15:12:16 GMT
- Logical Switch Port (2)**: Shows two entries:
 

Admin	Link	Fabric	Message	Name	Port	Switch Name	UUID	Attachment	Attached MAC
Up	Up	Up	-	v1	1	LS-A	3db81...e2af8	VIF: 5e690...42dfb	52:54:00:15:44:34
Up	Up	Up	-	v2	3	LS-B	a080d...b5588	VIF: 0afff...a8571	52:54:00:92:25:bd

 Last updated: January 13, 2014 9:03:07 GMT

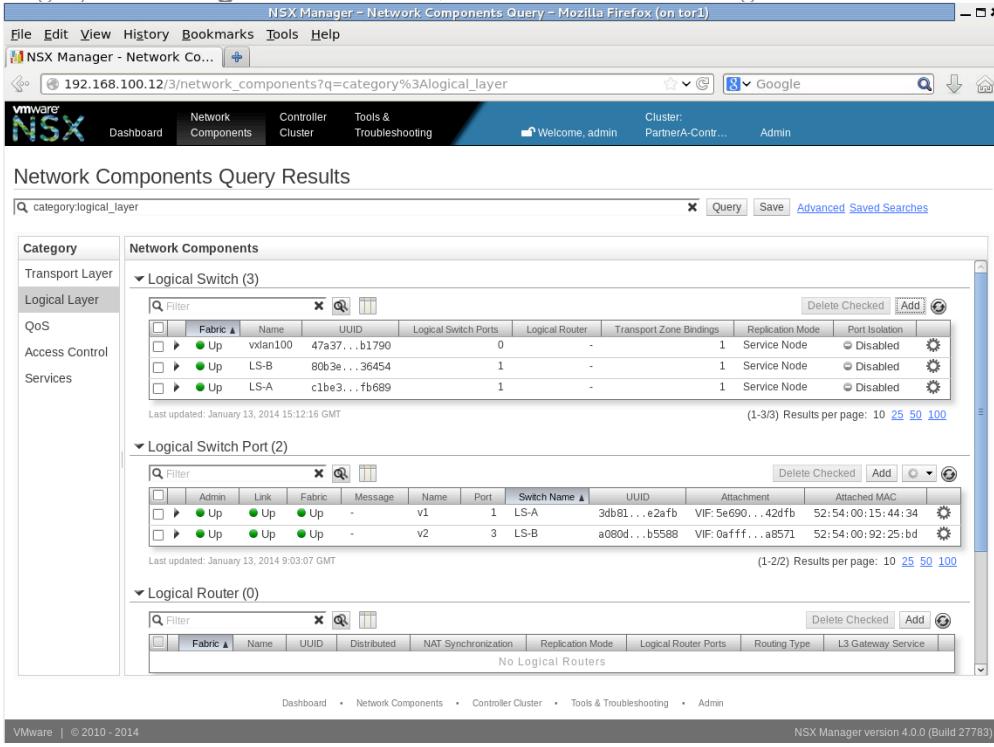
At the bottom, there are navigation links: Dashboard, Network Components, Controller Cluster, Tools & Troubleshooting, Admin, and footer text: VMware | © 2010 - 2014 and NSX Manager version 4.0.0 (Build 27783).

## Defining Logical Switch Ports

As the final step, define the logical switch ports. They can be virtual machine VIF interfaces from a registered OVS, or a VTEP gateway service instance on this switch, as defined above in the Configuring the Transport Layer. A VLAN binding can be defined for each VTEP gateway service associated with the particular logical switch.

To define the logical switch ports, do the following:

- In NSX Manager, add a new logical switch port. Click the **Network Components** tab, then the **Logical Layer** category. Under **Logical Switch Port**, click **Add**. The Create Logical Switch Port

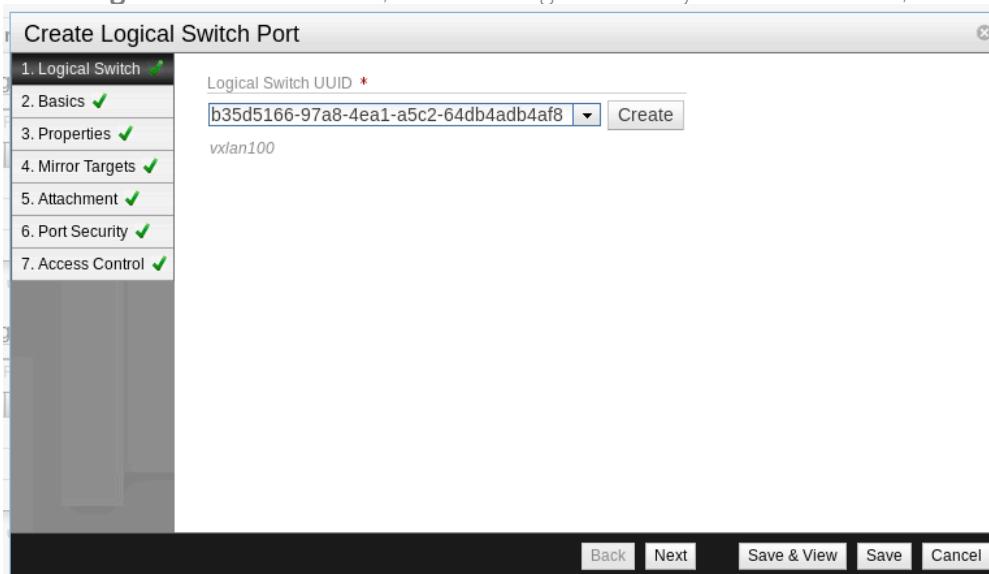


The screenshot shows the NSX Manager interface with the 'Network Components' tab selected. In the left sidebar, 'Logical Layer' is chosen. The main area displays 'Network Components Query Results' with two tables:

- Logical Switch (3)**: Shows three entries: vxlan100 (Fabric: Up, Name: vxlan100, UUID: 47a37...b1790, Logical Switch Ports: 0, Logical Router: -, Transport Zone Bindings: 1, Replication Mode: Service Node, Port Isolation: Disabled), LS-B (Fabric: Up, Name: LS-B, UUID: 80b3e...36454, Logical Switch Ports: 1, Logical Router: -, Transport Zone Bindings: 1, Replication Mode: Service Node, Port Isolation: Disabled), and LS-A (Fabric: Up, Name: LS-A, UUID: c1be3...fb689, Logical Switch Ports: 1, Logical Router: -, Transport Zone Bindings: 1, Replication Mode: Service Node, Port Isolation: Disabled).
- Logical Switch Port (2)**: Shows two entries: v1 (Admin: Up, Link: Up, Fabric: Up, Message: -, Name: v1, Port: 1, Switch Name: LS-A, UUID: 3db81...e2afb, VIF: 5e690...42dfb, Attachment: 52:54:00:15:44:34, Attached MAC: 00:0c:29:44:34:34) and v2 (Admin: Up, Link: Up, Fabric: Up, Message: -, Name: v2, Port: 3, Switch Name: LS-B, UUID: a080d...b5588, VIF: 0affff...a8571, Attachment: 52:54:00:92:25:bd, Attached MAC: 00:0c:29:44:34:34).

At the bottom, there are links to 'Dashboard', 'Network Components', 'Controller Cluster', 'Tools & Troubleshooting', and 'Admin'. The footer indicates 'NSX Manager version 4.0.0 (Build 27783)'.

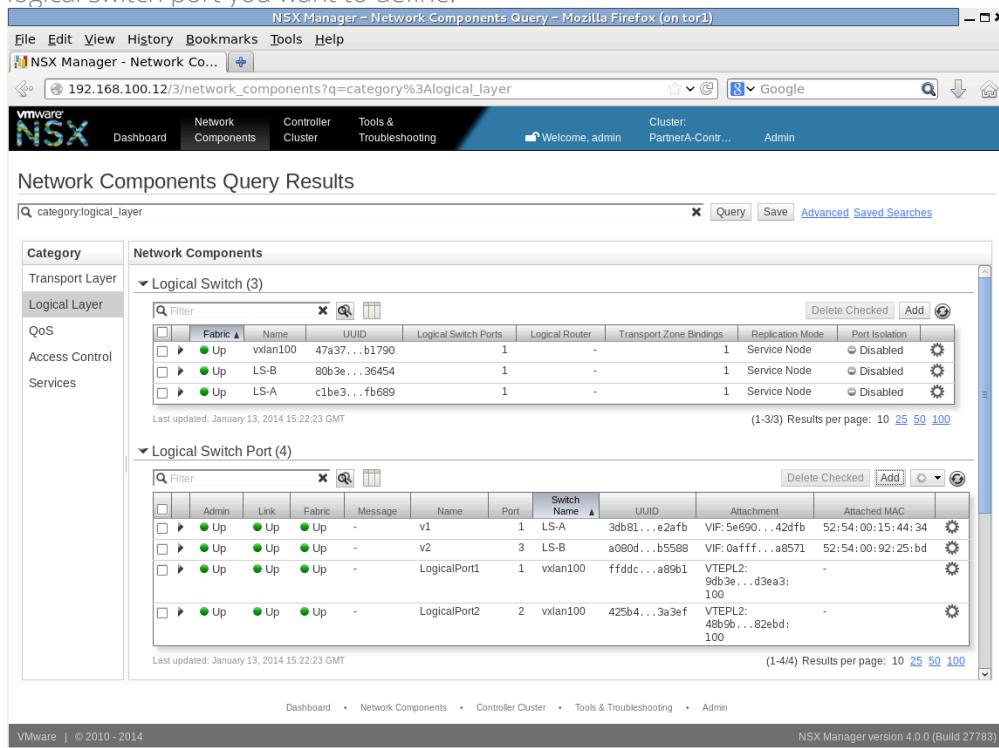
- wizard appears.
- In the **Logical Switch UUID** list, select the logical switch you created above, then click **Create**.



The screenshot shows the 'Create Logical Switch Port' wizard. Step 1: Logical Switch is selected. The 'Logical Switch UUID' dropdown contains 'b35d5166-97a8-4ea1-a5c2-64db4adb4af8' (vxlan100). Below the dropdown are buttons for 'Create', 'Back', 'Next', 'Save & View', 'Save', and 'Cancel'.

- In the **Display Name** field, give the port a name that indicates it is the port that connects the gateway, then click **Next**.
- In the **Attachment Type** list, select *VTEP L2 Gateway*.
- In the **VTEP L2 Gateway Service UUID** list, choose the name of the gateway service you created earlier.
- In the **VLAN** list, you can optionally choose a VLAN if you wish to connect only traffic on a specific VLAN of the physical network. Leave it blank to handle all traffic.

- Click **Save** to save the logical switch port. Connectivity is established. Repeat this procedure for each logical switch port you want to define.



The screenshot shows the NSX Manager interface for Network Components Query. The left sidebar has categories: Transport Layer, Logical Layer (selected), QoS, Access Control, and Services. The main area displays two tables:

- Logical Switch (3)**: Shows three entries. The first entry is vxlan100, which is up, has 47a37... b1790 as its UUID, and has 1 Logical Switch Ports, 1 Logical Router, and 1 Service Node. The second and third entries are LS-B and LS-A respectively, both with 80b3e... 36454 as their UUID, 1 Logical Switch Ports, 1 Logical Router, and 1 Service Node. All ports are marked as disabled.
- Logical Switch Port (4)**: Shows four entries. The first entry is v1, which is up, has 3db81... e2af as its UUID, and is attached to LS-A. The second entry is v2, which is up, has a080d... b5598 as its UUID, and is attached to LS-B. The third entry is LogicalPort1, which is up, has fffddc... a89b1 as its UUID, and is attached to vxlan100. The fourth entry is LogicalPort2, which is up, has 425b4... 3a3ef as its UUID, and is attached to vxlan100.

Both tables have pagination at the bottom: (1-3/3) Results per page: 10 25 50 100 and (1-4/4) Results per page: 10 25 50 100.

## Verifying the VXLAN Configuration

Once configured, you can verify the VXLAN configuration using these Cumulus Linux commands in a terminal connected to the switch:

```
cumulus@switch1:~$ sudo ip -d link show vxln100
71: vxln100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    master br-vxln100 state UNKNOWN mode DEFAULT
        link/ether d2:ca:78:bb:7c:9b brd ff:ff:ff:ff:ff:ff
        vxlan id 100 local 172.16.20.157 port 32768 61000 nolearning ageing 300
        svcnode 172.16.21.125
```

OR

```
cumulus@switch1:~$ sudo bridge fdb show
52:54:00:ae:2a:e0 dev vxln100 dst 172.16.21.150 self permanent
d2:ca:78:bb:7c:9b dev vxln100 permanent
90:e2:ba:3f:ce:34 dev swp2s1.100
90:e2:ba:3f:ce:35 dev swp2s0.100
44:38:39:00:48:0e dev swp2s1.100 permanent
44:38:39:00:48:0d dev swp2s0.100 permanent
```

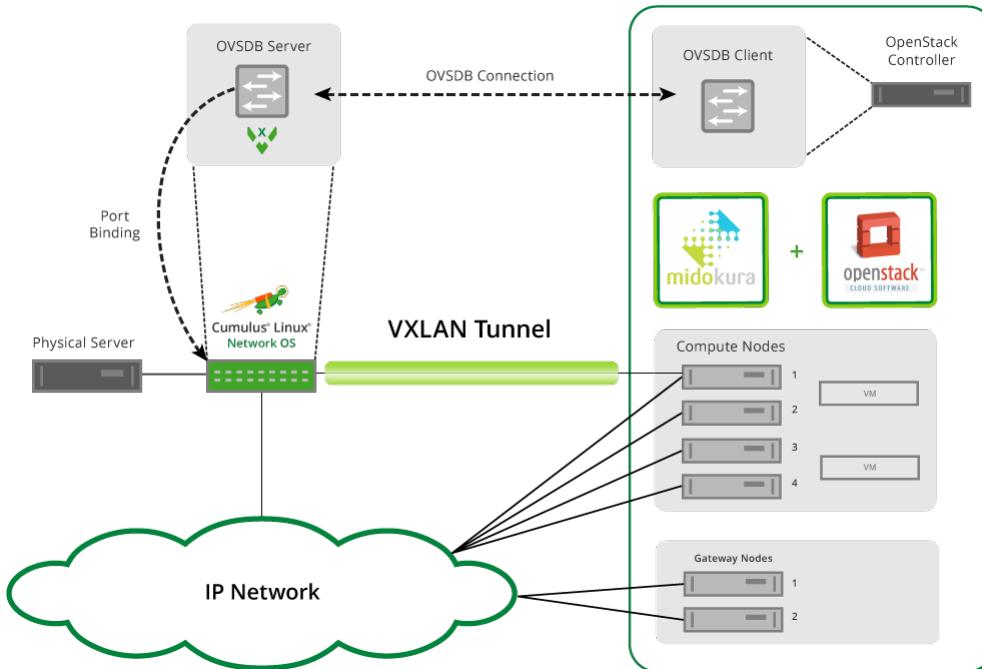
## Troubleshooting VXLANS in NSX

Use `ovsdb-client dump` to troubleshoot issues on the switch. It verifies that the controller and switch handshake is successful. This command works only for VXLANS integrated with NSX:

```
cumulus@switch:~$ sudo ovsdb-client dump -f list | grep -A 7 "Manager"
Manager table
_uuid : 505f32af-9acb-4182-a315-022e405aa479
_inactivity_probe : 30000
_is_connected : true
_max_backoff : []
_other_config : {}
_status : {sec_since_connect="18223", sec_since_disconnect="18225", state=ACTIVE}
_target : "ssl:192.168.100.17:6632"
```

## Integrating Hardware VTEPs with Midokura MidoNet and OpenStack

Cumulus Linux seamlessly integrates with the MidoNet OpenStack infrastructure, where the switches provide the VTEP gateway for terminating VXLAN tunnels from within MidoNet. MidoNet connects to the OVSDB server running on the Cumulus Linux switch, and exchanges information about the VTEPs and MAC addresses associated with the OpenStack Neutron networks. This provides seamless Ethernet connectivity between virtual and physical server infrastructures.



## Contents

- Contents (see page 284)

- Getting Started (see page 285)
  - Caveats and Errata (see page 286)
  - Preparing for the MidoNet Integration (see page 286)
    - Enabling the openvswitch-vtep Package (see page 286)
  - Bootstrapping the OVSDB Server and VTEP (see page 287)
    - Automating with the Bootstrap Script (see page 287)
    - Manually Bootstrapping (see page 288)
  - Configuring MidoNet VTEP and Port Bindings (see page 288)
    - Using the MidoNet Manager GUI (see page 288)
      - Creating a Tunnel Zone (see page 288)
      - Adding Hosts to a Tunnel Zone (see page 289)
      - Creating the VTEP (see page 290)
      - Binding Ports to the VTEP (see page 291)
    - Using the MidoNet CLI (see page 292)
- Troubleshooting MidoNet and Cumulus VTEPs (see page 294)
  - Troubleshooting the Control Plane (see page 295)
    - Verifying VTEP and OVSDB Services (see page 295)
    - Verifying OVSDB-server Connections (see page 295)
    - Verifying the VXLAN Bridge and VTEP Interfaces (see page 295)
  - Datapath Troubleshooting (see page 296)
    - Verifying IP Reachability (see page 297)
    - MidoNet VXLAN Encapsulation (see page 297)
  - Inspecting the OVSDB (see page 298)
    - Using VTEP-CTL (see page 298)
      - Listing the Physical Switch (see page 298)
      - Listing the Logical Switch (see page 298)
      - Listing Local or Remote MAC Addresses (see page 298)
    - Getting Open Vswitch Database (OVSDB) Data (see page 299)

## ***Getting Started***

Before you create VXLANs with MidoNet, make sure you have the following components:

- A switch (L2 gateway) with a Tomahawk, Trident II+ or Trident II chipset running Cumulus Linux 2.0 and later
- OVSDB server (`ovsdb-server`), included in Cumulus Linux 2.0 and later
- VTEPD (`ovs-vtepd`), included in Cumulus Linux 2.0 and later

Integrating a VXLAN with MidoNet involves:

- Preparing for the MidoNet integration
- Bootstrapping the OVS and VTEP

- Configuring the MidoNet VTEP binding
- Verifying the VXLAN configuration

## Caveats and Errata

- There is no support for VXLAN routing in the Tomahawk, Trident II+ and Trident II chipsets; use a loopback interface or external router.
- For more information about MidoNet, see the MidoNet Operations Guide, version 1.8 or later.

## Preparing for the MidoNet Integration

Before you start configuring the MidoNet tunnel zones, VTEP binding and connecting virtual ports to the VXLAN, you need to complete the bootstrap process on each switch to which you plan to build VXLAN tunnels. This creates the VTEP gateway and initializes the OVS database server. You only need to do the bootstrapping once, before you begin the MidoNet integration.

### Enabling the `openvswitch-vtep` Package

Before you start bootstrapping the integration, you need to enable the `openvswitch-vtep` package, since it is disabled by default in Cumulus Linux.

1. Edit the `/etc/default/openvswitch-vtep` file, changing the `START` option from `no` to `yes`. This simple `sed` command does this, and creates a backup as well:

```
sudo sed -i.bak s/START=no/START=yes/g /etc/default/openvswitch-vtep
```



Make sure to include this file in persistent storage prior to Cumulus Linux upgrades.

2. Start the daemon:

```
cumulus@switch$ sudo systemctl start openvswitch-vtep.service
```



Prior to Cumulus Linux 2.5.1, you must edit the control file `/usr/share/openvswitch/scripts/ovs-ctl-vtep`, by adding a parameter for the remote OVS connection. The OVS server connection is not encrypted, so it is necessary to change the PTCP port in this file. The following `sed` command makes the proper change:

```
sed -i.bak "/remote=db/a \\\tset \"\$@\\" --remote=ptcp:6632"
/usr/share/openvswitch/scripts/ovs-ctl-vtep
```

## Bootstrapping the OVSDB Server and VTEP

### Automating with the Bootstrap Script

The `vtep-bootstrap` script is available so you can do the bootstrapping automatically. For information, read `man vtep-bootstrap`. This script requires three parameters, in this order:

- Switch name: The name of the switch that is the VTEP gateway.
- Tunnel IP address: The datapath IP address of the VTEP.
- Management IP address: The IP address of the switch's management interface.

For example, click here ...

```
root@sw11:~# vtep-bootstrap sw11 10.111.1.1 10.50.20.21 --no_encryption
```

```
Executed:  
define physical switch  
().  
Executed:  
define local tunnel IP address on the switch  
().  
Executed:  
define management IP address on the switch  
().  
Executed:  
restart a service  
(Killing ovs-vtepd (28170).  
Killing ovsdb-server (28146).  
Starting ovsdb-server.  
Starting ovs-vtepd.).
```



Prior to Cumulus Linux 2.5.1, the `vtep-bootstrap` command required 4 parameters:  
`<switch_name> <controller_ip> <tunnel_ip> <management_ip>`

Since Midonet does not have a controller, you need to use a dummy IP address (for example, 1.1.1.1) for the controller parameter in the bootstrap script. After the script completes, delete the VTEP manager, since it is not needed and will otherwise fill the logs with inconsequential error messages:

```
vtep-ctl del-manager
```

## Manually Bootstrapping

If you don't use the bootstrap script, then you must initialize the OVS database instance manually, and create the VTEP.

Perform the following commands in order (see the automated bootstrapping example above for values):

1. Define the switch in OVSDB:

```
sudo vtep-ctl add-ps <switch_name>
```

2. Define the VTEP tunnel IP address:

```
sudo vtep-ctl set Physical_switch <switch_name> tunnel_ip=<tunnel_ip>
```

3. Define the management interface IP address:

```
sudo vtep-ctl set Physical_switch <switch_name>
management_ip=<management_ip>
```

4. Restart the OVSDB server and `vtep`:

```
sudo systemctl restart openvswitch-vtep.service
```

At this point, the switch is ready to connect to MidoNet. The rest of the configuration is performed in the MidoNet Manager GUI, or using the MidoNet API.

## Configuring MidoNet VTEP and Port Bindings

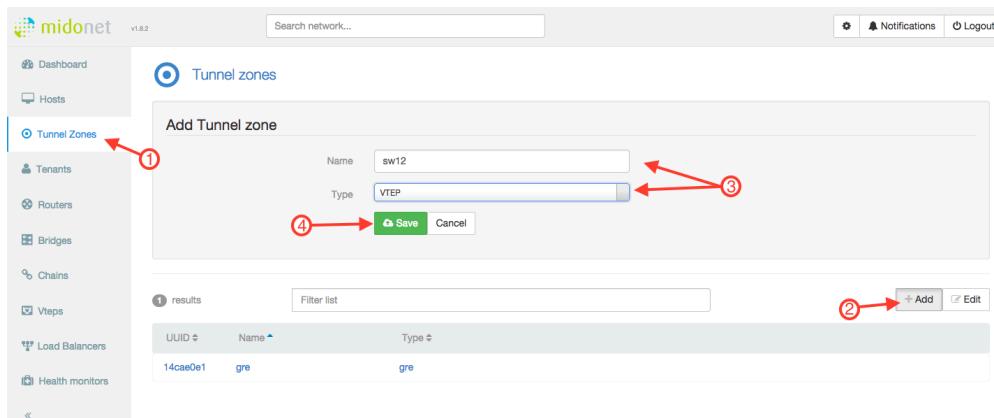
This part of the configuration sets up MidoNet and OpenStack to connect the virtualization environment to the Cumulus Linux switch. The `midonet-agent` is the networking component that manages the VXLAN, while the Open Virtual Switch (OVS) client on the OpenStack controller node communicates MAC address information between the `midonet-agent` and the Cumulus Linux OVS database (OVSDB) server.

## Using the MidoNet Manager GUI

Creating a Tunnel Zone

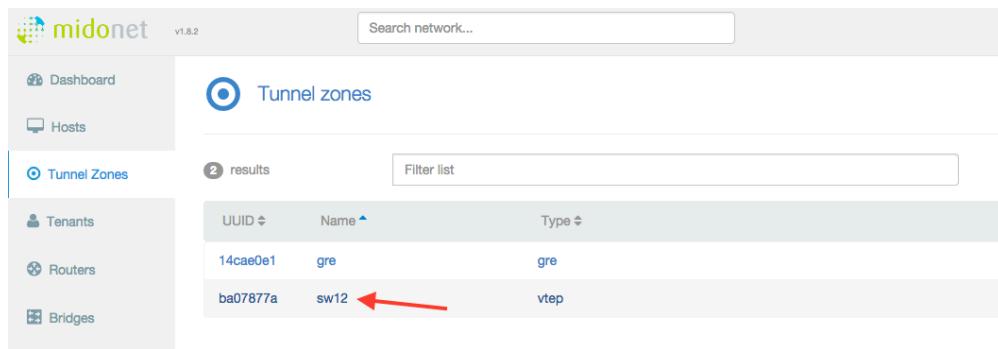
1. Click **Tunnel Zones** in the menu on the left side.
2. Click **Add**.
3. Give the tunnel zone a **Name** and select "**VTEP**" for the **Type**.

4. Click **Save**.



### Adding Hosts to a Tunnel Zone

Once the tunnel zone is created, click the name of the tunnel zone to view the hosts table.

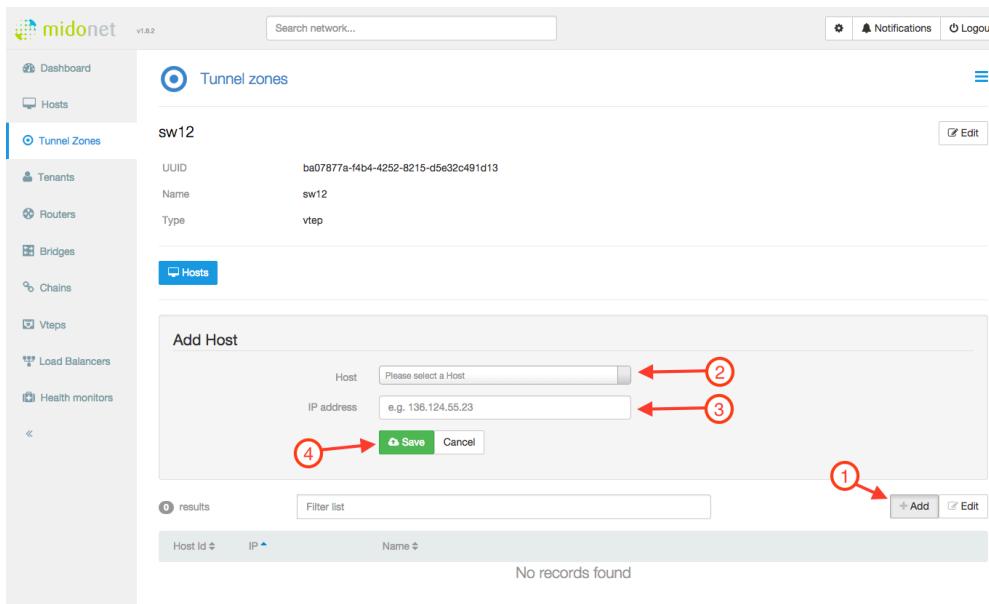


UUID	Name	Type
14cae0e1	gre	gre
ba07877a	sw12	vtep

The tunnel zone is a construct used to define the VXLAN source address used for the tunnel. This host's address is used for the source of the VXLAN encapsulation, and traffic will transit into the routing domain from this point. Thus, the host must have layer 3 reachability to the Cumulus Linux switch tunnel IP.

Next, add a host entry to the tunnel zone:

1. Click **Add**.
2. Select a host from the **Host** list.
3. Provide the tunnel source **IP Address** to use on the selected host.
4. Click **Save**.



The screenshot shows the midonet web interface. On the left sidebar, under the 'Tunnel Zones' section, there is a 'Hosts' tab which is currently selected. In the main content area, a 'Tunnel zones' page for 'sw12' is displayed. At the top right of this page is an 'Edit' button. Below the zone details, there is a 'Hosts' section with a 'Hosts' button. A modal window titled 'Add Host' is open, containing fields for 'Host' (with placeholder 'Please select a Host') and 'IP address' (with placeholder 'e.g. 136.124.55.23'). At the bottom of the modal are 'Save' and 'Cancel' buttons. Red numbered circles (1, 2, 3, 4) are overlaid on the interface: circle 1 points to the '+ Add' button in the 'Hosts' section; circle 2 points to the 'Host' input field in the modal; circle 3 points to the 'IP address' input field in the modal; circle 4 points to the 'Save' button in the modal. Below the modal, the host list table shows one entry: '4d03509b' (Host ID), '10.50.21.182' (IP), and 'os-compute1' (Name). The table has columns for Host Id, IP, and Name, with sorting arrows for IP and Name.

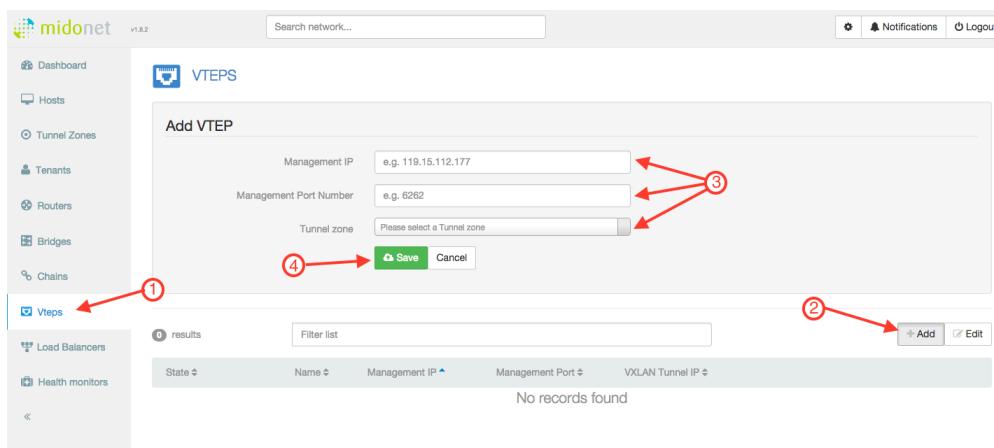
The host list now displays the new entry:



The screenshot shows the 'Hosts' list page. The 'Hosts' button in the sidebar is highlighted. The main area displays a table with one row. The columns are 'Host Id', 'IP', and 'Name'. The row contains the values: '4d03509b', '10.50.21.182', and 'os-compute1'. There are buttons for '+ Add' and 'Edit' at the top right of the table.

## Creating the VTEP

1. Click the **Vtaps** menu on the left side.
2. Click **Add**.
3. Fill out the fields using the same information you used earlier on the switch for the bootstrap procedure:
  - **Management IP** is typically the eth0 address of the switch. This tells the OVS-client to connect to the OVSDB-server on the Cumulus Linux switch.
  - **Management Port Number** is the PTCP port you configured in the `ovs-ctl-vtep` script earlier (the example uses 6632).
  - **Tunnel Zone** is the name of the zone you created in the previous procedure.
4. Click **Save**.



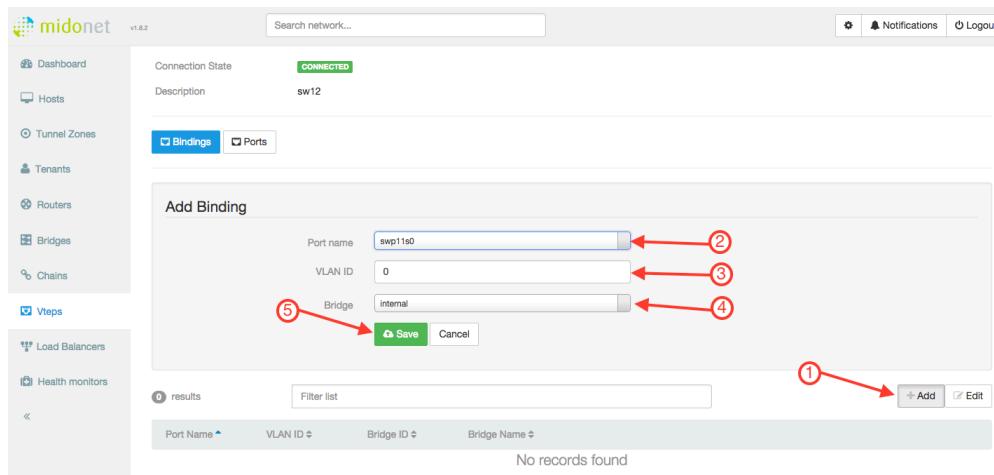
The new VTEP appears in the list below. MidoNet then initiates a connection between the OpenStack Controller and the Cumulus Linux switch. If the OVS client is successfully connected to the OVSDB server, the VTEP entry should display the switch name and VXLAN tunnel IP address, which you specified during the bootstrapping process.

VTEPs				
State	Name	Management IP	Management Port	VXLAN Tunnel IP
CONNECTED	sw11	10.50.20.21	6632	10.111.1.1
CONNECTED	sw12	10.50.20.22	6632	10.111.1.2

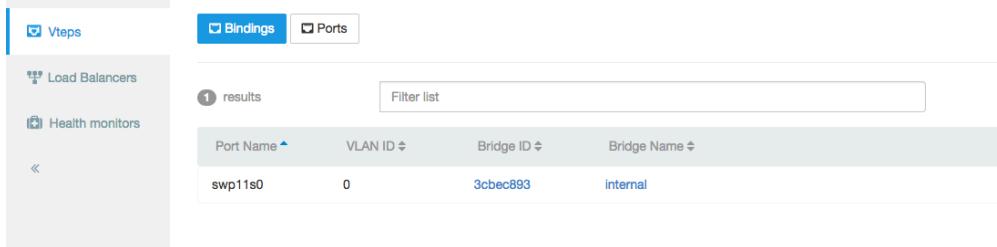
### Binding Ports to the VTEP

Now that connectivity is established to the switch, you need to add a physical port binding to the VTEP on the Cumulus Linux switch:

1. Click **Add**.
2. In the **Port Name** list, select the port on the Cumulus Linux switch that you are using to connect to the VXLAN segment.
3. Specify the **VLAN ID** (enter 0 for untagged).
4. In the **Bridge** list, select the MidoNet bridge that the instances (VMs) are using in OpenStack.
5. Click **Save**.



You should see the port binding displayed in the binding table under the VTEP.



Once the port is bound, this automatically configures a VXLAN bridge interface, and includes the VTEP interface and the port bound to the bridge. Now the OpenStack instances (VMs) should be able to ping the hosts connected to the bound port on the Cumulus switch. The Troubleshooting section below demonstrates the verification of the VXLAN data and control planes.

## Using the MidoNet CLI

To get started with the MidoNet CLI, you can access the CLI prompt on the OpenStack Controller:

```
root@os-controller:~# midonet-cli
midonet>
```

Now from the MidoNet CLI, the commands explained in this section perform the same operations depicted in the previous section with the MidoNet Manager GUI.

1. Create a tunnel zone with a name and type vtep:

```
midonet> tunnel-zone create name sw12 type vtep
tzone1
```

2. The tunnel zone is a construct used to define the VXLAN source address used for the tunnel. This host's address is used for the source of the VXLAN encapsulation, and traffic will transit into the routing domain from this point. Thus, the host must have layer 3 reachability to the Cumulus Linux switch tunnel IP.

- First, get the list of available hosts connected to the Neutron network and the MidoNet bridge.
- Next, get a listing of all the interfaces.
- Finally, add a host entry to the tunnel zone ID returned in the previous step, and specify which interface address to use.

```

midonet> list host
host host0 name os-compute1 alive true
host host1 name os-network alive true
midonet> host host0 list interface
iface midonet host_id host0 status 0 addresses [] mac 02:4b:
38:92:dd:ce mtu 1500 type Virtual endpoint DATAPATH
iface lo host_id host0 status 3 addresses [u'127.0.0.1',
u'169.254.169.254', u'0:0:0:0:0:0:0:1'] mac 00:00:00:00:00:
00 mtu 65536 type Virtual endpoint LOCALHOST
iface virbr0 host_id host0 status 1 addresses
[u'192.168.122.1'] mac 22:6e:63:90:1f:69 mtu 1500 type
Virtual endpoint UNKNOWN
iface tap7cf84c-26 host_id host0 status 3 addresses
[u'fe80:0:0:0:e822:94ff:fee2:d41b'] mac ea:22:94:e2:d4:1b
mtu 65000 type Virtual endpoint DATAPATH
iface eth1 host_id host0 status 3 addresses
[u'10.111.0.182', u'fe80:0:0:0:5054:ff:fe85:acd6'] mac 52:
54:00:85:ac:d6 mtu 1500 type Physical endpoint PHYSICAL
iface tapfd4abcea-df host_id host0 status 3 addresses
[u'fe80:0:0:0:14b3:45ff:fe94:5b07'] mac 16:b3:45:94:5b:07
mtu 65000 type Virtual endpoint DATAPATH
iface eth0 host_id host0 status 3 addresses
[u'10.50.21.182', u'fe80:0:0:0:5054:ff:feef:c5dc'] mac 52:
54:00:ef:c5:dc mtu 1500 type Physical endpoint PHYSICAL
midonet> tunnel-zone tzone0 add member host host0 address
10.111.0.182
zone tzone0 host host0 address 10.111.0.182

```

Repeat this procedure for each OpenStack host connected to the Neutron network and the MidoNet bridge.

3. Create a VTEP and assign it to the tunnel zone ID returned in the previous step. The management IP address (the destination address for the VXLAN/remote VTEP) and the port must be the same ones you configured in the `vtep-bootstrap` script or the manual bootstrapping:

```
midonet> vtep add management-ip 10.50.20.22 management-port 6632
tunnel-zone tzone0
name sw12 description sw12 management-ip 10.50.20.22 management-port
6632 tunnel-zone tzone0 connection-state CONNECTED
```

In this step, MidoNet initiates a connection between the OpenStack Controller and the Cumulus Linux switch. If the OVS client is successfully connected to the OVSDB server, the returned values should show the name and description matching the `switch-name` parameter specified in the bootstrap process.



Verify the connection-state as CONNECTED, otherwise if ERROR is returned, you must debug. Typically this only fails if the `management-ip` and/or `management-port` settings are wrong.

4. The VTEP binding uses the information provided to MidoNet from the OVSDB server, providing a list of ports that the hardware VTEP can use for layer 2 attachment. This binding virtually connects the physical interface to the overlay switch, and joins it to the Neutron bridged network.

First, get the UUID of the Neutron network behind the MidoNet bridge:

```
midonet> list bridge
bridge bridge0 name internal state up
bridge bridge1 name internal2 state up
midonet> show bridge bridge1 id
6c9826da-6655-4fe3-a826-4dcba6477d2d
```

Next, create the VTEP binding, using the UUID and the switch port being bound to the VTEP on the remote end. If there is no VLAN ID, set `vlan` to 0:

```
midonet> vtep name sw12 binding add network-id 6c9826da-6655-4fe3-a826-
4dcba6477d2d physical-port swp11s0 vlan 0
management-ip 10.50.20.22 physical-port swp11s0 vlan 0 network-id
6c9826da-6655-4fe3-a826-4dcba6477d2d
```

At this point, the VTEP should be connected, and the layer 2 overlay should be operational. From the openstack instance (VM), you should be able to ping a physical server connected to the port bound to the hardware switch VTEP.

## Troubleshooting MidoNet and Cumulus VTEPs

As with any complex system, there is a control plane and data plane.

### Troubleshooting the Control Plane

In this solution, the control plane consists of the connection between the OpenStack Controller, and each Cumulus Linux switch running the `ovsdb-server` and `vtep` daemons.

### Verifying VTEP and OVSDB Services

First, it is important that the OVSDB server and `ovs-vtep` daemon are running. Verify this is the case:

```
cumulus@switch12:~$ systemctl status openvswitch-vtep.service
ovsdb-server is running with pid 17440
ovs-vtep is running with pid 17444
```

### Verifying OVSDB-server Connections

From the OpenStack Controller host, verify that it can connect to the `ovsdb-server`. Telnet to the switch IP address on port 6632:

```
root@os-controller:~# telnet 10.50.20.22 6632
Trying 10.50.20.22...
Connected to 10.50.20.22.
Escape character is '^]'.
<Ctrl+c>
Connection closed by foreign host.
```

If the connection fails, verify IP reachability from the host to the switch. If that succeeds, it is likely the bootstrap process did not set up port 6632. Redo the bootstrapping procedures above.

```
root@os-controller:~# ping -c1 10.50.20.22
PING 10.50.20.22 (10.50.20.22) 56(84) bytes of data.
64 bytes from 10.50.20.22: icmp_seq=1 ttl=63 time=0.315 ms
--- 10.50.20.22 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.315/0.315/0.315/0.000 ms
```

### Verifying the VXLAN Bridge and VTEP Interfaces

After creating the VTEP in MidoNet and adding an interface binding, you should see `br-vxln` and `vxln` interfaces on the switch. You can verify that the VXLAN bridge and VTEP interface are created and UP:

```
cumulus@switch12:~$ sudo brctl show
bridge name  bridge id          STP      enabled interfaces
br-vxln10006 8000.00e0ec2749a2    no       swp11s0
                                         vxln10006
cumulus@switch12:~$ sudo ip -d link show vxln10006
55: vxln10006: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-vxln10006 state UNKNOWN mode DEFAULT
  link/ether 72:94:eb:b6:6c:c3 brd ff:ff:ff:ff:ff:ff
    vxlan id 10006 local 10.111.1.2 port 32768 61000 nolearning ageing 300
    svcnode 10.111.0.182
      bridge_slave
```

Next, look at the bridging table for the VTEP and the forwarding entries. The bound interface and the VTEP should be listed along with the MAC addresses of those interfaces. When the hosts attached to the bound port send data, those MACs are learned, and entered into the bridging table, as well as the OVSDB.

```
cumulus@switch12:~$ brctl showmacs br-vxln10006
port name      mac addr           vlan      is
local?        ageing timer
swp11s0        00:e0:ec:27:49:a2   0
yes            0.00
swp11s0        64:ae:0c:32:f1:41   0
no              0.01
vxln10006     72:94:eb:b6:6c:c3   0
yes            0.00

cumulus@switch12:~$ sudo bridge fdb show br-vxln10006
fa:16:3e:14:04:2e dev vxln10004 dst 10.111.0.182 vlan 65535 self permanent
00:e0:ec:27:49:a2 dev swp11s0 vlan 0 master br-vxln10004 permanent
b6:71:33:3b:a7:83 dev vxln10004 vlan 0 master br-vxln10004 permanent
64:ae:0c:32:f1:41 dev swp11s0 vlan 0 master br-vxln10004
```

## Datapath Troubleshooting

If you have verified the control plane is correct, and you still cannot get data between the OpenStack instances and the physical nodes on the switch, there may be something wrong with the data plane. The data plane consists of the actual VXLAN encapsulated path, between one of the OpenStack nodes running the `midolman` service. This is typically the compute nodes, but can include the MidoNet gateway nodes. If the OpenStack instances can ping the tenant router address but cannot ping the physical device connected to the switch (or vice versa), then something is wrong in the data plane.

## Verifying IP Reachability

First, there must be IP reachability between the encapsulating node, and the address you bootstrapped as the tunnel IP on the switch. Verify the OpenStack host can ping the tunnel IP. If this doesn't work, check the routing design, and fix the layer 3 problem first.

```
root@os-compute1:~# ping -c1 10.111.1.2
PING 10.111.1.2 (10.111.1.2) 56(84) bytes of data.
64 bytes from 10.111.1.2: icmp_seq=1 ttl=62 time=0.649 ms
--- 10.111.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.649/0.649/0.649/0.000 ms
```

## MidoNet VXLAN Encapsulation

If the instance (VM) cannot ping the physical server, or the reply is not returning, look at the packets on the OpenStack node. Initiate a ping from the OpenStack instance, then using `tcpdump`, hopefully you can see the VXLAN data. This example displays what it looks like when it is working.

```
root@os-compute1:~# tcpdump -i eth1 -l -nnn -vvv -X -e port 4789
52:54:00:85:ac:d6 > 00:e0:ec:26:50:36, ethertype IPv4 (0x0800), length 148:
(tos 0x0, ttl 255, id 7583, offset 0, flags [none], proto UDP (17), length
134)
 10.111.0.182.41568 > 10.111.1.2.4789: [no cksum] VXLAN, flags [I] (0x08),
vni 10008
fa:16:3e:14:04:2e > 64:ae:0c:32:f1:41, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 64, id 64058, offset 0, flags [DF], proto ICMP (1), length 84)
 10.111.102.104 > 10.111.102.2: ICMP echo request, id 15873, seq 0, length
64
 0x0000: 4500 0086 1d9f 0000 ff11 8732 0a6f 00b6 E.....2.o..
 0x0010: 0a6f 0102 a260 12b5 0072 0000 0800 0000 .o...`....r....
 0x0020: 0027 1800 64ae 0c32 f141 fa16 3e14 042e .'..d..2.A..>...
 0x0030: 0800 4500 0054 fa3a 4000 4001 5f26 0a6f ..E..T.:@._&.
 0x0040: 6668 0a6f 6602 0800 f9de 3e01 0000 4233 fh.of.....>...B3
 0x0050: 7dec 0000 0000 0000 0000 0000 0000 0000 }.....
 0x0060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
 0x0070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
 0x0080: 0000 0000 0000 .....
00:e0:ec:26:50:36 > 52:54:00:85:ac:d6, ethertype IPv4 (0x0800), length 148:
(tos 0x0, ttl 62, id 2689, offset 0, flags [none], proto UDP (17), length
134)
 10.111.1.2.63385 > 10.111.0.182.4789: [no cksum] VXLAN, flags [I] (0x08),
```

```
vni 10008
64:ae:0c:32:f1:41 > fa:16:3e:14:04:2e, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 255, id 64058, offset 0, flags [DF], proto ICMP (1), length
84)
10.111.102.2 > 10.111.102.104: ICMP echo reply, id 15873, seq 0, length 64
0x0000: 4500 0086 0a81 0000 3e11 5b51 0a6f 0102 E.....>.Q.o..
0x0010: 0a6f 00b6 f799 12b5 0072 0000 0800 0000 .o.....r.....
0x0020: 0027 1800 fa16 3e14 042e 64ae 0c32 f141 .'....>...d..2.A
0x0030: 0800 4500 0054 fa3a 4000 ff01 a025 0a6f ..E..T.:@....%..o
0x0040: 6602 0a6f 6668 0000 01df 3e01 0000 4233 f..ofh....>...B3
0x0050: 7dec 0000 0000 0000 0000 0000 0000 0000 }.....
0x0060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x0080: 0000 0000 0000 .....
```

## Inspecting the OVSDB

### Using VTEP-CTL

These commands show you the information installed in the OVSDB. This database is structured using the *physical switch* ID, with one or more *logical switch* IDs associated with it. The bootstrap process creates the physical switch, and MidoNet creates the logical switch after the control session is established.

Listing the Physical Switch

```
cumulus@switch12:~$ vtep-ctl list-ps
sw12
```

Listing the Logical Switch

```
cumulus@switch12:~$ vtep-ctl list-ls
mn-6c9826da-6655-4fe3-a826-4dcba6477d2d
```

Listing Local or Remote MAC Addresses

These commands show the MAC addresses learned from the connected port bound to the logical switch, or the MAC addresses advertised from MidoNet. The *unknown-dst* entries are installed to satisfy the ethernet flooding of unknown unicast, and important for learning.

```
cumulus@switch12:~$ vtep-ctl list-local-macs mn-6c9826da-6655-4fe3-a826-
4dcba6477d2d
ucast-mac-local
64:ae:0c:32:f1:41 -> vxlan_over_ipv4/10.111.1.2
```

```

mcast-mac-local
    unknown-dst -> vxlan_over_ipv4/10.111.1.2

cumulus@switch12:~$ vtep-ctl list-remote-macs mn-6c9826da-6655-4fe3-a826-
4dcba6477d2d
ucast-mac-remote
    fa:16:3e:14:04:2e -> vxlan_over_ipv4/10.111.0.182
mcast-mac-remote
    unknown-dst -> vxlan_over_ipv4/10.111.0.182oh

```

## Getting Open Vswitch Database (OVSDB) Data

The `ovsdb-client dump` command is large, but shows all of the information and tables that are used in communication between the OVS client and server.

```

cumulus@switch12:~$ ovsdb-client dump
Arp_Sources_Local table
_uuid locator src_mac
-----
Arp_Sources_Remote table
_uuid locator src_mac
-----
Global table
_uuid managers switches
-----
76672d6a-2740-4c8d-9618-9e8dfb4b0bd7 [ ] [6d459554-0c75-4170-bb3d-
117eb4ce1f4d]
Logical_Binding_Stats table
_uuid bytes_from_local bytes_to_local packets_from_local packets_to_local
-----
d2e378b4-61c1-4daf-9aec-a7fd352d3193 5782569 1658250 21687 14589
Logical_Router table
_uuid description name static_routes switch_binding
-----
Logical_Switch table
_uuid description name tunnel_key
-----
44d162dc-0372-4749-a802-5b153c7120ec "" "mn-6c9826da-6655-4fe3-a826-
4dcba6477d2d" 10006
Manager table
_uuid inactivity_probe is_connected max_backoff other_config status target

```

```
-----  
Mcast_Macs_Local table  
MAC _uuid ipaddr locator_set logical_switch  
-----  
-----  
unknown-dst 25eaf29a-c540-46e3-8806-3892070a2de5 "" 7a4c000a-244e-4b37-8f25-  
fd816c1a80dc 44d162dc-0372-4749-a802-5b153c7120ec  
Mcast_Macs_Remote table  
MAC _uuid ipaddr locator_set logical_switch  
-----  
-----  
unknown-dst b122b897-5746-449e-83ba-fa571a64b374 "" 6c04d477-18d0-41df-8d52-  
dc7b17845ebe 44d162dc-0372-4749-a802-5b153c7120ec  
Physical_Locator table  
_uuid dst_ip encapsulation_type  
-----  
2fcf8b7e-e084-4bcb-b668-755ae7ac0bfb "10.111.0.182" "vxlan_over_ipv4"  
3f78dbb0-9695-42ef-a31f-aaaf525147f1 "10.111.1.2" "vxlan_over_ipv4"  
Physical_Locator_Set table  
_uuid locators  
-----  
6c04d477-18d0-41df-8d52-dc7b17845ebe [2fcf8b7e-e084-4bcb-b668-755ae7ac0bfb]  
7a4c000a-244e-4b37-8f25-fd816c1a80dc [3f78dbb0-9695-42ef-a31f-aaaf525147f1]  
Physical_Port table  
_uuid description name port_fault_status vlan_bindings vlan_stats  
-----  
-----  
bf69fcbb-36b3-4dbc-a90d-fc7412e57076 "swp1" "swp1" [] {} {}  
bf38137d-3a14-454e-8df0-9c56e4b4e640 "swp10" "swp10" [] {} {}  
69585fff-4360-4177-901d-8360ade5391b "swp11s0" "swp11s0" [] {0=44d162dc-  
0372-4749-a802-5b153c7120ec} {0=d2e378b4-61c1-4daf-9aec-a7fd352d3193}  
2a2d04fa-7190-41fe-8cee-318fcbafb2ea "swp11s1" "swp11s1" [] {} {}  
684f99d5-426c-45c8-b964-211489f45599 "swp11s2" "swp11s2" [] {} {}  
47cc66fb-eef8a-4a9b-a497-1844b89f7d32 "swp11s3" "swp11s3" [] {} {}  
5be3a052-be0f-4258-94cb-5e8be9afb896 "swp12" "swp12" [] {} {}  
631b19bd-3022-4353-bb2d-f498b0c1cb17 "swp13" "swp13" [] {} {}  
3001c904-b152-4dc4-9d8e-718f24ffa439 "swp14" "swp14" [] {} {}  
a6f8a88a-3877-4f81-b9b4-d75394a09d2c "swp15" "swp15" [] {} {}  
7cb681f4-2206-4c70-85b7-23b60963cd21 "swp16" "swp16" [] {} {}  
3943fb6a-0b49-4806-a014-2bcd4d469537 "swp17" "swp17" [] {} {}  
109a9911-d6c7-4142-b6c9-7c985506abb4 "swp18" "swp18" [] {} {}  
93b85c31-be38-4384-8b7a-9696764f9ba9 "swp19" "swp19" [] {} {}  
bcfb2920-6676-494c-9dcb-b474123b7e59 "swp2" "swp2" [] {} {}  
4223559a-dal1c-4c34-b8bf-bff7ced376ad "swp20" "swp20" [] {} {}
```

```

6bbccda8-d7e5-4b19-b978-4ec7f5b868e0 "swp21" "swp21" [] {} {}
c6876886-8386-4e34-a307-931909fca58f "swp22" "swp22" [] {} {}
c5a88dd6-d931-4b2c-9baa-a0abfb9d41f5 "swp23" "swp23" [] {} {}
124d1e01-a187-4427-819f-21de66e76f13 "swp24" "swp24" [] {} {}
55b49814-b5c5-405e-8e9f-898f3df4f872 "swp25" "swp25" [] {} {}
b2b2cd14-662d-45a5-87c1-277acbccdfffd "swp26" "swp26" [] {} {}
c35f55f5-8ec6-4fed-bef4-49801cd0934c "swp27" "swp27" [] {} {}
a44c5402-6218-4f09-bf1e-518f41a5546e "swp28" "swp28" [] {} {}
a9294152-2b32-4058-8796-23520fffb7379 "swp29" "swp29" [] {} {}
e0ee993a-8383-4701-a766-d425654dbb7f "swp3" "swp3" [] {} {}
d9db91a6-1c10-4154-9269-84877faa79b4 "swp30" "swp30" [] {} {}
b26ce4dd-b771-4d7b-8647-41fa97aa40e3 "swp31" "swp31" [] {} {}
652c6cd1-0823-4585-bb78-658e6ca2abfc "swp32" "swp32" [] {} {}
5b15372b-89f0-4e14-a50b-b6c6f937d33d "swp4" "swp4" [] {} {}
e00741f1-ba34-47c5-ae23-9269c5d1a871 "swp5" "swp5" [] {} {}
7096abaf-eebf-4ee3-b0cc-276224bc3e71 "swp6" "swp6" [] {} {}
439afb62-067e-4bbe-a0d9-ee33a23d2a9c "swp7" "swp7" [] {} {}
54f6c9df-01a1-4d96-9dcf-3035a33ffb3e "swp8" "swp8" [] {} {}
c85ed6cd-a7d4-4016-b3e9-34df592072eb "swp9s0" "swp9s0" [] {} {}
cf382ed6-60d3-43f5-8586-81f4f0f2fb28 "swp9s1" "swp9s1" [] {} {}
c32a9ff9-fd11-4399-815f-806322f26ff5 "swp9s2" "swp9s2" [] {} {}
9a7e42c4-228f-4b55-b972-7c3b8352c27d "swp9s3" "swp9s3" [] {} {}

Physical_Switch table
_uuid description management_ips name ports switch_fault_status tunnel_ips
tunnels
-----
```

```
-----
6d459554-0c75-4170-bb3d-117eb4ce1f4d "sw12" ["10.50.20.22"] "sw12"
[109a9911-d6c7-4142-b6c9-7c985506abb4, 124d1e01-a187-4427-819f-
21de66e76f13, 2a2d04fa-7190-41fe-8cee-318fcbafb2ea, 3001c904-b152-4dc4-9d8e-
718f24ffa439, 3943fb6a-0b49-4806-a014-2bcd4d469537, 4223559a-dalc-4c34-b8bf-
bff7ced376ad, 439afb62-067e-4bbe-a0d9-ee33a23d2a9c, 47cc66fb-ef8a-4a9b-a497-
1844b89f7d32, 54f6c9df-01a1-4d96-9dcf-3035a33ffb3e, 55b49814-b5c5-405e-8e9f-
898f3df4f872, 5b15372b-89f0-4e14-a50b-b6c6f937d33d, 5be3a052-be0f-4258-94cb-
5e8be9afb896, 631b19bd-3022-4353-bb2d-f498b0c1cb17, 652c6cd1-0823-4585-bb78-
658e6ca2abfc, 684f99d5-426c-45c8-b964-211489f45599, 69585fff-4360-4177-901d-
8360ade5391b, 6bccda8-d7e5-4b19-b978-4ec7f5b868e0, 7096abaf-eebf-4ee3-b0cc-
276224bc3e71, 7cb681f4-2206-4c70-85b7-23b60963cd21, 93b85c31-be38-4384-8b7a-
9696764f9ba9, 9a7e42c4-228f-4b55-b972-7c3b8352c27d, a44c5402-6218-4f09-bf1e-
518f41a5546e, a6f8a88a-3877-4f81-b9b4-d75394a09d2c, a9294152-2b32-4058-8796-
23520fffb7379, b26ce4dd-b771-4d7b-8647-41fa97aa40e3, b2b2cd14-662d-45a5-87c1-
277acbccdfdf, bcfb2920-6676-494c-9dc9-b474123b7e59, bf38137d-3a14-454e-8df0-
9c56e4b4e640, bf69fcbb-36b3-4dbc-a90d-fc7412e57076, c32a9ff9-fd11-4399-815f-
806322f26ff5, c35f55f5-8ec6-4fed-bef4-49801cd0934c, c5a88dd6-d931-4b2c-9baa-
a0abfb9d41f5, c6876886-8386-4e34-a307-931909fca58f, c85ed6cd-a7d4-4016-b3e9-
34df592072eb, cf382ed6-60d3-43f5-8586-81f4f0f2fb28, d9db91a6-1c10-4154-9269-
84877faa79b4, e00741f1-ba34-47c5-ae23-9269c5d1a871, e0ee993a-8383-4701-a766-
d425654dbb7f] [] ["10.111.1.2"] [062eaf89-9bd5-4132-8b6b-09db254325af]
```

## Tunnel table

```
_uuid bfd_config_local bfd_config_remote bfd_params bfd_status local remote
```

```
-----
062eaf89-9bd5-4132-8b6b-09db254325af {bfd_dst_ip="169.254.1.0",
bfd_dst_mac="00:23:20:00:00:01"} {} {} {} 3f78dbb0-9695-42ef-a31f-
aaaf525147f1 2fcf8b7e-e084-4bcb-b668-755ae7ac0bfb
```

## Ucast\_Macs\_Local table

```
MAC _uuid ipaddr locator logical_switch
```

```
-----
"64:ae:0c:32:f1:41" 47a83a7c-bd2d-4c02-9814-8222229c592f "" 3f78dbb0-9695-
42ef-a31f-aaaf525147f1 44d162dc-0372-4749-a802-5b153c7120ec
```

## Ucast\_Macs\_Remote table

```
MAC _uuid ipaddr locator logical_switch
```

```
-----
"fa:16:3e:14:04:2e" 65605488-9ee5-4c8e-93e5-7b1cc15cfcc7 "" 2fcf8b7e-e084-
4bcb-b668-755ae7ac0bfb 44d162dc-0372-4749-a802-5b153c7120ec
```

## Lightweight Network Virtualization - LNV

Lightweight Network Virtualization (LNV) is a technique for deploying VXLANS (see page 269) without a central controller on bare metal switches. This solution requires no external controller or software suite; it runs the VXLAN service and registration daemons on Cumulus Linux itself. The data path between bridge entities is established on top of a layer 3 fabric by means of a simple service node coupled with traditional MAC address learning.

To see an example of a full solution before reading the following background information, [please read this chapter](#) (see page 363).



LNV is a lightweight controller option. Please [contact Cumulus Networks](#) with your scale requirements so we can make sure this is the right fit for you. There are also other controller options that can work on Cumulus Linux.

## Contents

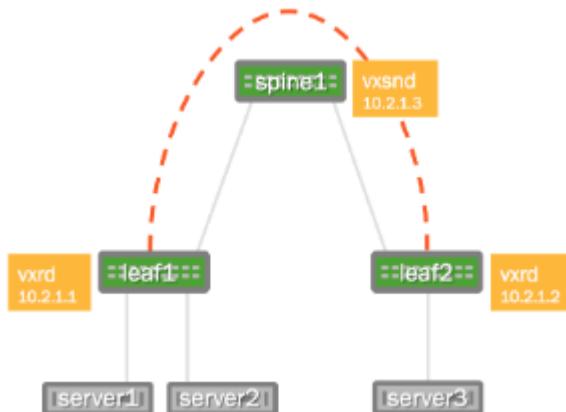
(Click to expand)

- [Contents \(see page 303\)](#)
- [Understanding LNV Concepts \(see page 304\)](#)
  - [Acquiring the Forwarding Database at the Service Node \(see page 304\)](#)
  - [MAC Learning and Flooding \(see page 304\)](#)
  - [Handling BUM Traffic \(see page 305\)](#)
- [Requirements \(see page 305\)](#)
  - [Hardware Requirements \(see page 306\)](#)
  - [Configuration Requirements \(see page 306\)](#)
  - [Installing the LNV Packages \(see page 306\)](#)
- [Sample LNV Configuration \(see page 306\)](#)
  - [Network Connectivity \(see page 307\)](#)
  - [Layer 3 IP Addressing \(see page 307\)](#)
  - [Layer 3 Fabric \(see page 308\)](#)
  - [Host Configuration \(see page 310\)](#)
- [Configuring the VLAN to VXLAN Mapping \(see page 311\)](#)
- [Verifying the VLAN to VXLAN Mapping \(see page 312\)](#)
- [Enabling and Managing Service Node and Registration Daemons \(see page 313\)](#)
  - [Enabling the Service Node Daemon \(see page 313\)](#)
  - [Enabling the Registration Daemon \(see page 313\)](#)
  - [Checking the Daemon Status \(see page 314\)](#)
- [Configuring the Registration Node \(see page 315\)](#)
- [Configuring the Service Node \(see page 317\)](#)
- [Verification and Troubleshooting \(see page 318\)](#)
  - [Verifying the Registration Node Daemon \(see page 318\)](#)

- Verifying the Service Node Daemon (see page 319)
- Verifying Traffic Flow and Checking Counters (see page 320)
- Pinging to Test Connectivity (see page 320)
- Troubleshooting with MAC Addresses (see page 322)
- Checking the Service Node Configuration (see page 322)
- Creating a Layer 3 Gateway (see page 323)
- Advanced LNV Usage (see page 323)
  - Scaling LNV by Load Balancing with Anycast (see page 323)
- Additional Resources (see page 329)
- See Also (see page 330)

## ***Understanding LNV Concepts***

To best describe this feature, consider the following example deployment:



The two switches running Cumulus Linux, called leaf1 and leaf2, each have a bridge configured. These two bridges contain the physical switch port interfaces connecting to the servers as well as the logical VXLAN interface associated with the bridge. By creating a logical VXLAN interface on both leaf switches, the switches become *VTEPs* (virtual tunnel end points). The IP address associated with this VTEP is most commonly configured as its loopback address — in the image above, the loopback address is 10.2.1.1 for leaf1 and 10.2.1.2 for leaf2.

## ***Acquiring the Forwarding Database at the Service Node***

In order to connect these two VXLANs together and forward BUM (Broadcast, Unknown-unicast, Multicast) packets to members of a VXLAN, the service node needs to acquire the addresses of all the VTEPs for every VXLAN it serves. The service node daemon does this through a registration daemon running on each leaf switch that contains a VTEP participating in LNV. The registration process informs the service node of all the VXLANs to which the switch belongs.

## ***MAC Learning and Flooding***

With LNV, as with traditional bridging of physical LANs or VLANs, a bridge automatically learns the location of hosts as a side effect of receiving packets on a port.

For example, when server1 sends an L2 packet to server3, leaf2 learns that server1's MAC address is located on that particular VXLAN, and the VXLAN interface learns that the IP address of the VTEP for server1 is 10.2.1.1. So when server3 sends a packet to server1, the bridge on leaf2 forwards the packet out of the port to the VXLAN interface and the VXLAN interface sends it, encapsulated in a UDP packet, to the address 10.2.1.1.

But what if server3 sends a packet to some address that has yet to send it a packet (server2, for example)? In this case, the VXLAN interface sends the packet to the service node, which sends a copy to every other VTEP that belongs to the same VXLAN.

## Handling BUM Traffic

Cumulus Linux has two ways of handling BUM (Broadcast Unknown-unicast and Multicast) traffic:

- Head end replication
- Service node replication

Head end replication is enabled by default in Cumulus Linux.



You cannot have both service node and head end replication configured simultaneously, as this causes the BUM traffic to be duplicated — both the source VTEP and the service node sending their own copy of each packet to every remote VTEP.

## Head End Replication

The Tomahawk, Trident II+ and Trident II chipsets are capable of head end replication — the ability to generate all the BUM traffic in hardware. The most scalable solution available with LNV is to have each VTEP (top of rack switch) generate all of its own BUM traffic rather than relying on an external service node.

Cumulus Linux verified support for up to 128 VTEPs with head end replication.

To disable head end replication, edit `/etc/vxrd.conf` and set `head_rep` to `False`.

## Service Node Replication

Cumulus Linux also supports service node replication for VXLAN BUM packets. This is useful with LNV if you have more than 128 VTEPs. However, it is not recommended because it forces the spine switches running the `vxsnd` (service node daemon) to replicate the packets in software instead of in hardware, unlike head end replication. If you're not using a controller but have more than 128 VTEPs, contact [Cumulus Networks](#).

To enable service node replication:

1. Disable head end replication; set `head_rep` to `False` in `/etc/vxrd.conf`.
2. Edit `/etc/network/interfaces` and configure a service node IP address for VXLAN interfaces using `vxrd-svcnode-ip <>`.
3. Edit `/etc/vxsnd.conf`, and configure the following:
  - Set the same service node IP address that you did in the previous step:  
`svcnode_ip = <>`
  - To forward VXLAN data traffic, set the following variable to `True`:  
`enable_vxlan_listen = true`

## Requirements

### Hardware Requirements

- Switches with a Tomahawk, Trident II+ or Trident II chipset running Cumulus Linux 2.5.4 or later. Please refer to the Cumulus Networks [hardware compatibility list](#) for a list of supported switch models.

### Configuration Requirements

- The VXLAN has an associated **VXLAN Network Identifier** (VNI), also interchangeably called a VXLAN ID.
- The VNI should not be 0 or 16777215, as these two numbers are reserved values under Cumulus Linux.
- The VXLAN link and physical interfaces are added to the bridge to create the association between the port, VLAN and VXLAN instance.
- Each bridge on the switch has only one VXLAN interface. Cumulus Linux does not support more than one VXLAN link in a bridge; however, a switch can have multiple bridges.
- Only use bridges in [traditional mode \(see page 214\)](#); [VLAN-aware bridges \(see page 235\)](#) are not supported with VXLAN at this time.
- An SVI (Switch VLAN Interface) or L3 address on the bridge is not supported. For example, you can't ping from the leaf1 SVI to the leaf2 SVI via the VXLAN tunnel; you would need to use server1 and server2 to verify. See [Creating a Layer 3 Gateway \(see page 322\)](#) below for more information.

### Installing the LNV Packages

The LNV packages are not installed automatically if you upgrade Cumulus Linux. You can install LNV in one of two ways:

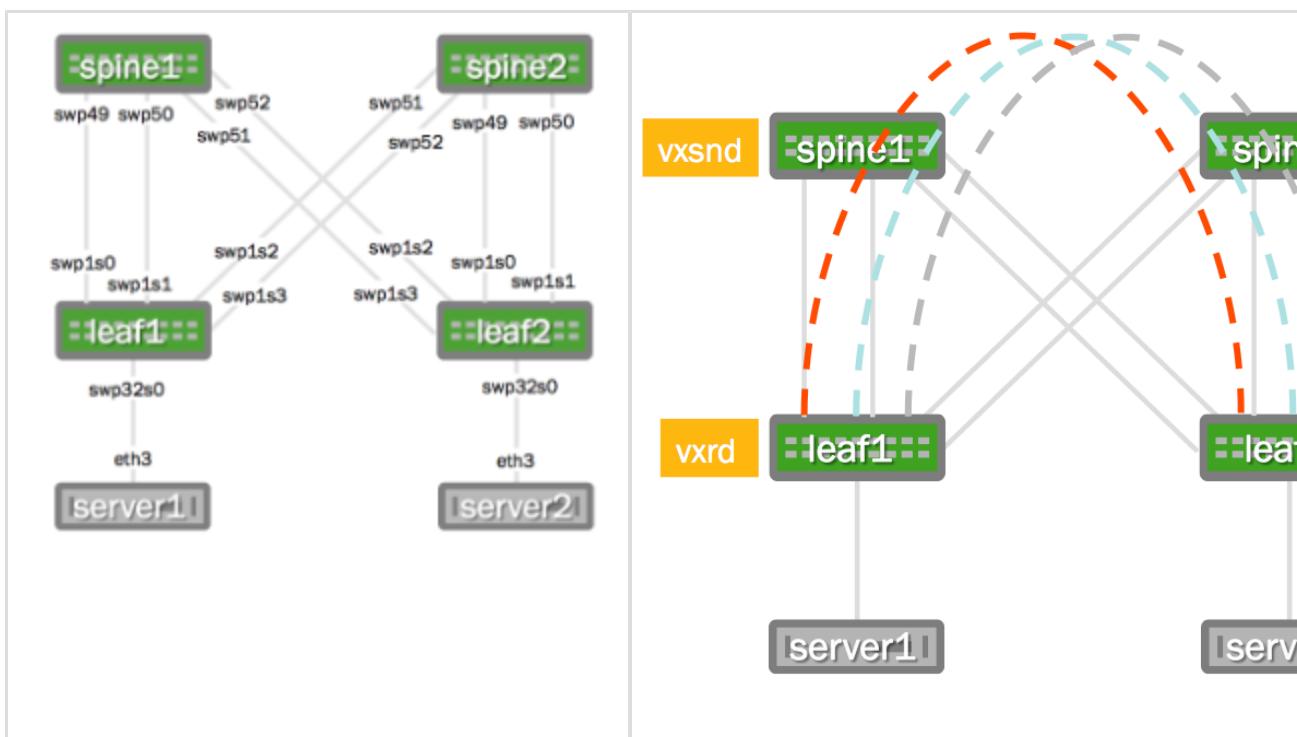
- Do a [binary image install \(see page 26\)](#) of Cumulus Linux, using [ONIE](#)
- Install the LNV packages for the registration and service node daemons using `apt-get install vxflid-vxrd` and/or `apt-get install vxflid-vxsnd`, depending upon how you intend to use LNV

### Sample LNV Configuration

The following images illustrate the configuration that is referenced throughout this chapter.

Physical Cabling Diagram

Network Virtualization Diagram



Want to try out configuring LNV and don't have a Cumulus Linux switch? Sign up to use the [Cumulus Workbench](#), which has this exact topology.

## Network Connectivity

There must be full network connectivity before you can configure LNV. The layer 3 IP addressing information as well as the OSPF configuration (`/etc/quagga/Quagga.conf`) below is provided to make the LNV example easier to understand.



OSPF is not a requirement for LNV, LNV just requires L3 connectivity. With Cumulus Linux this can be achieved with static routes, OSPF or BGP.

## Layer 3 IP Addressing

Here is the configuration for the IP addressing information used in this example.

**spine1:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
    address 10.2.1.3/32

auto eth0
iface eth0 inet dhcp
```

**spine2:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
    address 10.2.1.4/32

auto eth0
iface eth0 inet dhcp
```

```

auto swp49
iface swp49
  address 10.1.1.2/30

auto swp50
iface swp50
  address 10.1.1.6/30

auto swp51
iface swp51
  address 10.1.1.50/30

auto swp52
iface swp52
  address 10.1.1.54/30
  
```

```

auto swp49
iface swp49
  address 10.1.1.18/30

auto swp50
iface swp50
  address 10.1.1.22/30

auto swp51
iface swp51
  address 10.1.1.34/30

auto swp52
iface swp52
  address 10.1.1.38/30
  
```

**leaf1:** /etc/network/interfaces

```

auto lo
iface lo inet loopback
  address 10.2.1.1/32

auto eth0
iface eth0 inet dhcp

auto swp1s0
iface swp1s0
  address 10.1.1.1/30

auto swp1s1
iface swp1s1
  address 10.1.1.5/30

auto swp1s2
iface swp1s2
  address 10.1.1.33/30

auto swp1s3
iface swp1s3
  address 10.1.1.37/30
  
```

**leaf2:** /etc/network/interfaces

```

auto lo
iface lo inet loopback
  address 10.2.1.2/32

auto eth0
iface eth0 inet dhcp

auto swp1s0
iface swp1s0
  address 10.1.1.17/30

auto swp1s1
iface swp1s1
  address 10.1.1.21/30

auto swp1s2
iface swp1s2
  address 10.1.1.49/30

auto swp1s3
iface swp1s3
  address 10.1.1.53/30
  
```

## Layer 3 Fabric

The service nodes and registration nodes must all be routable between each other. The L3 fabric on Cumulus Linux can either be [BGP](#) (see page 419) or [OSPF](#) (see page 406). In this example, OSPF is used to demonstrate full reachability. Expand the Quagga configurations below.

Quagga configuration using OSPF:

### spine1

```

interface lo
 ip ospf area 0.0.0.0
interface swp49
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp50
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp51
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp52
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
!
!
!
!
router-id 10.2.1.3
router ospf
 ospf router-id 10.2.1.3

```

### spine2

```

interface lo
 ip ospf area 0.0.0.0
interface swp49
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp50
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp51
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
interface swp52
 ip ospf network point-to-point
 ip ospf area 0.0.0.0
!
!
!
!
!
router-id 10.2.1.4
router ospf
 ospf router-id 10.2.1.4

```

### leaf1

```

interface lo
 ip ospf area 0.0.0.0
interface swp1s0
 ip ospf network point-to-
point
 ip ospf area 0.0.0.0
!
interface swp1s1
 ip ospf network point-to-
point
 ip ospf area 0.0.0.0

```

### leaf2

```

interface lo
 ip ospf area 0.0.0.0
interface swp1s0
 ip ospf network point-to-
point
 ip ospf area 0.0.0.0
!
interface swp1s1
 ip ospf network point-to-
point
 ip ospf area 0.0.0.0

```

```
!
interface swp1s2
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s3
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
!
!
!
!
!
router-id 10.2.1.1
router ospf
  ospf router-id 10.2.1.1
```

```
!
interface swp1s2
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s3
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
!
!
!
!
!
router-id 10.2.1.2
router ospf
  ospf router-id 10.2.1.2
```

## Host Configuration

In this example, the servers are running Ubuntu 14.04. There needs to be a trunk mapped from server1 and server2 to the respective switch. In Ubuntu this is done with subinterfaces. You can expand the configurations below.

server1

```
auto eth3.10
iface eth3.10 inet
static
  address 10.10.10.1/24

auto eth3.20
iface eth3.20 inet
static
  address 10.10.20.1/24

auto eth3.30
iface eth3.30 inet
static
  address 10.10.30.1/24
```

server2

```
auto eth3.10
iface eth3.10 inet
static
  address 10.10.10.2/24

auto eth3.20
iface eth3.20 inet
static
  address 10.10.20.2/24

auto eth3.30
iface eth3.30 inet
static
  address 10.10.30.2/24
```

On Ubuntu it is more reliable to use `ifup` and `if down` to bring the interfaces up and down individually, rather than restarting networking entirely (that is, there is no equivalent to `if reload` like there is in Cumulus Linux):

```
cumulus@server1:~$ sudo ifup eth3.10
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan
/config
Added VLAN with VID == 10 to IF -:eth3:-
cumulus@server1:~$ sudo ifup eth3.20
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan
/config
Added VLAN with VID == 20 to IF -:eth3:-
cumulus@server1:~$ sudo ifup eth3.30
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan
/config
Added VLAN with VID == 30 to IF -:eth3:-
```

## Configuring the VLAN to VXLAN Mapping

Configure the VLANS and associated VXLANs. In this example, there are 3 VLANS and 3 VXLAN IDs (VNIs). VLANS 10, 20 and 30 are used and associated with VNIs 10, 2000 and 30 respectively. The loopback address, used as the `vxlan-local-tunnelip`, is the only difference between leaf1 and leaf2 for this demonstration.

For leaf1:

```
cumulus@leaf1$ sudo nano /etc
/network/interfaces
```

Add the following to the loopback stanza

```
auto lo
iface lo
    vxrd-src-ip 10.2.1.1
    vxrd-svcnode-ip 10.2.1.3
```

Now append the following for the VXLAN configuration itself:

```
leaf1: /etc/network/interfaces

auto vni-10
iface vni-10
    vxlan-id 10
    vxlan-local-tunnelip
    10.2.1.1

auto vni-2000
iface vni-2000
    vxlan-id 2000
    vxlan-local-tunnelip 10.2.1.1
```

For leaf2:

```
cumulus@leaf2$ sudo nano /etc
/network/interfaces
```

Add the following to the loopback stanza

```
auto lo
iface lo
    vxrd-src-ip 10.2.1.2
    vxrd-svcnode-ip 10.2.1.3
```

Now append the following for the VXLAN configuration itself:

```
leaf2: /etc/network/interfaces

auto vni-10
iface vni-10
    vxlan-id 10
    vxlan-local-tunnelip
    10.2.1.2

auto vni-2000
iface vni-2000
    vxlan-id 2000
    vxlan-local-tunnelip 10.2.1.2
```

```

auto vni-30
iface vni-30
  vxlan-id 30
  vxlan-local-tunnelip 10.2.1.1

auto br-10
iface br-10
  bridge-ports swp32s0.10 vni-
  10

auto br-20
iface br-20
  bridge-ports swp32s0.20 vni-
  2000

auto br-30
iface br-30
  bridge-ports swp32s0.30 vni-
  30
  
```

To bring up the bridges and VNIs, use the `ifreload` command:

```
cumulus@leaf1$ sudo ifreload -a
```

```

auto vni-30
iface vni-30
  vxlan-id 30
  vxlan-local-tunnelip 10.2.1.2

auto br-10
iface br-10
  bridge-ports swp32s0.10 vni-
  10

auto br-20
iface br-20
  bridge-ports swp32s0.20 vni-
  2000

auto br-30
iface br-30
  bridge-ports swp32s0.30 vni-
  30
  
```

To bring up the bridges and VNIs, use the `ifreload` command:

```
cumulus@leaf2$ sudo ifreload -a
```

- i** Why is br-20 not vni-20? For example, why not tie VLAN 20 to VNI 20, or why was 2000 used? VXLANs and VLANs do not need to be the same number. This was done on purpose to highlight this fact. However if you are using fewer than 4096 VLANs, there is no reason not to make it easy and correlate VLANs to VXLANs. It is completely up to you.

## Verifying the VLAN to VXLAN Mapping

Use the `brctl show` command to see the physical and logical interfaces associated with that bridge:

```

cumulus@leaf1:~$ brctl show
bridge name      bridge id      STP enabled      interfaces
br-10           8000.443839008404    no            swp32s0.10
                                         vni-10
br-20           8000.443839008404    no            swp32s0.20
                                         vni-2000
br-30           8000.443839008404    no            swp32s0.30
                                         vni-30
  
```

As with any logical interfaces on Linux, the name does not matter (other than a 15-character limit). To verify the associated VNI for the logical name, use the `ip -d link show` command:

```
cumulus@leaf1$ ip -d link show vni-10
43: vni-10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    master br-10 state UNKNOWN mode DEFAULT
        link/ether 02:ec:ec:bd:7f:c6 brd ff:ff:ff:ff:ff:ff
        vxlan id 10 srcport 32768 61000 dstport 4789 ageing 300
        bridge_slave
```

The `vxlan id 10` indicates the VXLAN ID/VNI is indeed 10 as the logical name suggests.

## **Enabling and Managing Service Node and Registration Daemons**

Every VTEP must run the registration daemon (`vxrd`). Typically, every leaf switch acts as a VTEP. A minimum of 1 switch (a switch not already acting as a VTEP) must run the service node daemon (`vxsnd`). The instructions for enabling these daemons follows.

### **Enabling the Service Node Daemon**

The service node daemon (`vxsnd`) is included in the Cumulus Linux repository as `vxfld-vxsnd`. The service node daemon can run on any switch running Cumulus Linux as long as that switch is not also a VXLAN VTEP. In this example, enable the service node only on the spine1 switch.



Do not run `vxsnd` on a switch that is already acting as a VTEP.

Edit the `/etc/default/vxsnd` configuration file:

```
cumulus@spine1$ sudo nano /etc/default/vxsnd
```

Change the `vxsnd` file by changing `no` to `yes`:

```
START=yes
```

Save and quit the text editor and reboot the `vxsnd` daemon:

```
cumulus@spine1$ sudo systemctl restart vxsnd.service
```

### **Enabling the Registration Daemon**

The registration daemon (`vxrd`) is included in the Cumulus Linux package as `vxfld-vxrd`. The registration daemon must run on each VTEP participating in LNV, so you must enable it on every TOR (leaf) switch acting as a VTEP.

Edit the `/etc/default/vxrd` configuration file on leaf1:

```
cumulus@leaf1$ sudo nano /etc/default/vxrd
```

Change the `vxrd` file by changing `no` to `yes`:

```
START=yes
```

Save and quit the text editor and reboot the `vxrd` daemon:

```
cumulus@leaf1$ sudo systemctl restart vxrd.service
```

Open the `vxrd` configuration file on leaf2 with the following commands:

```
cumulus@leaf2$ sudo nano /etc/default/vxrd
```

Change the `vxsnd` file by changing `no` to `yes`:

```
START=yes
```

Save and quit the text editor and reboot the `vxrd` daemon:

```
cumulus@leaf1$ sudo systemctl restart vxrd.service
```

## Checking the Daemon Status

To determine if the daemon is running, use the `sudo systemctl <daemon name>.service status` command.

For the service node daemon:

```
cumulus@spine1$ sudo systemctl status vxsnd.service
vxsnd.service - Lightweight Network Virt Discovery Svc and Replicator
   Loaded: loaded (/lib/systemd/system/vxsnd.service; enabled)
     Active: active (running) since Wed 2016-05-11 11:42:55 UTC; 10min
            ago
       Main PID: 774 (vxsnd)
          CGroup: /system.slice/vxsnd.service
                    774 /usr/bin/python /usr/bin/vxsnd

May 11 11:42:55 cumulus vxsnd[774]: INFO: Starting (pid 774) ...
```

For the registration daemon:

```
cumulus@leaf1$ sudo systemctl status vxrd.service
vxrd.service - Lightweight Network Virtualization Peer Discovery
Daemon
   Loaded: loaded (/lib/systemd/system/vxrd.service; enabled)
   Active: active (running) since Wed 2016-05-11 11:42:55 UTC; 10min
ago
     Main PID: 929 (vxrd)
        CGroup: /system.slice/vxrd.service
                  929 /usr/bin/python /usr/bin/vxrd

May 11 11:42:55 cumulus vxrd[929]: INFO: Starting (pid 929) ...
```

## Configuring the Registration Node

The registration node was configured earlier in `/etc/network/interfaces` in the [VXLAN mapping](#) (see [page 311](#)) section above; no additional configuration is typically needed. However, if you need to modify the registration node configuration, edit `/etc/vxrd.conf`.

Alternate location for configuration and additional knobs for the registration node are found in `/etc/vxrd.conf`

```
cumulus@leaf1$ sudo nano /etc/vxrd.conf
```

Then edit the `svcnod_ip` variable:

```
svcnod_ip = 10.2.1.3
```

Then perform the same on leaf2:

```
cumulus@leaf2$ sudo nano /etc/vxrd.conf
```

And again edit the `svcnod_ip` variable:

```
svcnod_ip = 10.2.1.3
```

Restart the registration node daemon for the change to take effect:

```
cumulus@leaf1$ sudo systemctl restart vxrd.service
```

Restart the daemon on leaf2:

```
cumulus@leaf2$ sudo systemctl restart vxrd.service
```

The complete list of options you can configure is listed below:

Name	Description	Default
loglevel	The log level, which can be DEBUG, INFO, WARNING, ERROR, CRITICAL.	INFO
logdest	The destination for log messages. It can be a file name, <code>stdout</code> or <code>syslog</code> .	syslog
logfilesize	Log file size in bytes. Used when <code>logdest</code> is a file name.	512000
logbackupcount	Maximum number of log files stored on the disk. Used when <code>logdest</code> is a file name.	14
pidfile	The PIF file location for the <code>vxrd</code> daemon.	/var/run/vxrd.pid
udsfile	The file name for the Unix domain socket used for management.	/var/run/vxrd.sock
vxfld_port	The UDP port used for VXLAN control messages.	10001
svcnod_ip	The address to which registration daemons send control messages for registration and/or BUM packets for replication. This can also be configured under <code>/etc/network/interfaces</code> with the <code>vxrd-svcnode-ip</code> keyword.	
holdtime	Hold time (in seconds) for soft state, which is how long the service node waits before ageing out an IP address for a VNI. The <code>vxrd</code> includes this in the register messages it sends to a <code>vxsnd</code> .	90 seconds
src_ip	Local IP address to bind to for receiving control traffic from the service node daemon.	
refresh_rate	Number of times to refresh within the hold time. The higher this number, the more lost UDP refresh messages can be tolerated.	3 seconds
config_check_rate	The number of seconds to poll the system for current VXLAN membership.	5 seconds
head_rep	Enables self replication. Instead of using the service node to replicate BUM packets, it will be done in hardware on the VTEP switch.	true





Use *1, yes, true* or *on* for True for each relevant option. Use *0, no, false* or *off* for False.

## Configuring the Service Node

To configure the service node daemon, edit the `/etc/vxsnd.conf` configuration file.



For the example configuration, default values are used, except for the `svcnod_ip` field.

```
cumulus@spine1$ sudo nano /etc/vxsnd.conf
```

The address field is set to the loopback address of the switch running the `vxsnd` dameon.

```
svcnod_ip = 10.2.1.3
```

Restart the service node daemon for the change to take effect:

```
cumulus@spine1$ sudo systemctl restart vxsnd.service
```

The complete list of options you can configure is listed below:

Name	Description	Default
loglevel	The log level, which can be DEBUG, INFO, WARNING, ERROR, CRITICAL.	INFO
logdest	Destination for log messages. It can be a file name, <code>stdout</code> or <code>syslog</code> .	syslog
logfilesize	The log file size in bytes. Used when <code>logdest</code> is a file name.	512000
logbackupcount	Maximum number of log files stored on disk. Used when <code>logdest</code> is a file name.	14
pidfile	The PID file location for the <code>vxrd</code> daemon.	/var/run/ /vxrd. pid
udsfile	The file name for the Unix domain socket used for management.	/var/run/ /vxrd. sock
vxld_port	The UDP port used for VXLAN control messages.	10001

Name	Description	Default
svcnod_ip	This is the address to which registration daemons send control messages for registration and/or BUM packets for replication.	0.0.0.0
holdtime	Holddate (in seconds) for soft state. It is used when sending a register message to peers in response to learning a <vn, addr> from a VXLAN data packet.	90
src_ip	Local IP address to bind to for receiving inter-vxsn control traffic.	0.0.0.0
svcnod_peers	Space-separated list of IP addresses with which the <b>vxsn</b> shares its state.	
enable_vxlan_listen	When set to true, the service node listens for VXLAN data traffic.	true
install_svcnode_ip	When set to true, the <b>sdn_peer_address</b> gets installed on the loopback interface. It gets withdrawn when the <b>vxsn</b> is not in service. If set to true, you must define the <b>sdn_peer_address</b> configuration variable.	false
age_check	Number of seconds to wait before checking the database to age out stale entries.	90 seconds



Use *1, yes, true* or *on* for True for each relevant option. Use *0, no, false* or *off* for False.

## Verification and Troubleshooting

### Verifying the Registration Node Daemon

Use the **vxrdctl vxlans** command to see the configured VNIs, the local address being used to source the VXLAN tunnel and the service node being used.

```
cumulus@leaf1$ vxrdctl vxlans
VNI      Local Addr      Svc
Node
===
=====
 10      10.2.1.1
10.2.1.3
 30      10.2.1.1
10.2.1.3
2000     10.2.1.1
10.2.1.3
```

```
cumulus@leaf2$ vxrdctl vxlans
VNI      Local Addr      Svc
Node
===
=====
 10      10.2.1.2
10.2.1.3
 30      10.2.1.2
10.2.1.3
2000     10.2.1.2
10.2.1.3
```

Use the `vxrdctl peers` command to see configured VNIs and all VTEPs (leaf switches) within the network that have them configured.

```
cumulus@leaf1$ vxrdctl peers
VNI      Peer Addrs
==      =====
10       10.2.1.1,
10.2.1.2
30       10.2.1.1,
10.2.1.2
2000     10.2.1.1,
10.2.1.2
```

```
cumulus@leaf2$ vxrdctl peers
VNI      Peer Addrs
==      =====
10       10.2.1.1,
10.2.1.2
30       10.2.1.1,
10.2.1.2
2000     10.2.1.1,
10.2.1.2
```



When head end replication mode is disabled, the command won't work.

Use the `vxrdctl peers` command to see the other VTEPs (leaf switches) and what VNIs are associated with them. This does not show anything unless you enabled head end replication mode by setting the `head_rep` option to *True*. Otherwise, replication is done by the service node.

```
cumulus@leaf2$ vxrdctl peers
Head-end replication is turned off on this device.
This command will not provide any output
```

## Verifying the Service Node Daemon

Use the `vxsndctl fdb` command to verify which VNIs belong to which VTEP (leaf switches).

```
cumulus@spine1$ vxsndctl fdb
VNI      Address      Ageout
==      =====      =====
10       10.2.1.1      82
10       10.2.1.2      77
30       10.2.1.1      82
30       10.2.1.2      77
2000     10.2.1.1      82
2000     10.2.1.2      77
```

## Verifying Traffic Flow and Checking Counters

VXLAN transit traffic information is stored in a flat file located at `/cumulus/switchd/run/stats/vxlan/all`.

```
cumulus@leaf1$ cat /cumulus/switchd/run/stats/vxlan/all
VNI                               : 10
Network In Octets                  : 1090
Network In Packets                 : 8
Network Out Octets                 : 1798
Network Out Packets                : 13
Total In Octets                    : 2818
Total In Packets                   : 27
Total Out Octets                   : 3144
Total Out Packets                  : 39
VN Interface                       : vni: 10, swp32s0.10
Total In Octets                    : 1728
Total In Packets                   : 19
Total Out Octets                   : 552
Total Out Packets                  : 18
VNI                               : 30
Network In Octets                  : 828
Network In Packets                 : 6
Network Out Octets                 : 1224
Network Out Packets                : 9
Total In Octets                    : 2374
Total In Packets                   : 23
Total Out Octets                   : 2300
Total Out Packets                  : 32
VN Interface                       : vni: 30, swp32s0.30
Total In Octets                    : 1546
Total In Packets                   : 17
Total Out Octets                   : 552
Total Out Packets                  : 17
VNI                               : 2000
Network In Octets                  : 676
Network In Packets                 : 5
Network Out Octets                 : 1072
Network Out Packets                : 8
Total In Octets                    : 2030
Total In Packets                   : 20
Total Out Octets                   : 2042
Total Out Packets                  : 30
VN Interface                       : vni: 2000, swp32s0.20
Total In Octets                    : 1354
Total In Packets                   : 15
Total Out Octets                   : 446
```

## Pinging to Test Connectivity

To test the connectivity across the VXLAN tunnel with an ICMP echo request (ping), make sure to ping from the server rather than the switch itself.



As mentioned above, SVIs (switch VLAN interfaces) are not supported when using VXLAN. That is, there cannot be an IP address on the bridge that also contains a VXLAN.

Following is the IP address information used in this example configuration.

VNI	server1	server2
10	10.10.10.1	10.10.10.2
2000	10.10.20.1	10.10.20.2
30	10.10.30.1	10.10.30.2

To test connectivity between VNI 10 connected servers by pinging from server1:

```
cumulus@server1:~$ ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=3.90 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=0.195 ms
^C
--- 10.10.10.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.195/1.432/3.900/1.745 ms
cumulus@server1:~$
```

The other VNIs were also tested and can be viewed in the expanded output below.

Test connectivity between VNI-2000 connected servers by pinging from server1:

```
cumulus@server1:~$ ping 10.10.20.2
PING 10.10.20.2 (10.10.20.2) 56(84) bytes of data.
64 bytes from 10.10.20.2: icmp_seq=1 ttl=64 time=1.81 ms
64 bytes from 10.10.20.2: icmp_seq=2 ttl=64 time=0.194 ms
64 bytes from 10.10.20.2: icmp_seq=3 ttl=64 time=0.206 ms
^C
--- 10.10.20.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.194/0.739/1.819/0.763 ms
```

Test connectivity between VNI-30 connected servers by pinging from server1:

```
cumulus@server1:~$ ping 10.10.30.2
PING 10.10.30.2 (10.10.30.2) 56(84) bytes of data.
64 bytes from 10.10.30.2: icmp_seq=1 ttl=64 time=1.85 ms
64 bytes from 10.10.30.2: icmp_seq=2 ttl=64 time=0.239 ms
64 bytes from 10.10.30.2: icmp_seq=3 ttl=64 time=0.185 ms
64 bytes from 10.10.30.2: icmp_seq=4 ttl=64 time=0.212 ms
^C
--- 10.10.30.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.185/0.622/1.853/0.711 ms
```

## Troubleshooting with MAC Addresses

Since there is no SVI, there is no way to ping from the server to the directly attached leaf (top of rack) switch without cabling the switch to itself (see [Creating a Layer 3 Gateway \(see page 322\)](#) below). The easiest way to see if the server can reach the leaf switch is to check the MAC address table of the leaf switch.

First, get the MAC address of the server:

```
cumulus@server1:~$ ip addr show eth3.10 | grep ether
link/ether 90:e2:ba:55:f0:85 brd ff:ff:ff:ff:ff:ff
```

Next, check the MAC address table of the leaf switch:

```
cumulus@leaf1$ brctl showmacs br-10
port name mac addr      vlan   is local?    ageing timer
vni-10  46:c6:57:fc:1f:54  0     yes        0.00
swp32s0.10 90:e2:ba:55:f0:85  0     no         75.87
vni-10  90:e2:ba:7e:a9:c1  0     no         75.87
swp32s0.10 ec:f4:bb:fc:67:a1  0     yes        0.00
```

90:e2:ba:55:f0:85 appears in the MAC address table, which indicates that connectivity is occurring between leaf1 and server1.

## Checking the Service Node Configuration

Use `ip -d link show` to verify the service node, VNI and administrative state of a particular logical VNI interface:

```
cumulus@leaf1$ ip -d link show vni-10
35: vni-10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
  master br-10 state UNKNOWN mode DEFAULT
    link/ether 46:c6:57:fc:1f:54 brd ff:ff:ff:ff:ff:ff
    vxlan id 10 remote 10.2.1.3 local 10.2.1.1 srcport 32768 dstport 4789 ageing 300 svcnode 10.2.1.3
      bridge_slave
```

## ***Creating a Layer 3 Gateway***

The Trident II ASIC has a limitation because of a restriction in the hardware, where an IP address cannot be configured on the same bridge of which a VXLAN is also a part. This limitation will not exist in future ASICs. For example, the Trident II+ has the [RIOT \(Routing In/Out of Tunnels\)](#) feature.

For the Trident II, this limitation means a physical cable must be attached from one port on leaf1 to another port on leaf1. One port is an L3 port while the other is a member of the bridge. For example, following the configuration above, in order for a layer 3 address to be used as the gateway for vni-10, you could configure the following on leaf1:

```
auto swp47
iface swp47
alias 12 port connected to swp48

auto swp48
iface swp48
alias gateway
address 10.10.10.3/24

auto vni-10
iface vni-10
vxlan-id 10
vxlan-local-tunnelip 10.2.1.1

auto br-10
iface br-10
bridge-ports swp47 swp32s0.10 vni-10
```

A loopback cable must be connected between swp47 and swp48 for this to work. This will be addressed in a future version of Cumulus Linux so a physical port does not need to be used for this purpose.

## ***Advanced LNV Usage***

### ***Scaling LNV by Load Balancing with Anycast***

The above configuration assumes a single service node. A single service node can quickly be overwhelmed by BUM traffic. To load balance BUM traffic across multiple service nodes, use [Anycast](#). Anycast enables BUM traffic to reach the topologically nearest service node rather than overwhelming a single service node.

### ***Enabling the Service Node Daemon on Additional Spine Switches***

In this example, spine1 already has the service node daemon enabled. Enable it on the spine2 switch with the following commands:

Edit the `/etc/default/vxsnd` configuration file:

```
cumulus@spine2$ sudo nano /etc/default/vxsnd
```

Change the **vxsnd** file by changing *no* to *yes*:

```
START=yes
```

Save and quit the text editor and reboot the **vxsnd** daemon:

```
cumulus@spine2$ sudo systemctl restart vx snd.service
```

## **Configuring the AnyCast Address on All Participating Service Nodes**

### **spine1**

Use a text editor to edit the network configuration:

```
cumulus@spine1$ sudo nano /etc/network/interfaces
```

Add the 10.10.10.10/32 address to the loopback address:

```
auto lo
iface lo inet loopback
    address 10.2.1.3/32
    address 10.10.10.10/32
```

Run ifreload -a:

```
cumulus@spine1$ sudo ifreload -a
```

Verify the IP address is configured:

```
cumulus@spine1$ ip addr show lo
1: lo: <LOOPBACK,UP,LOWER_UP>
mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host
            global
                inet 10.2.1.3/32 scope
            global
                inet 10.10.10.10/32 scope
            global
                inet6 ::1/128 scope host
                    valid_lft forever
                    preferred_lft forever
```

### **spine2**

Use a text editor to edit the network configuration:

```
cumulus@spine2$ sudo nano /etc/network/interfaces
```

Add the 10.10.10.10/32 address to the loopback address:

```
auto lo
iface lo inet loopback
    address 10.2.1.4/32
    address 10.10.10.10/32
```

Run ifreload -a:

```
cumulus@spine2$ sudo ifreload -a
```

Verify the IP address is configured:

```
cumulus@spine2$ ip addr show lo
1: lo: <LOOPBACK,UP,LOWER_UP>
mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host
            global
                inet 10.2.1.4/32 scope
            global
                inet 10.10.10.10/32 scope
            global
                inet6 ::1/128 scope host
                    valid_lft forever
                    preferred_lft forever
```

## Configuring the Service Node vxsnd.conf File

### spine1

Use a text editor to edit the network configuration:

```
cumulus@spine1$ sudo nano /etc/vxsnd.conf
```

Change the following values:

```
svcnode_ip = 10.10.10.10  
svcnode_peers = 10.2.1.4  
src_ip = 10.2.1.3
```

- i** This sets the address on which the service node listens to VXLAN messages to the configured Anycast address and sets it to sync with spine2.

Restart the vxsnd daemon:

```
cumulus@spine1$ sudo systemctl  
restart vxsnd.service
```

### spine2

Use a text editor to edit the network configuration:

```
cumulus@spine2$ sudo nano /etc/vxsnd.conf
```

Change the following values:

```
svcnode_ip = 10.10.10.10  
svcnode_peers = 10.2.1.3  
src_ip = 10.2.1.4
```

- i** This sets the address on which the service node listens to VXLAN messages to the configured Anycast address and sets it to sync with spine1.

Restart the vxsnd daemon:

```
cumulus@spine1$ sudo systemctl  
restart vxsnd.service
```

## Reconfiguring the VTEPs (Leafs) to Use the Anycast Address

### leaf1

Use a text editor to edit the network configuration:

```
cumulus@leaf1$ sudo nano /etc/network/interfaces
```

Change the `vxrd-svcnode-ip` field to the Anycast address:

```
auto lo
iface lo inet loopback
    address 10.2.1.1
    vxrd-svcnode-ip 10.10.10.10
```

Run `ifreload -a`:

```
cumulus@leaf1$ sudo ifreload -a
```

Verify the new service node is configured:

```
cumulus@leaf1$ ip -d link show vni-10
35: vni-10: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc noqueue master br-10 state UNKNOWN mode DEFAULT
    link/ether 46:c6:57:fc:1f:54 brd ff:ff:ff:ff:ff:ff
        vxlan id 10 remote 10.10.10.10 local 10.2.1.1 srcport 32768 61000 dstport 4789 ageing 300 svcnode 10.10.10.10
                    bridge_slave
```

```
cumulus@leaf1$ ip -d link show vni-2000
39: vni-2000: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc noqueue master br-20 state UNKNOWN mode DEFAULT
```

### leaf2

Use a text editor to edit the network configuration:

```
cumulus@leaf2$ sudo nano /etc/network/interfaces
```

Change the `vxrd-svcnode-ip` field to the Anycast address:

```
auto lo
iface lo inet loopback
    address 10.2.1.2
    vxrd-svcnode-ip 10.10.10.10
```

Run `ifreload -a`:

```
cumulus@leaf2$ sudo ifreload -a
```

Verify the new service node is configured:

```
cumulus@leaf2$ ip -d link show vni-10
35: vni-10: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc noqueue master br-10 state UNKNOWN mode DEFAULT
    link/ether 4e:03:a7:47:a7:9d brd ff:ff:ff:ff:ff:ff
        vxlan id 10 remote 10.10.10.10 local 10.2.1.2 srcport 32768 61000 dstport 4789 ageing 300 svcnode 10.10.10.10
                    bridge_slave
```

```
cumulus@leaf2$ ip -d link show vni-2000
39: vni-2000: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc noqueue master br-20 state UNKNOWN mode DEFAULT
```

```

link/ether 4a:fd:88:c3:fa:
df brd ff:ff:ff:ff:ff:ff
  vxlan id 2000 remote
10.10.10.10 local 10.2.1.1
srcport 32768 61000 dstport
4789 ageing 300 svcnode
10.10.10.10
  bridge_slave

cumulus@leaf1$ ip -d link show
vni-30
37: vni-30: <BROADCAST,
MULTICAST,UP,LOWER_UP> mtu
1500 qdisc noqueue master br-
30 state UNKNOWN mode DEFAULT
  link/ether 3e:b3:dc:f3:bd:
2b brd ff:ff:ff:ff:ff:ff
  vxlan id 30 remote
10.10.10.10 local 10.2.1.1
srcport 32768 61000 dstport
4789 ageing 300 svcnode
10.10.10.10
  bridge_slave
  
```

**⚠** The **svcnode** 10.10.10.10 means the interface has the correct service node configured.

Use the **vxrdctl vxlans** command to check the service node:

```

cumulus@leaf1$ vxrdctl vxlans
VNI      Local Addr      Svc
Node
===
=====
10        10.2.1.1
10.2.1.3
30        10.2.1.1
10.2.1.3
2000      10.2.1.1
10.2.1.3
  
```

```

link/ether 72:3a:bd:06:00:
b7 brd ff:ff:ff:ff:ff:ff
  vxlan id 2000 remote
10.10.10.10 local 10.2.1.2
srcport 32768 61000 dstport
4789 ageing 300 svcnode
10.10.10.10
  bridge_slave
  
```

```

cumulus@leaf2$ ip -d link show
vni-30
37: vni-30: <BROADCAST,
MULTICAST,UP,LOWER_UP> mtu
1500 qdisc noqueue master br-
30 state UNKNOWN mode DEFAULT
  link/ether 22:65:3f:63:08:
bd brd ff:ff:ff:ff:ff:ff
  vxlan id 30 remote
10.10.10.10 local 10.2.1.2
srcport 32768 61000 dstport
4789 ageing 300 svcnode
10.10.10.10
  bridge_slave
  
```

**⚠** The **svcnode** 10.10.10.10 means the interface has the correct service node configured.

Use the **vxrdctl vxlans** command to check the service node:

```

cumulus@leaf2$ vxrdctl vxlans
VNI      Local Addr      Svc
Node
===
=====
10        10.2.1.2
10.2.1.3
30        10.2.1.2
10.2.1.3
2000      10.2.1.2
10.2.1.3
  
```

## Testing Connectivity

Repeat the ping tests from the previous section. Here is the table again for reference:

VNI	server1	server2
10	10.10.10.1	10.10.10.2
2000	10.10.20.1	10.10.20.2
30	10.10.30.1	10.10.30.2

```
cumulus@server1:~$ ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=5.32 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.206 ms
^C
--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.206/2.767/5.329/2.562 ms

PING 10.10.20.2 (10.10.20.2) 56(84) bytes of data.
64 bytes from 10.10.20.2: icmp_seq=1 ttl=64 time=1.64 ms
64 bytes from 10.10.20.2: icmp_seq=2 ttl=64 time=0.187 ms
^C
--- 10.10.20.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.187/0.914/1.642/0.728 ms

cumulus@server1:~$ ping 10.10.30.2
PING 10.10.30.2 (10.10.30.2) 56(84) bytes of data.
64 bytes from 10.10.30.2: icmp_seq=1 ttl=64 time=1.63 ms
64 bytes from 10.10.30.2: icmp_seq=2 ttl=64 time=0.191 ms
^C
--- 10.10.30.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.191/0.913/1.635/0.722 ms
```

## Additional Resources

Both `vxsnd` and `vxrd` have man pages in Cumulus Linux.

For `vxsnd`:

```
cumulus@spine1$ man vxsnd
```

For vxrd:

```
cumulus@leaf1$ man vxrd
```

## See Also

- <https://tools.ietf.org/html/rfc7348>
- <http://en.wikipedia.org/wiki/Anycast>

## LNV VXLAN Active-Active Mode

*LNV active-active mode* allows a pair of **MLAG** switches to act as a single VTEP, providing active-active VXLAN termination for bare metal as well as virtualized workloads.

## Contents

- Terminology and Definitions (see page 330)
- Configuring LNV Active-Active Mode (see page 331)
  - Active-Active VTEP Anycast IP Behavior (see page 332)
  - Failure Scenario Behaviors (see page 332)
  - Checking VXLAN Interface Configuration Consistency (see page 333)
  - Configuring the Anycast IP Address (see page 333)
    - Explanation of Variables (see page 334)
- Example VXLAN Active-Active Configuration (see page 335)
  - Quagga Configuration (see page 335)
  - Layer 3 IP Addressing (see page 335)
  - Quagga Configuration (see page 340)
  - Host Configuration (see page 344)
  - Enable the Registration Daemon (see page 345)
  - Configuring a VTEP (see page 346)
  - Enable the Service Node Daemon (see page 346)
  - Configuring the Service Node (see page 346)
- North-South Traffic in an LNV Environment (Advanced) (see page 348)
- Considerations for Virtual Topologies Using Cumulus VX (see page 360)
  - Node ID (see page 360)
  - Bonds with Vagrant (see page 360)
- Troubleshooting with LNV Active-Active (see page 361)
- Caveats and Errata (see page 363)
- See Also (see page 363)

## Terminology and Definitions

Term	Definition
vxrd	VXLAN registration daemon. Runs on the switch that is mapping VLANs to VXLANs. The vxrd daemon needs to be configured to register to a service node. This turns the switch into a VTEP.
VTEP	Virtual tunnel endpoint. This is an encapsulation and decapsulation point for VXLANs.
active-active VTEP	A pair of switches acting as a single VTEP.
ToR	Top of Rack switch. Also referred to as a <i>leaf</i> or access switch.
Spine	The aggregation switch for multiple leafs. Specifically used when a data center is using a <a href="#">Clos network architecture</a> . Read more about the spine-leaf architecture in this <a href="#">white paper</a> .
vxsnd	VXLAN service node daemon, that can be run to register multiple VTEPs.
vxsn	The switch running the <b>vxsnd</b> . Also known as the VXLAN service node.
exit leaf	A switch dedicated to peering the Clos network to an outside network. Also referred to as border leafs, service leafs or edge leafs.
anycast	When an IP address is advertised from multiple locations. Allows multiple devices to share the same IP and effectively load balance traffic across them. With LNV, anycast is used in 2 places: <ol style="list-style-type: none"> <li>1. To share a VTEP IP address between a pair of MLAG switches.</li> <li>2. To load balance traffic for service nodes (for example, service nodes share an IP address).</li> </ol>
ASIC	Application-specific integrated circuit. Also referred to as hardware, or hardware accelerated. Encapsulation and decapsulation are required for the best performance VXLAN supported ASIC.
RIOT	Routing In and Out of Tunnels. Allows a VXLAN bridge to have a Switch VLAN interface associated with it, and traffic to exit a VXLAN into the layer 3 fabric.

## Configuring LNV Active-Active Mode

LNV requires the following underlying technologies to work correctly.

Technology	More Information
MLAG	Refer to the <a href="#">MLAG chapter (see page )</a> for more detailed configuration information. Configurations for the demonstration are provided below.

Technology	More Information
OSPF or BGP	Refer to the <a href="#">OSPF chapter</a> or the <a href="#">BGP Chapter</a> for more detailed configuration information. Configurations for the demonstration are provided below.
LNV	Refer to the <a href="#">LNV chapter (see page 330)</a> for more detailed configuration information. Configurations for the demonstration are provided below.
STP	<a href="#">BPDU filter and BPDU guard (see page )</a> should be enabled in the VXLAN interfaces if <a href="#">STP (see page 330)</a> is enabled in the bridge that is connected to the VXLAN. Configurations for the demonstration are provided below.

## Active-Active VTEP Anycast IP Behavior

Each individual switch within an MLAG pair should be provisioned with a virtual IP address in the form of an anycast IP address for VXLAN data-path termination. The VXLAN termination address is an anycast IP address that you configure as a `cldagd` parameter (`cldagd-vxlan-anycast-ip`) under the loopback interface. `cldagd` dynamically adds and removes this address as the loopback interface address as follows:

1	When the switches boot up, <code>ifupdown2</code> places all VXLAN interfaces in a <a href="#">PROTO_DOWN state (see page )</a> . The configured anycast addresses are not configured yet.
2	MLAG peering takes place, and a successful VXLAN interface consistency check between the switches occurs.
3	<code>cldagd</code> (the daemon responsible for MLAG) adds the anycast address to the loopback interface. It then changes the local IP address of the VXLAN interface from a unique address to the anycast virtual IP address and puts the interface in an UP state.

## Failure Scenario Behaviors

Scenario	Behavior
The peer link goes down.	The primary MLAG switch continues to keep all VXLAN interfaces up with the anycast IP address while the secondary switch brings down all VXLAN interfaces and places them in a PROTO_DOWN state. The secondary MLAG switch removes the anycast IP address from the loopback interface and changes the local IP address of the VXLAN interface to the configured unique IP address.
One of the switches goes down.	The other operational switch continues to use the anycast IP address.
<code>cldagd</code> is stopped.	All VXLAN interfaces are put in a PROTO_DOWN state. The anycast IP address is removed from the loopback interface and the local IP addresses of the VXLAN interfaces are changed from the anycast IP address to unique non-virtual IP addresses.

Scenario	Behavior
MLAG peering could not be established between the switches.	<code>clagd</code> brings up all the VXLAN interfaces after the reload timer expires with the configured unique IP address. This allows the VXLAN interface to be up and running on both switches even though peering is not established.
When the peer link goes down but the peer switch is up ( that is, the backup link is active).	All VXLAN interfaces are put into a PROTO_DOWN state on the secondary switch.
A configuration mismatch between the MLAG switches	The VXLAN interface is placed into a PROTO_DOWN state on the secondary switch.

## Checking VXLAN Interface Configuration Consistency

The LNV active-active configuration for a given VXLAN interface has to be consistent between the MLAG switches for correct traffic behavior. MLAG ensures that the configuration consistency is met before bringing the VXLAN interfaces up.

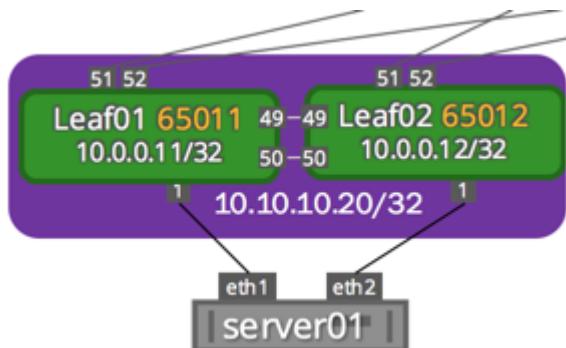
The consistency checks include:

- The anycast virtual IP address for VXLAN termination must be the same on each pair of switches.
- A VXLAN interface with the same VXLAN-ID must be configured and administratively up on both switches.

You can use the `clagctl` command to check if any VXLAN switches are in a PROTO\_DOWN state.

## Configuring the Anycast IP Address

With MLAG peering, both switches use an anycast IP address for VXLAN encapsulation and decapsulation. This allows remote VTEPs to learn the host MAC addresses attached to the MLAG switches against one logical VTEP, even though the switches independently encapsulate and decapsulate layer 2 traffic originating from the host. The anycast address under the loopback interface can be configured as shown below.



**leaf01: /etc/network/interfaces snippet**

```
auto lo
iface lo inet loopback
  address 10.0.0.11/32
  vxrd-src-ip 10.0.0.11
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip 10.10.10.20
```

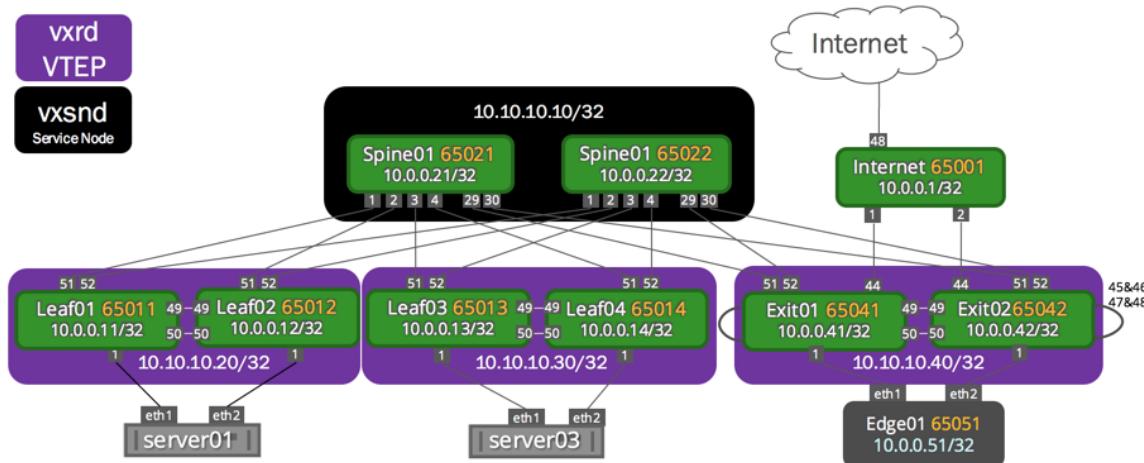
**leaf02: /etc/network/interfaces snippet**

```
auto lo
iface lo inet loopback
  address 10.0.0.12/32
  vxrd-src-ip 10.0.0.12
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip 10.10.10.20
```

## Explanation of Variables

Variable	Explanation
vxrd-src-ip	The unique IP address for the vxrd to bind to.
vxrd-svcnode-ip	The service node anycast IP address in the topology. In this demonstration, this is an anycast IP address being shared by both spine switches.
clagd-vxlan-anycast-ip	The anycast address for the MLAG pair to share and bind to when MLAG is up and running.

## Example VXLAN Active-Active Configuration



Note the configuration of the local IP address in the VXLAN interfaces below. They are configured with individual IP addresses, which `clagd` changes to anycast upon MLAG peering.

## Quagga Configuration

The layer 3 fabric can be configured using [BGP](#) (see page 330) or [OSPF](#) (see page 330). The following example uses BGP unnumbered. The MLAG switch configuration for the topology above is shown below.

## Layer 3 IP Addressing

The IP address configuration for this example:

**spine01: /etc/network/interfaces**

```
auto lo
iface lo inet loopback
    address 10.0.0.21/32
    address 10.10.10.10/32

auto eth0
iface eth0 inet dhcp

# downlinks
auto swp1
iface swp1

auto swp2
iface swp2

auto swp3
iface swp3
```

**spine02: /etc/network/interfaces**

```
auto lo
iface lo inet loopback
    address 10.0.0.22/32
    address 10.10.10.10/32

auto eth0
iface eth0 inet dhcp

# downlinks
auto swp1
iface swp1

auto swp2
iface swp2

auto swp3
iface swp3
```

```
auto swp4
iface swp4
```

```
auto swp29
iface swp29
```

```
auto swp30
iface swp30
```

```
auto swp4
iface swp4
```

```
auto swp29
iface swp29
```

```
auto swp30
iface swp30
```

**leaf01: /etc/network/interfaces**

```
auto lo
iface lo inet loopback
  address 10.0.0.11/32
  vxrd-src-ip 10.0.0.11
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.20
```

```
auto eth0
iface eth0 inet dhcp
```

```
# peerlinks
auto swp49
iface swp49
```

```
auto swp50
iface swp50
```

```
auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  bond-mode 802.3ad
  bond-miimon 100
  bond-use-carrier 1
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy
layer3+4
```

```
auto peerlink.4094
iface peerlink.4094
  address 169.254.1.1/30
  clagd-peer-ip 169.254.1.2
  clagd-backup-ip 10.0.0.12
  clagd-sys-mac 44:39:39:FF:40:
```

94

**leaf02: /etc/network/interfaces**

```
auto lo
iface lo inet loopback
  address 10.0.0.12/32
  vxrd-src-ip 10.0.0.12
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.20
```

```
auto eth0
iface eth0 inet dhcp
```

```
# peerlinks
auto swp49
iface swp49
```

```
auto swp50
iface swp50
```

```
auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  bond-mode 802.3ad
  bond-miimon 100
  bond-use-carrier 1
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy
layer3+4
```

```
auto peerlink.4094
iface peerlink.4094
  address 169.254.1.2/30
  clagd-peer-ip 169.254.1.1
  clagd-backup-ip 10.0.0.11
  clagd-sys-mac 44:39:39:FF:40:
```

94

```

# Downlinks
auto swp1
iface swp1

auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1

# bridges / vlan that contain
peerlink and downlinks for L2
connectivity

auto native
iface native
    bridge-ports peerlink bond0
vxlan1
    bridge-stp on
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard vxlan1=yes

auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10
    bridge-stp on
    mstpctl-portbpdufilter
vxlan10=yes
    mstpctl-bpduguard
vxlan10=yes

auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20
    bridge-stp on
    mstpctl-portbpdufilter
vxlan20=yes
    mstpctl-bpduguard vxlan20=yes

#vxlan config
auto vxlan1

```

```

# Downlinks
auto swp1
iface swp1

auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1

# bridges / vlan that contain
peerlink and downlinks for L2
connectivity

auto native
iface native
    bridge-ports peerlink bond0
vxlan1
    bridge-stp on
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard
vxlan1=yes

auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10
    bridge-stp on
    mstpctl-portbpdufilter
vxlan10=yes
    mstpctl-bpduguard
vxlan10=yes

auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20
    bridge-stp on
    mstpctl-portbpdufilter
vxlan20=yes
    mstpctl-bpduguard vxlan20=yes

#vxlan config
auto vxlan1

```

```

iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
  10.0.0.11

auto vxlan10
iface vxlan10
  vxlan-id 10
  vxlan-local-tunnelip
  10.0.0.11

auto vxlan20
iface vxlan20
  vxlan-id 20
  vxlan-local-tunnelip
  10.0.0.11

# uplinks
auto swp51
iface swp51

auto swp52
iface swp52
  
```

```

auto vxlan1
iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
  10.0.0.12

auto vxlan10
iface vxlan10
  vxlan-id 10
  vxlan-local-tunnelip
  10.0.0.12

auto vxlan20
iface vxlan20
  vxlan-id 20
  vxlan-local-tunnelip
  10.0.0.12

# uplinks
auto swp51
iface swp51

auto swp52
iface swp52
  
```

**leaf3:** /etc/network/interfaces

```

auto lo
iface lo inet loopback
  address 10.0.0.13/32
  vxrd-src-ip 10.0.0.13
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.30

auto eth0
iface eth0 inet dhcp

# peerlinks
auto swp49
iface swp49

auto swp50
iface swp50p

auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  
```

**leaf4:** /etc/network/interfaces

```

auto lo
iface lo inet loopback
  address 10.0.0.14/32
  vxrd-src-ip 10.0.0.14
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.30

auto eth0
iface eth0 inet dhcp

# peerlinks
auto swp49
iface swp49

auto swp50
iface swp50

auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  
```

```

bond-mode 802.3ad
bond-miimon 100
bond-use-carrier 1
bond-lacp-rate 1
bond-min-links 1
bond-xmit-hash-policy
layer3+4

auto peerlink.4094
iface peerlink.4094
    address 169.254.1.1/30
    clagd-peer-ip 169.254.1.2
    clagd-backup-ip 10.0.0.14
    clagd-sys-mac 44:39:39:FF:40:
95

# Downlinks
auto swp1
iface swp1

auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1

# bridges / vlan that contain
# peerlink and downlinks for L2
# connectivity

auto native
iface native
    bridge-ports peerlink bond0
vxlan1
    bridge-stp on
    mstpcctl-portbpdufilter
vxlan1=yes
    mstpcctl-bpduguard
vxlan1=yes

auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10
    bridge-stp on

```

```

bond-mode 802.3ad
bond-miimon 100
bond-use-carrier 1
bond-lacp-rate 1
bond-min-links 1
bond-xmit-hash-policy
layer3+4

auto peerlink.4094
iface peerlink.4094
    address 169.254.1.2/30
    clagd-peer-ip 169.254.1.1
    clagd-backup-ip 10.0.0.13
    clagd-sys-mac 44:39:39:FF:40:
95

# Downlinks
auto swp1
iface swp1

auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1

# bridges / vlan that contain
# peerlink and downlinks for L2
# connectivity

auto native
iface native
    bridge-ports peerlink bond0
vxlan1
    bridge-stp on
    mstpcctl-portbpdufilter
vxlan1=yes
    mstpcctl-bpduguard
vxlan1=yes

auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10
    bridge-stp on

```

```

mstpctl-portbpdufilter
vxlan10=yes
  mstpctl-bpduguard
vxlan10=yes

auto vlan20
iface vlan20
  bridge-ports peerlink.20
bond0.20 vxlan20
  bridge-stp on
  mstpctl-portbpdufilter
vxlan20=yes
  mstpctl-bpduguard vxlan20=yes

#vxlan config
auto vxlan1
iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
10.0.0.13

auto vxlan10
iface vxlan10
  vxlan-id 10
  vxlan-local-tunnelip
10.0.0.13

auto vxlan20
iface vxlan20
  vxlan-id 20
  vxlan-local-tunnelip
10.0.0.13

# uplinks
auto swp51
iface swp51

auto swp52
iface swp52
  
```

```

mstpctl-portbpdufilter
vxlan10=yes
  mstpctl-bpduguard
vxlan10=yes

auto vlan20
iface vlan20
  bridge-ports peerlink.20
bond0.20 vxlan20
  bridge-stp on
  mstpctl-portbpdufilter
vxlan20=yes
  mstpctl-bpduguard vxlan20=yes

#vxlan config
auto vxlan1
iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
10.0.0.14

auto vxlan10
iface vxlan10
  vxlan-id 10
  vxlan-local-tunnelip
10.0.0.14

auto vxlan20
iface vxlan20
  vxlan-id 20
  vxlan-local-tunnelip
10.0.0.14

# uplinks
auto swp51
iface swp51

auto swp52
iface swp52
  
```

## Quagga Configuration

The service nodes and registration nodes must all be routable between each other. The L3 fabric on Cumulus Linux can either be [BGP](#) (see page 330) or [OSPF](#) (see page 330). In this example, OSPF is used to demonstrate full reachability.

The Quagga configuration using OSPF:

**spine01:**/etc/quagga/Quagga.conf

**spine02:** /etc/quagga/Quagga.conf

```
!
interface swp1
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp2
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp3
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp4
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp29
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp30
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
router bgp 65020
  bgp router-id 10.0.0.21
  network 10.0.0.21/32
  network 10.10.10.10/32
  maximum-paths 64
  bgp bestpath as-path
  multipath-relax
    bgp bestpath compare-routerid
    bgp default show-hostname
    neighbor FABRIC peer-group
    neighbor FABRIC remote-as
  external
    neighbor FABRIC description
  Internal Fabric Network
    neighbor FABRIC
  advertisement-interval 0
    neighbor FABRIC timers 1 3
    neighbor FABRIC timers
  connect 3
    neighbor FABRIC capability
  extended-nexthop
    neighbor FABRIC prefix-list
  dc-spine in
    neighbor FABRIC prefix-list
  dc-spine out
    neighbor swp1 interface
```

```
!
interface swp1
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp2
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp3
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp4
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp29
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp30
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
router bgp 65020
  bgp router-id 10.0.0.22
  network 10.0.0.22/32
  network 10.10.10.10/32
  maximum-paths 64
  bgp bestpath as-path
  multipath-relax
    bgp bestpath compare-routerid
    bgp default show-hostname
    neighbor FABRIC peer-group
    neighbor FABRIC remote-as
  external
    neighbor FABRIC description
  Internal Fabric Network
    neighbor FABRIC
  advertisement-interval 0
    neighbor FABRIC timers 1 3
    neighbor FABRIC timers
  connect 3
    neighbor FABRIC capability
  extended-nexthop
    neighbor FABRIC prefix-list
  dc-spine in
    neighbor FABRIC prefix-list
  dc-spine out
    neighbor swp1 interface
```

```

neighbor swp1 peer-group
FABRIC
  neighbor swp2 interface
  neighbor swp2 peer-group
FABRIC
  neighbor swp3 interface
  neighbor swp3 peer-group
FABRIC
  neighbor swp4 interface
  neighbor swp4 peer-group
FABRIC
  neighbor swp29 interface
  neighbor swp29 peer-group
FABRIC
  neighbor swp30 interface
  neighbor swp30 peer-group
FABRIC
!
ip prefix-list dc-spine seq 10
permit 0.0.0.0/0
ip prefix-list dc-spine seq 15
permit 10.0.0.0/24 le 32
ip prefix-list dc-spine seq 20
permit 10.10.10.0/24 le 32
ip prefix-list dc-spine seq 30
permit 172.16.1.0/24
ip prefix-list dc-spine seq 40
permit 172.16.2.0/24
ip prefix-list dc-spine seq 50
permit 172.16.3.0/24
ip prefix-list dc-spine seq 60
permit 172.16.4.0/24
ip prefix-list dc-spine seq
500 deny any
!
```

```

neighbor swp1 peer-group
FABRIC
  neighbor swp2 interface
  neighbor swp2 peer-group
FABRIC
  neighbor swp3 interface
  neighbor swp3 peer-group
FABRIC
  neighbor swp4 interface
  neighbor swp4 peer-group
FABRIC
  neighbor swp29 interface
  neighbor swp29 peer-group
FABRIC
  neighbor swp30 interface
  neighbor swp30 peer-group
FABRIC
!
ip prefix-list dc-spine seq 10
permit 0.0.0.0/0
ip prefix-list dc-spine seq 15
permit 10.0.0.0/24 le 32
ip prefix-list dc-spine seq 20
permit 10.10.10.0/24 le 32
ip prefix-list dc-spine seq 30
permit 172.16.1.0/24
ip prefix-list dc-spine seq 40
permit 172.16.2.0/24
ip prefix-list dc-spine seq 50
permit 172.16.3.0/24
ip prefix-list dc-spine seq 60
permit 172.16.4.0/24
ip prefix-list dc-spine seq
500 deny any
!
```

**leaf01:** /etc/quagga/Quagga.conf

```

!
interface swp51
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp52
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
router bgp 65011
```

**leaf02:** /etc/quagga/Quagga.conf

```

!
interface swp51
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
interface swp52
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
router bgp 65012
```

```

bgp router-id 10.0.0.11
network 10.0.0.11/32
network 172.16.1.0/24
network 10.10.10.20/32
maximum-paths 64
bgp bestpath as-path
multipath-relax
bgp bestpath compare-routerid
bgp default show-hostname
neighbor FABRIC peer-group
neighbor FABRIC remote-as
external
neighbor FABRIC description
Internal Fabric Network
neighbor FABRIC
advertisement-interval 0
neighbor FABRIC timers 1 3
neighbor FABRIC timers
connect 3
neighbor FABRIC capability
extended-nexthop
neighbor FABRIC filter-list
dc-leaf-out out
neighbor swp51 interface
neighbor swp51 peer-group
FABRIC
neighbor swp52 interface
neighbor swp52 peer-group
FABRIC
!
ip as-path access-list dc-leaf-
out permit ^$*
!
```

```

bgp router-id 10.0.0.12
network 10.0.0.12/32
network 172.16.1.0/24
network 10.10.10.20/32
maximum-paths 64
bgp bestpath as-path
multipath-relax
bgp bestpath compare-routerid
bgp default show-hostname
neighbor FABRIC peer-group
neighbor FABRIC remote-as
external
neighbor FABRIC description
Internal Fabric Network
neighbor FABRIC
advertisement-interval 0
neighbor FABRIC timers 1 3
neighbor FABRIC timers
connect 3
neighbor FABRIC capability
extended-nexthop
neighbor FABRIC filter-list
dc-leaf-out out
neighbor swp51 interface
neighbor swp51 peer-group
FABRIC
neighbor swp52 interface
neighbor swp52 peer-group
FABRIC
!
ip as-path access-list dc-leaf-
out permit ^$*
!
```

#### **leaf03:** /etc/quagga/Quagga.conf

```

!
interface swp51
no ipv6 nd suppress-ra
ipv6 nd ra-interval 3
!
interface swp52
no ipv6 nd suppress-ra
ipv6 nd ra-interval 3
!
router bgp 65013
bgp router-id 10.0.0.13
network 10.0.0.13/32
```

#### **leaf04:** /etc/quagga/Quagga.conf

```

!
interface swp51
no ipv6 nd suppress-ra
ipv6 nd ra-interval 3
!
interface swp52
no ipv6 nd suppress-ra
ipv6 nd ra-interval 3
!
router bgp 65014
bgp router-id 10.0.0.14
network 10.0.0.14/32
```

```

network 172.16.3.0/24
network 10.10.10.30/32
maximum-paths 64
bgp bestpath as-path
multipath-relax
  bgp bestpath compare-routerid
  bgp default show-hostname
  neighbor FABRIC peer-group
  neighbor FABRIC remote-as
external
  neighbor FABRIC description
Internal Fabric Network
  neighbor FABRIC
advertisement-interval 0
  neighbor FABRIC timers 1 3
  neighbor FABRIC timers
connect 3
  neighbor FABRIC capability
extended-nexthop
  neighbor FABRIC filter-list
dc-leaf-out out
  neighbor swp51 interface
  neighbor swp51 peer-group
FABRIC
  neighbor swp52 interface
  neighbor swp52 peer-group
FABRIC
!
ip as-path access-list dc-leaf-
out permit ^$
```

```

network 172.16.3.0/24
network 10.10.10.30/32
maximum-paths 64
bgp bestpath as-path
multipath-relax
  bgp bestpath compare-routerid
  bgp default show-hostname
  neighbor FABRIC peer-group
  neighbor FABRIC remote-as
external
  neighbor FABRIC description
Internal Fabric Network
  neighbor FABRIC
advertisement-interval 0
  neighbor FABRIC timers 1 3
  neighbor FABRIC timers
connect 3
  neighbor FABRIC capability
extended-nexthop
  neighbor FABRIC filter-list
dc-leaf-out out
  neighbor swp51 interface
  neighbor swp51 peer-group
FABRIC
  neighbor swp52 interface
  neighbor swp52 peer-group
FABRIC
!
ip as-path access-list dc-leaf-
out permit ^$
```

## Host Configuration

In this example, the servers are running Ubuntu 14.04. A layer2 bond must be mapped from server01 and server03 to the respective switch. In Ubuntu this is done with subinterfaces.

### server01

```

auto lo
iface lo inet loopback

auto lo
iface lo inet static
  address 10.0.0.31/32

auto eth0
iface eth0 inet dhcp
```

### server03

```

auto lo
iface lo inet loopback

auto lo
iface lo inet static
  address 10.0.0.33/32

auto eth0
iface eth0 inet dhcp
```

```

auto eth1
iface eth1 inet manual
    bond-master bond0

auto eth2
iface eth2 inet manual
    bond-master bond0

auto bond0
iface bond0 inet static
    bond-slaves none
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1
    address 172.16.1.101/24

auto bond0.10
iface bond0.10 inet static
    address 172.16.10.101/24

auto bond0.20
iface bond0.20 inet static
    address 172.16.20.101/24

```

```

auto eth1
iface eth1 inet manual
    bond-master bond0

auto eth2
iface eth2 inet manual
    bond-master bond0

auto bond0
iface bond0 inet static
    bond-slaves none
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1
    address 172.16.1.103/24

auto bond0.10
iface bond0.10 inet static
    address 172.16.10.103/24

auto bond0.20
iface bond0.20 inet static
    address 172.16.20.103/24

```

## **Enable the Registration Daemon**

The registration daemon (`vxrd`) must be enabled on each ToR switch acting as a VTEP, that is participating in LNV. The daemon is installed by default.

1. Open the `/etc/default/vxrd` configuration file in a text editor.
2. Enable the daemon, then save the file.

```
START=yes
```

3. Restart the `vxrd` daemon.

```
cumulus@leaf:~$ sudo systemctl restart vxrd.service
```

## Configuring a VTEP

The registration node was configured earlier in `/etc/network/interfaces`; no additional configuration is typically needed. Alternatively, the configuration can be done in `/etc/vxrd.conf`, which has additional configuration knobs available.

## Enable the Service Node Daemon

1. Open the `/etc/default/vxsnd` configuration file in a text editor.
2. Enable the daemon, then save the file:

```
START=yes
```

3. Restart the daemon.

```
cumulus@spine:~$ sudo systemctl restart vxsnd.service
```

## Configuring the Service Node

To configure the service node daemon, edit the `/etc/vxsnd.conf` configuration file:

**spine01:** /etc/vxsnd.conf

```
svcnodes_ip = 10.10.10.10
src_ip = 10.0.0.21
svcnodes_peers = 10.0.0.21
10.0.0.22
```

Full configuration of vxsnd.conf

**spine02:** /etc/vxsnd.conf

```
svcnodes_ip = 10.10.10.10
src_ip = 10.0.0.22
svcnodes_peers = 10.0.0.21
10.0.0.22
```

Full configuration of vxsnd.conf

```
[common]
# Log level is one of DEBUG,
INFO, WARNING, ERROR, CRITICAL
#loglevel = INFO
# Destination for log
message. Can be a file name,
'stdout', or 'syslog'
#logdest = syslog
# log file size in bytes. Used
when logdest is a file
```

```
[common]
# Log level is one of DEBUG,
INFO, WARNING, ERROR, CRITICAL
#loglevel = INFO
# Destination for log
message. Can be a file name,
'stdout', or 'syslog'
#logdest = syslog
# log file size in bytes. Used
when logdest is a file
```

```

#logfilesize = 512000
# maximum number of log files
stored on disk. Used when
logdest is a file
#logbackupcount = 14
# The file to write the pid.
If using monit, this must
match the one
# in the vxsnd.rc
#pidfile = /var/run/vxsnd.pid
# The file name for the unix
domain socket used for mgmt.
#udsfile = /var/run/vxsnd.sock
# UDP port for vxfld control
messages
#vxfld_port = 10001
# This is the address to which
registration daemons send
control messages for
# registration and/or BUM
packets for replication
svcnode_ip = 10.10.10.10
# Holdtime (in seconds) for
soft state. It is used when
sending a
# register msg to peers in
response to learning a <vni,
addr> from a
# VXLAN data pkt
#holdtime = 90
# Local IP address to bind to
for receiving inter-vxsnd
control traffic
src_ip = 10.0.0.21
[vxsnd]
# Space separated list of IP
addresses of vxsnd to share
state with
svcnode_peers = 10.0.0.21
10.0.0.22
# When set to true, the
service node will listen for
vxlan data traffic
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#enable_vxlan_listen = true
# When set to true, the
svcnode_ip will be installed
on the loopback

```

```

#logfilesize = 512000
# maximum number of log files
stored on disk. Used when
logdest is a file
#logbackupcount = 14
# The file to write the pid.
If using monit, this must
match the one
# in the vxsnd.rc
#pidfile = /var/run/vxsnd.pid
# The file name for the unix
domain socket used for mgmt.
#udsfile = /var/run/vxsnd.sock
# UDP port for vxfld control
messages
#vxfld_port = 10001
# This is the address to which
registration daemons send
control messages for
# registration and/or BUM
packets for replication
svcnode_ip = 10.10.10.10
# Holdtime (in seconds) for
soft state. It is used when
sending a
# register msg to peers in
response to learning a <vni,
addr> from a
# VXLAN data pkt
#holdtime = 90
# Local IP address to bind to
for receiving inter-vxsnd
control traffic
src_ip = 10.0.0.22
[vxsnd]
# Space separated list of IP
addresses of vxsnd to share
state with
svcnode_peers = 10.0.0.21
10.0.0.22
# When set to true, the
service node will listen for
vxlan data traffic
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#enable_vxlan_listen = true
# When set to true, the
svcnode_ip will be installed
on the loopback

```

```

# interface, and it will be
withdrawn when the vxsnd is no
longer in
# service. If set to true,
the svcnode_ip configuration
# variable must be defined.
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#install_svcnode_ip = false
# Seconds to wait before
checking the database to age
out stale entries
#age_check = 90
  
```

```

# interface, and it will be
withdrawn when the vxsnd is no
longer in
# service. If set to true,
the svcnode_ip configuration
# variable must be defined.
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#install_svcnode_ip = false
# Seconds to wait before
checking the database to age
out stale entries
#age_check = 90
  
```

## **North-South Traffic in an LNV Environment (Advanced)**

The following configuration is recommended for advanced users, because it describes a non-standard configuration where the service node and registration node reside on the same switch, which is not the supported use case.

Read about north-south traffic in an LNV ...

The table below covers some of the scenarios for configuring VXLAN gateways (for traffic to exit an L2 VXLAN), as well as a few advantages/caveats of each:

Gateway Is Configured	Advantages	Caveats
Using a pair of exit leafs	This is the simplest configuration available. Many large data centers use exit leafs to allow for consolidation of network services (firewalls, load balancers, etc); performing the VXLAN gateway on the exit leafs is thus a logical choice. None of the caveats for the other two configuration methods exist for exit leafs.	<ul style="list-style-type: none"> <li>• Exit leafs should be able to handle all VXLANs simultaneously (since they have to act as a gateway for every VXLAN configured).</li> <li>• The bandwidth that the data center holistically has to service providers should be able to be handled by the exit leafs.</li> <li>• In high bandwidth scenarios these caveats can be mitigated by having multiple pairs of exit leafs.</li> </ul>
Using every pair of leafs	Any pair of leafs can allow traffic to exit to the internet.	If VXLAN tenant separation must be maintained (so VXLAN10 can't talk to VXLAN20 ever without going through a firewall), iptables/ACLs or VRFs must be used to separate traffic, to ensure it can't bypass security policies. This increases complexity on the rest of the network, as policies must exist on all devices to keep traffic segregated. If there is no segregation requirements, this caveat does not exist.

Gateway Is Configured	Advantages	Caveats
Using a pair of spines	<p>Configuration complexity only exists on the Spine switches, Leaf switches have no configuration changes from a default LNV configuration. This also works for small data centers or a PoP (Point of Presence) where exit leafs are out of scope.</p>	<p>The <b>vxrd</b> and <b>vxsnd</b> daemon must be on the same pair of switches. This means that 4 IP addresses must be used on conjunction and distributed into the fabric:</p> <ul style="list-style-type: none"> <li>• One unique loopback IP address for <b>vxrd</b></li> <li>• One anycast for <b>vxrd</b></li> <li>• One unique loopback IP address for <b>vxsnd</b></li> <li>• One anycast for <b>vxsnd</b></li> </ul> <p>This is because <b>vxrd</b> and <b>vxsnd</b> use the same port to communicate, and need another set of IP addresses to make a different socket (IP address + port) to allow communication to happen between the <b>vxrd</b> and <b>vxsnd</b> daemons properly.</p>

## Using a Pair of Exit Leafs

One method of exiting a network is using the gateway on the exit leafs themselves.

### Exiting a VXLAN with a Broadcom Trident II

The Trident II ASIC has a limitation where a L2 bridge that contains a VXLAN interface can not also have an IP address assigned to it. This is an expected limitation with this ASIC, because of the ordering of the decapsulation. A packet that is decapsulated will already have passed the portion of the ASIC capable of reading the IP address lookup (for example, VXLAN lookup happens before IP address lookup).

Refer to the [Cumulus Networks Hardware Compatibility List](#) to determine which ASIC is running on the switch.

The code snippet below illustrates this concept:

```
auto example_bridge
iface example_bridge
    bridge-ports VXLAN10
    address 5.5.5.1/24
auto VXLAN10
iface VXLAN10
    vxlan-id 10
```



The above snippet will not work on ASICs not capable of RIOT. This will not work on the Broadcom Trident II.

To solve this issue in environments where the Broadcom Trident II is used, a loopback cable can be configured (literally take a cable and attach both ends to the same switch). This will allow the switch to attach an IP address to a bridge that also contains a VXLAN. If additional bandwidth is required for the gateway, a bond (also known as an Etherchannel) can be configured as well.

The example configurations below represent a reference topology for exit leafs with Broadcom Trident II ASICs:

**exit01: /etc/network/interfaces snippet**

```
#interface for T2 configured
as loopback cable
auto hyperloopin
iface hyperloopin
    bond-slaves swp45 swp47
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+
    bond-lACP-rate 1
auto hyperloopout
iface hyperloopout
    bond-slaves swp46 swp48
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+
    bond-lACP-rate 1
auto vlan1svi
iface vlan1svi
    bridge-ports hyperloopout
    address 172.16.1.2/24
    address-virtual 00:00:5e:00:
01:01 172.16.1.1/24
auto native
iface native
    bridge-ports peerlink bond0
vxlan1 hyperloopin
    bridge-stp on
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard
vxlan1=yes
```

Full Configuration for /etc/network/interfaces

```
auto lo
```

**exit02: /etc/network/interfaces snippet**

```
#interface for T2 configured
as loopback cable
auto hyperloopin
iface hyperloopin
    bond-slaves swp45 swp47
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+
    bond-lACP-rate 1
auto hyperloopout
iface hyperloopout
    bond-slaves swp46 swp48
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+
    bond-lACP-rate 1
auto vlan1svi
iface vlan1svi
    bridge-ports hyperloopout
    address 172.16.1.3/24
    address-virtual 00:00:5e:00:
01:01 172.16.1.1/24
auto native
iface native
    bridge-ports peerlink bond0
vxlan1 hyperloopin
    bridge-stp on
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard
vxlan1=yes
```

Full Configuration for /etc/network/interfaces

```
auto lo
```

```

iface lo inet loopback
    address 10.0.0.41/32
    vxrd-src-ip 10.0.0.41
    vxrd-svcnode-ip 10.10.10.10
    clagd-vxlan-anycast-ip
        10.10.10.40
auto eth0
iface eth0 inet dhcp
# peerlinks
auto swp49
iface swp49
auto swp50
iface swp50
auto peerlink
iface peerlink
    bond-slaves swp49 swp50
    bond-mode 802.3ad
    bond-miimon 100
    bond-use-carrier 1
    bond-lacp-rate 1
    bond-min-links 1
    bond-xmit-hash-policy
layer3+4
auto peerlink.4094
iface peerlink.4094
    address 169.254.1.1/30
    clagd-peer-ip 169.254.1.2
    clagd-backup-ip 10.0.0.42
    clagd-sys-mac 44:39:39:FF:40:
96
# Downlinks
auto swp1
iface swp1
auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1
#interface for T2 configured
as loopback cable
auto hyperloopin
iface hyperloopin
    bond-slaves swp45 swp47
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad

```

```

iface lo inet loopback
    address 10.0.0.42/32
    vxrd-src-ip 10.0.0.42
    vxrd-svcnode-ip 10.10.10.10
    clagd-vxlan-anycast-ip
        10.10.10.40
auto eth0
iface eth0 inet dhcp
# peerlinks
auto swp49
iface swp49
auto swp50
iface swp50
auto peerlink
iface peerlink
    bond-slaves swp49 swp50
    bond-mode 802.3ad
    bond-miimon 100
    bond-use-carrier 1
    bond-lacp-rate 1
    bond-min-links 1
    bond-xmit-hash-policy
layer3+4
auto peerlink.4094
iface peerlink.4094
    address 169.254.1.2/30
    clagd-peer-ip 169.254.1.1
    clagd-backup-ip 10.0.0.41
    clagd-sys-mac 44:39:39:FF:40:
96
# Downlinks
auto swp1
iface swp1
auto bond0
iface bond0
    bond-slaves swp1
    clag-id 1
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad
    bond-xmit-hash-policy
layer3+4
    bond-lacp-rate 1
#interface for T2 configured
as loopback cable
auto hyperloopin
iface hyperloopin
    bond-slaves swp45 swp47
    bond-miimon 100
    bond-min-links 1
    bond-mode 802.3ad

```

```

bond-xmit-hash-policy
layer3+4
  bond-lacp-rate 1
auto hyperloopout
iface hyperloopout
  bond-slaves swp46 swp48
  bond-miimon 100
  bond-min-links 1
  bond-mode 802.3ad
  bond-xmit-hash-policy
layer3+4
  bond-lacp-rate 1
auto vlan1svi
iface vlan1svi
  bridge-ports hyperloopout
  address 172.16.1.2/24
  address-virtual 00:00:5e:00:01:01 172.16.1.1/24
auto vlan10svi
iface vlan10svi
  bridge-ports hyperloopout.10
  address 172.16.10.2/24
  address-virtual 00:00:5e:00:10:10 172.16.10.1/24
auto vlan20svi
iface vlan20svi
  bridge-ports hyperloopout.20
  address 172.16.20.2/24
  address-virtual 00:00:5e:00:20:20 172.16.20.1/24
# bridges / vlan that contain
peerlink and downlinks for L2
connectivity
auto native
iface native
  bridge-ports peerlink bond0
vxlan1 hyperloopin
  bridge-stp on
  mstpcctl-portbpdufilter
vxlan1=yes
  mstpcctl-bpduguard
vxlan1=yes
auto vlan10
iface vlan10
  bridge-ports peerlink.10
bond0.10 vxlan10 hyperloopin.10
  bridge-stp on
  mstpcctl-portbpdufilter
vxlan10=yes
  mstpcctl-bpduguard
vxlan10=yes

```

```

bond-xmit-hash-policy
layer3+4
  bond-lacp-rate 1
auto hyperloopout
iface hyperloopout
  bond-slaves swp46 swp48
  bond-miimon 100
  bond-min-links 1
  bond-mode 802.3ad
  bond-xmit-hash-policy
layer3+4
  bond-lacp-rate 1
auto vlan1svi
iface vlan1svi
  bridge-ports hyperloopout
  address 172.16.1.3/24
  address-virtual 00:00:5e:00:01:01 172.16.1.1/24
auto vlan10svi
iface vlan10svi
  bridge-ports hyperloopout.10
  address 172.16.10.3/24
  address-virtual 00:00:5e:00:10:10 172.16.10.1/24
auto vlan20svi
iface vlan20svi
  bridge-ports hyperloopout.20
  address 172.16.20.3/24
  address-virtual 00:00:5e:00:20:20 172.16.20.1/24
# bridges / vlan that contain
peerlink and downlinks for L2
connectivity
auto native
iface native
  bridge-ports peerlink bond0
vxlan1 hyperloopin
  bridge-stp on
  mstpcctl-portbpdufilter
vxlan1=yes
  mstpcctl-bpduguard
vxlan1=yes
auto vlan10
iface vlan10
  bridge-ports peerlink.10
bond0.10 vxlan10 hyperloopin.10
  bridge-stp on
  mstpcctl-portbpdufilter
vxlan10=yes
  mstpcctl-bpduguard
vxlan10=yes

```

```

auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20 hyperloopin.20
    bridge-stp on
    mstpcctl-portbpdufilter
vxlan20=yes
    mstpcctl-bpduguard vxlan20=yes
#vxlan config
auto vxlan1
iface vxlan1
    vxlan-id 1
    vxlan-local-tunnelip
10.0.0.41
auto vxlan10
iface vxlan10
    vxlan-id 10
    vxlan-local-tunnelip
10.0.0.41
auto vxlan20
iface vxlan20
    vxlan-id 20
    vxlan-local-tunnelip
10.0.0.41
# uplinks
auto swp51
iface swp51
auto swp52
iface swp52
#internet
auto swp44
iface swp44

```

```

auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20 hyperloopin.20
    bridge-stp on
    mstpcctl-portbpdufilter
vxlan20=yes
    mstpcctl-bpduguard vxlan20=yes
#vxlan config
auto vxlan1
iface vxlan1
    vxlan-id 1
    vxlan-local-tunnelip
10.0.0.42
auto vxlan10
iface vxlan10
    vxlan-id 10
    vxlan-local-tunnelip
10.0.0.42
auto vxlan20
iface vxlan20
    vxlan-id 20
    vxlan-local-tunnelip
10.0.0.42
# uplinks
auto swp51
iface swp51
auto swp52
iface swp52
#internet
auto swp44
iface swp44

```

One of the caveats of doing this is that each pair of exit leafs is restricted to 4094 tags to keep traffic separated (for example, 802.1q tags need to be utilized on the hyperloop/loopback to keep traffic separated).

Exiting a VXLAN with a RIOT-capable ASIC

**exit01:** /etc/network/interfaces **snippet**

```

auto native
iface native
    bridge-ports peerlink bond0
vxlan1 hyperloopin
    bridge-stp on
    address 172.16.1.2/24
    address-virtual 00:00:5e:00:
01:01 172.16.1.1/24

```

**exit02:** /etc/network/interfaces **snippet**

```

auto native
iface native
    bridge-ports peerlink bond0
vxlan1 hyperloopin
    bridge-stp on
    address 172.16.1.3/24
    address-virtual 00:00:5e:00:
01:01 172.16.1.1/24

```

```
mstpctl-portbpdufilter
vxlan1=yes
  mstpctl-bpduguard
vxlan1=yes
auto vxlan1
iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
10.0.0.41
```

```
mstpctl-portbpdufilter
vxlan1=yes
  mstpctl-bpduguard
vxlan1=yes
auto vxlan1
iface vxlan1
  vxlan-id 1
  vxlan-local-tunnelip
10.0.0.42
```

```
auto lo
iface lo inet loopback
  address 10.0.0.41/32
  vxrd-src-ip 10.0.0.41
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
10.10.10.40
auto eth0
iface eth0 inet dhcp
# peerlinks
auto swp49
iface swp49
auto swp50
iface swp50
auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  bond-mode 802.3ad
  bond-miimon 100
  bond-use-carrier 1
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy
layer3+4
auto peerlink.4094
iface peerlink.4094
  address 169.254.1.1/30
  clagd-peer-ip 169.254.1.2
  clagd-backup-ip 10.0.0.42
  clagd-sys-mac 44:39:39:FF:40:
96
# bridges / vlan that contain
peerlink and downlinks for L2
connectivity
auto native
iface native
  bridge-ports peerlink bond0
vxlan1 hyperloopin
  bridge-stp on
```

```
auto lo
iface lo inet loopback
  address 10.0.0.42/32
  vxrd-src-ip 10.0.0.42
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
10.10.10.40
auto eth0
iface eth0 inet dhcp
# peerlinks
auto swp49
iface swp49
auto swp50
iface swp50
auto peerlink
iface peerlink
  bond-slaves swp49 swp50
  bond-mode 802.3ad
  bond-miimon 100
  bond-use-carrier 1
  bond-lacp-rate 1
  bond-min-links 1
  bond-xmit-hash-policy
layer3+4
auto peerlink.4094
iface peerlink.4094
  address 169.254.1.2/30
  clagd-peer-ip 169.254.1.1
  clagd-backup-ip 10.0.0.41
  clagd-sys-mac 44:39:39:FF:40:
96
# bridges / vlan that contain
peerlink and downlinks for L2
connectivity
auto native
iface native
  bridge-ports peerlink bond0
vxlan1 hyperloopin
  bridge-stp on
```

```

address 172.16.1.2/24
address-virtual 00:00:5e:00:
01:01 172.16.1.1/24
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard
vxlan1=yes
auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10 hyperloopin.10
    bridge-stp on
    address 172.16.10.2/24
    address-virtual 00:00:5e:00:
10:10 172.16.10.1/24
    mstpctl-portbpdufilter
vxlan10=yes
    mstpctl-bpduguard
vxlan10=yes
auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20 hyperloopin.20
    bridge-stp on
    address 172.16.20.2/24
    address-virtual 00:00:5e:00:
20:20 172.16.20.1/24
    mstpctl-portbpdufilter
vxlan20=yes
    mstpctl-bpduguard vxlan20=yes
#vxlan config
auto vxlan1
iface vxlan1
    vxlan-id 1
    vxlan-local-tunnelip
10.0.0.41
auto vxlan10
iface vxlan10
    vxlan-id 10
    vxlan-local-tunnelip
10.0.0.41
auto vxlan20
iface vxlan20
    vxlan-id 20
    vxlan-local-tunnelip
10.0.0.41
# uplinks
auto swp51
iface swp51
auto swp52
iface swp52

```

```

address 172.16.1.3/24
address-virtual 00:00:5e:00:
01:01 172.16.1.1/24
    mstpctl-portbpdufilter
vxlan1=yes
    mstpctl-bpduguard
vxlan1=yes
auto vlan10
iface vlan10
    bridge-ports peerlink.10
bond0.10 vxlan10 hyperloopin.10
    bridge-stp on
    address 172.16.10.3/24
    address-virtual 00:00:5e:00:
10:10 172.16.10.1/24
    mstpctl-portbpdufilter
vxlan10=yes
    mstpctl-bpduguard
vxlan10=yes
auto vlan20
iface vlan20
    bridge-ports peerlink.20
bond0.20 vxlan20 hyperloopin.20
    bridge-stp on
    address 172.16.20.3/24
    address-virtual 00:00:5e:00:
20:20 172.16.20.1/24
    mstpctl-portbpdufilter
vxlan20=yes
    mstpctl-bpduguard vxlan20=yes
#vxlan config
auto vxlan1
iface vxlan1
    vxlan-id 1
    vxlan-local-tunnelip
10.0.0.42
auto vxlan10
iface vxlan10
    vxlan-id 10
    vxlan-local-tunnelip
10.0.0.42
auto vxlan20
iface vxlan20
    vxlan-id 20
    vxlan-local-tunnelip
10.0.0.42
# uplinks
auto swp51
iface swp51
auto swp52
iface swp52

```

```
#internet
auto swp44
iface swp44
```

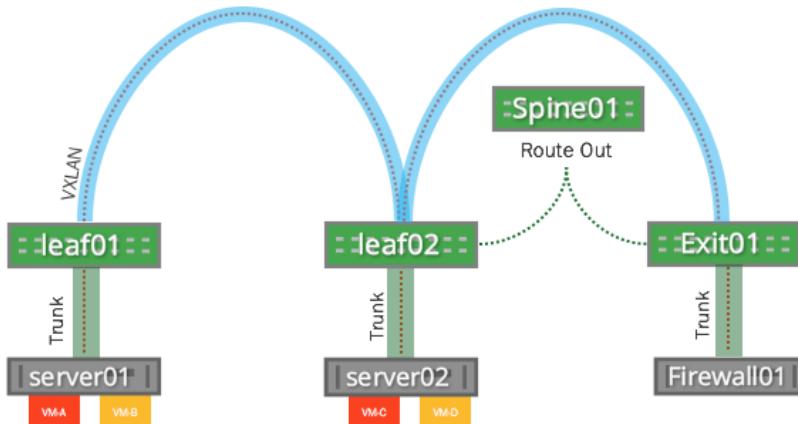
```
#internet
auto swp44
iface swp44
```

## Using Every Pair of Leafs

The configuration for using multiple or all leafs is the same as for [exit leafs \(see page 349\)](#), but with more switch pairs. However, as gateways are present in the form of SVIs, traffic between certain tenants can occur unintentionally. The figure below shows an example of this:



This is a different example than the main demonstration; these servers are single attached to make the diagram simple and easy for clarification.



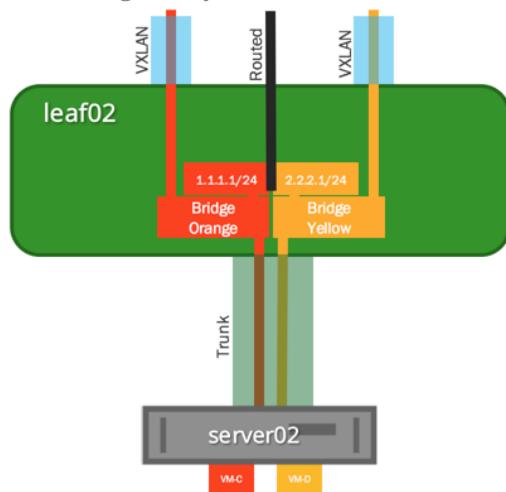
The figure above shows the following:

VM Name	VXLAN	Server
VM-A	Orange	server01
VM-B	Yellow	server01
VM-C	Orange	server02
VM-D	Yellow	server02

- VM-A has a layer 2 adjacency across the VXLAN tunnel to VM-C.
- VM-B has a layer 2 adjacency across the VXLAN tunnel to VM-D.
- All VMs have a layer 2 adjacency to Firewall01.

However, Leaf02 also has a gateway configured (either with a loopback cable or RIOT support). This means that if VM-C wants to talk to VM-D, they could talk directly on the same switch. This could be undesired behavior, depending on the environment and network architecture design, as the path of the packet could be:

VM-C -> VLAN Orange -> Bridge Orange with both an IP address and VXLAN configured. This creates a scenario where traffic has two possible paths — either the VXLAN tunnel, or the gateway. If VM-C chooses to use the gateway 1.1.1.1/24, traffic could bypass the VXLAN altogether, which may be undesired behavior.



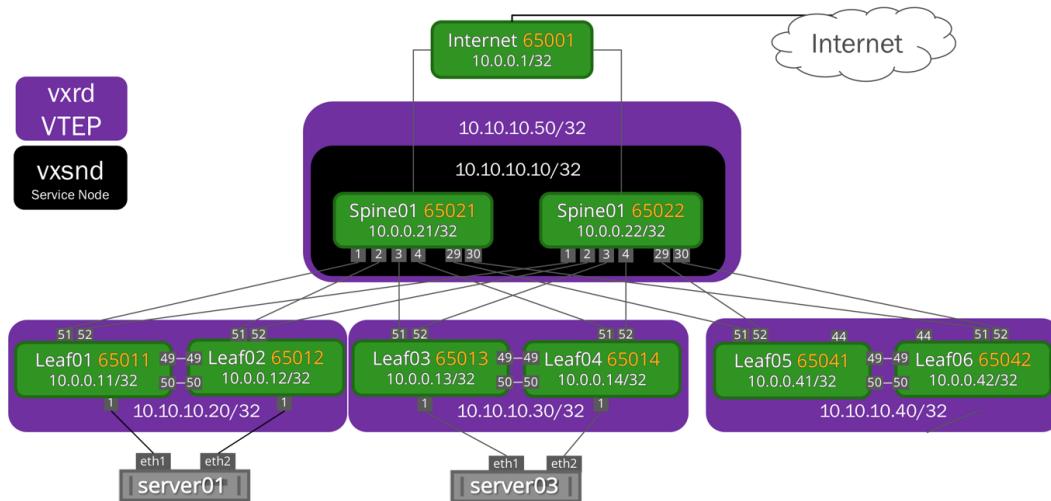
The network architecture may be configured for all traffic to go to the firewall before it is allowed to go to another tenant (in the form of a VXLAN). This can be accomplished in multiple ways, but the two most popular methods are listed below:

- Use [VRF](#) (available on Cumulus Linux 3.0 and newer).
- Force the gateway for the VXLAN to be the firewall's IP address.
  - Enforce this with `cl-acltool` (for example, ACLs and `iptables`).

In the above example, Bridge Orange and Bridge Yellow would be placed into their own VRF so that they have no awareness of the other route table, or `iptables` rules via `cl-acltool` could be enforced to deny any traffic between the 1.1.1.X/24 and 2.2.2.X/24 subnets. This adds some complexity to network configuration and does not always add benefits; as such, exit leafs are often used as the only gateway for a VXLAN.

## ***Using a Pair of Spines***

The `vxdrd` and `vxsnd` services can run on the same switch. This can be on a pair of leafs or on a pair of spines. The main limitation is that only two switches can be in a MLAG pair.


**Spine01:** /etc/network/interfaces snippet

```
auto lo
iface lo inet loopback
  address 10.2.1.21/32
  address 10.10.10.10/32
  address 10.2.1.23/32
  vxrd-src-ip 10.2.1.21
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.50
```

**Spine01:** /etc/vxsnd.conf

```
[common]
svcnode_ip=10.10.10.10
src_ip=10.2.1.23
[vxsnd]
svcnode_peers = 10.2.1.23
10.2.1.24
```

**Spine02:** /etc/network/interfaces snippet

```
auto lo
iface lo inet loopback
  address 10.2.1.22/32
  address 10.10.10.10/32
  address 10.2.1.24/32
  vxrd-src-ip 10.2.1.22
  vxrd-svcnode-ip 10.10.10.10
  clagd-vxlan-anycast-ip
  10.10.10.50
```

**Spine02 :** /etc/vxsnd.conf

```
[common]
svcnode_ip=10.10.10.10
src_ip=10.2.1.24
[vxsnd]
svcnode_peers = 10.2.1.23
10.2.1.24
```

Explanation of variables configured on Spine01:

Variable and Value	Explanation
address 10.2.1. 21/32	This is a statically configured IP address. This is used by vxrd as indicated below by the vxrd-src-ip.

Value	
<b>address</b> 10.10.1 0.10/32	This is a statically configured IP address. This is used by vxsnd.conf for the anycast address.
<b>address</b> 10.2.1. 23/32	This is a statically configured IP address. This is used by vxsnd.conf for the unique IP address.
<b>vxrd-</b> <b>src-ip</b> 10.2.1. 21	The registration daemon ( <b>vxrd</b> ) will bind to the IP address 10.2.1.12.
<b>vxrd-</b> <b>svcnod</b> e - <b>ip</b> 10.10.1 0.10	The service node that the registration node will register to is 10.10.10.10. In this scenario the IP is on the same switch so it will just register to itself. The vxsnd will propagate this information to all other VTEPs so they are aware this switch is also
<b>cldg-</b> <b>vxlan-</b> <b>anycast</b> - <b>ip</b> 10.10.1 0.50	Shared VTEP address for MLAG pair is using 10.10.10.50 virtual IP address

## Considerations for Virtual Topologies Using Cumulus VX

### Node ID

vxrd requires that there is a unique `node_id` for each individual switch. This `node_id` is based off of the first interface's MAC address; when using certain virtual topologies like Vagrant, both Leaf switches within an MLAG pair can generate the same exact unique `node_id`. One of the `node_ids` will then need to be configured manually (or make sure the first interface always has a unique MAC address), as they are not unique.

To see the `node_id` that gets configured by your box, use the `vxrdctl get config` command:

```
cumulus@leaf01$ vxrdctl get config
{
    "concurrency": 1000,
    "config_check_rate": 60,
    "debug": false,
    "eventlet_backdoor_port": 9000,
    "head_rep": true,
    "holdtime": 90,
    "logbackupcount": 14,
    "logdest": "syslog",
    "logfilesize": 512000,
    "loglevel": "INFO",
    "max_packet_size": 1500,
    "node_id": 13,
    "pidfile": "/var/run/vxrd.pid",
    "refresh_rate": 3,
    "src_ip": "10.2.1.50",
    "svcnode_ip": "10.10.10.10",
    "udsfile": "/var/run/vxrd.sock",
    "vxfld_port": 10001
}
```

To set the `node_id` manually:

1. Open `/etc/vxrd.conf` in a text editor.
2. Set the `node_id` value within the `common` section, then save the file:

```
[common]
node_id = 13
```



Ensure that each leaf has a separate `node_id` so that LNV can function correctly.

## Bonds with Vagrant

Bonds (or LACP Etherchannels) will fail to work in a Vagrant setup unless the link is set to promiscuous mode. This is a limitation for virtual topologies only and is not needed on real hardware.

```
auto swp49
iface swp49
    #for vagrant so bonds work correctly
    post-up ip link set $IFACE promisc on

auto swp50
iface swp50
    #for vagrant so bonds work correctly
    post-up ip link set $IFACE promisc on
```

For more information on using Cumulus VX and Vagrant, refer to the [Cumulus VX documentation](#).

## Troubleshooting with LNV Active-Active

In addition to the [troubleshooting for single-attached LNV](#), there is now the MLAG daemon (`clagd`) to consider. The command `clagctl` gives the output of MLAG behavior and any inconsistencies that may arise between a MLAG pair.

```
cumulus@leaf01$ clagctl
The peer is alive
    Our Priority, ID, and Role: 32768 44:38:39:00:00:35 primary
    Peer Priority, ID, and Role: 32768 44:38:39:00:00:36 secondary
        Peer Interface and IP: peerlink.4094 169.254.1.2
        VxLAN Anycast IP: 10.10.10.30
        Backup IP: 10.0.0.14 (inactive)
        System MAC: 44:39:39:ff:40:95

CLAG Interfaces
Our Interface      Peer Interface      CLAG Id
Conflicts          Proto-Down Reason
-----  -----
bond0             bond0              1
-
vxlan20           vxlan20            -
-
vxlan1            vxlan1             -
-
vxlan10           vxlan10            -
```

The additions to normal MLAG behavior are the following:

Output	Explanation
VxLAN Anycast IP: 10.10.10.30	The anycast IP address being shared by the MLAG pair for VTEP termination is in use and is 10.10.10.30.
<b>Conflicts:</b> -	There are no conflicts for this MLAG Interface.
<b>Proto-Down</b> <b>Reason:</b> -	The VXLAN is up and running (there is no Proto-Down).

In the next example the `vxlan-id` on VXLAN10 was switched to the wrong `vxlan-id`. Now when you run the `clagctl` command, you will see that VXLAN10 went down because this switch was the secondary switch and the peer switch took control of VXLAN. The reason code is `vxlan-single` indicating that there is a `vxlan-id` mis-match on VXLAN10.

```
cumulus@leaf02$ clagctl
The peer is alive
  Peer Priority, ID, and Role: 32768 44:38:39:00:00:11 primary
  Our Priority, ID, and Role: 32768 44:38:39:00:00:12 secondary
  Peer Interface and IP: peerlink.4094 169.254.1.1
  VxLAN Anycast IP: 10.10.10.20
                Backup IP: 10.0.0.11 (inactive)
                System MAC: 44:39:39:ff:40:94

CLAG Interfaces
Our Interface      Peer Interface      CLAG Id
Conflicts          Proto-Down        Reason
-----            -----           -----
-----            -----           -----
      bond0        bond0           1
-
      vxlan20      vxlan20         -
-
      vxlan1       vxlan1         -
-
      vxlan10      -             vxlan-single
-
```

## Caveats and Errata

- VLAN-aware bridge mode (see page 330) is not supported for VXLAN active-active mode in this release.
- The VLAN used for the peer link layer 3 subinterface should not be reused for any other interface in the system. A high VLAN ID value is recommended. For more information on VLAN ID ranges, refer here (see page ).
- Active-active mode only works with LNV in this release. Integration with controller-based VXLANs such as VMware NSX and Midokura MidoNet will be supported in the future.

## See Also

- Lightweight Network Virtualization
- LNV Full Example (Single Attached)

## LNV Full Example

Lightweight Network Virtualization (LNV) is a technique for deploying VXLANs (see page 269) without a central controller on bare metal switches. This a full example complete with diagram. Please reference the [Lightweight Network Virtualization chapter \(see page 302\)](#) for more detailed information. This full example uses the **recommended way** of deploying LNV, which is to use Anycast to load balance the service nodes.



LNV is a lightweight controller option. Please [contact Cumulus Networks](#) with your scale requirements and we can make sure this is the right fit for you. There are also other controller options that can work on Cumulus Linux.

## Contents

(Click to expand)

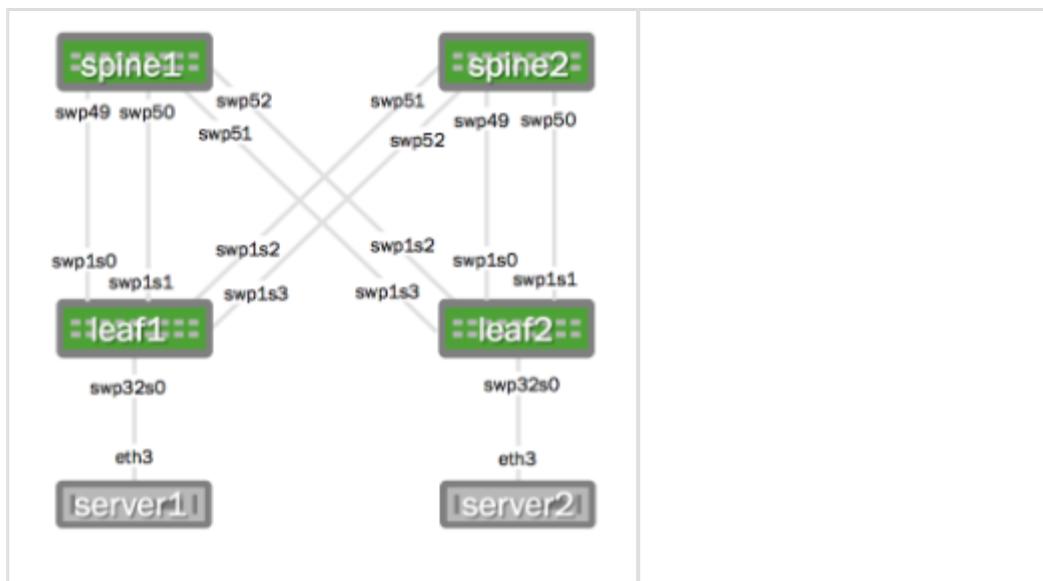
- Contents (see page 363)
- Example LNV Configuration (see page 363)
  - Layer 3 IP Addressing (see page 364)
  - Quagga Configuration (see page 366)
  - Host Configuration (see page 367)
  - Service Node Configuration (see page 369)
- See Also (see page 370)

## Example LNV Configuration

The following images illustrate the configuration:

Physical Cabling Diagram

Network Virtualization Diagram



 Want to try out configuring LNV and don't have a Cumulus Linux switch? Sign up to use the [Cumulus Workbench](#), which has this exact topology.

 Feeling Overwhelmed? Come join a [Cumulus Boot Camp](#) and get instructor-led training!

## Layer 3 IP Addressing

Here is the configuration for the IP addressing information used in this example:

**spine1:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
  address 10.2.1.3/32
  address 10.10.10.10/32

auto eth0
iface eth0 inet dhcp

auto swp49
iface swp49
  address 10.1.1.2/30

auto swp50
iface swp50
  address 10.1.1.6/30
```

**spine2:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
  address 10.2.1.4/32
  address 10.10.10.10/32

auto eth0
iface eth0 inet dhcp

auto swp49
iface swp49
  address 10.1.1.18/30

auto swp50
iface swp50
  address 10.1.1.22/30
```

```
auto swp51
iface swp51
    address 10.1.1.50/30
```

```
auto swp52
iface swp52
    address 10.1.1.54/30
```

#### **leaf1:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
    address 10.2.1.1/32
    vxrd-src-ip 10.2.1.1
    vxrd-svcnode-ip 10.10.10.10
```

```
auto eth0
iface eth0 inet dhcp
```

```
auto swp1s0
iface swp1s0
    address 10.1.1.1/30
```

```
auto swp1s1
iface swp1s1
    address 10.1.1.5/30
```

```
auto swp1s2
iface swp1s2
    address 10.1.1.33/30
```

```
auto swp1s3
iface swp1s3
    address 10.1.1.37/30
```

```
auto vni-10
iface vni-10
    vxlan-id 10
    vxlan-local-tunnelip 10.2.1.1
```

```
auto vni-2000
iface vni-2000
    vxlan-id 2000
    vxlan-local-tunnelip 10.2.1.1
```

```
auto vni-30
iface vni-30
```

```
auto swp51
iface swp51
    address 10.1.1.34/30
```

```
auto swp52
iface swp52
    address 10.1.1.38/30
```

#### **leaf2:** /etc/network/interfaces

```
auto lo
iface lo inet loopback
    address 10.2.1.2/32
    vxrd-src-ip 10.2.1.2
    vxrd-svcnode-ip 10.10.10.10
```

```
auto eth0
iface eth0 inet dhcp
```

```
auto swp1s0
iface swp1s0 inet static
    address 10.1.1.17/30
```

```
auto swp1s1
iface swp1s1 inet static
    address 10.1.1.21/30
```

```
auto swp1s2
iface swp1s2 inet static
    address 10.1.1.49/30
```

```
auto swp1s3
iface swp1s3 inet static
    address 10.1.1.53/30
```

```
auto vni-10
iface vni-10
    vxlan-id 10
    vxlan-local-tunnelip 10.2.1.2
```

```
auto vni-2000
iface vni-2000
    vxlan-id 2000
    vxlan-local-tunnelip 10.2.1.2
```

```
auto vni-30
iface vni-30
```

```

vxlan-id 30
vxlan-local-tunnelip 10.2.1.1

auto br-10
iface br-10
  bridge-ports swp32s0.10 vni-
10

auto br-20
iface br-20
  bridge-ports swp32s0.20 vni-
2000

auto br-30
iface br-30
  bridge-ports swp32s0.30 vni-
30
  
```

```

vxlan-id 30
vxlan-local-tunnelip 10.2.1.2

auto br-10
iface br-10
  bridge-ports swp32s0.10 vni-
10

auto br-20
iface br-20
  bridge-ports swp32s0.20 vni-
2000

auto br-30
iface br-30
  bridge-ports swp32s0.30 vni-
30
  
```

## Quagga Configuration

The service nodes and registration nodes must all be routable between each other. The L3 fabric on Cumulus Linux can either be [BGP](#) (see page 419) or [OSPF](#) (see page 406). In this example, OSPF is used to demonstrate full reachability.

Here is the Quagga configuration using OSPF:

**spine1:** /etc/quagga/Quagga.conf

```

interface lo
  ip ospf area 0.0.0.0
interface swp49
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp50
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp51
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp52
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
  
```

**spine2:** /etc/quagga/Quagga.conf

```

interface lo
  ip ospf area 0.0.0.0
interface swp49
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp50
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp51
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
!
interface swp52
  ip ospf network point-to-
point
  ip ospf area 0.0.0.0
  
```

```
!
!
!
!
!
router-id 10.2.1.3
router ospf
  ospf router-id 10.2.1.3
```

```
!
!
!
!
!
router-id 10.2.1.4
router ospf
  ospf router-id 10.2.1.4
```

**leaf1:** /etc/quagga/Quagga.conf

```
interface lo
  ip ospf area 0.0.0.0
interface swp1s0
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s1
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s2
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s3
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
!
!
!
!
router-id 10.2.1.1
router ospf
  ospf router-id 10.2.1.1
```

**leaf2:** /etc/quagga/Quagga.conf

```
interface lo
  ip ospf area 0.0.0.0
interface swp1s0
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s1
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s2
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
interface swp1s3
  ip ospf network point-to-
  point
  ip ospf area 0.0.0.0
!
!
!
!
!
router-id 10.2.1.2
router ospf
  ospf router-id 10.2.1.2
```

## Host Configuration

In this example, the servers are running Ubuntu 14.04. A trunk must be mapped from server1 and server2 to the respective switch. In Ubuntu this is done with subinterfaces.

**server1**

```
auto eth3.10
iface eth3.10 inet
static
    address 10.10.10.1/24

auto eth3.20
iface eth3.20 inet
static
    address 10.10.20.1/24

auto eth3.30
iface eth3.30 inet
static
    address 10.10.30.1/24
```

**server2**

```
auto eth3.10
iface eth3.10 inet
static
    address 10.10.10.2/24

auto eth3.20
iface eth3.20 inet
static
    address 10.10.20.2/24

auto eth3.30
iface eth3.30 inet
static
    address 10.10.30.2/24
```

## Service Node Configuration

**spine1:/etc/vxsnd.conf**

```
[common]
# Log level is one of DEBUG,
INFO, WARNING, ERROR, CRITICAL
#loglevel = INFO
# Destination for log
message. Can be a file name, '
stdout', or 'syslog'
#logdest = syslog
# log file size in bytes. Used
when logdest is a file
#logfilesize = 512000
# maximum number of log files
stored on disk. Used when
logdest is a file
#logbackupcount = 14
# The file to write the pid.
If using monit, this must
match the one
# in the vxsnd.rc
#pidfile = /var/run/vxsnd.pid
# The file name for the unix
domain socket used for mgmt.
#udsfile = /var/run/vxsnd.sock
# UDP port for vxfld control
messages
#vxfld_port = 10001
# This is the address to which
registration daemons send
control messages for
# registration and/or BUM
packets for replication
svcnode_ip = 10.10.10.10
# Holdtime (in seconds) for
soft state. It is used when
sending a
# register msg to peers in
response to learning a <vni,
addr> from a
# VXLAN data pkt
#holdtime = 90
# Local IP address to bind to f
or receiving inter-vxsnd
control traffic
src_ip = 10.2.1.3
```

**spine2:/etc/vxsnd.conf**

```
[common]
# Log level is one of DEBUG,
INFO, WARNING, ERROR, CRITICAL
#loglevel = INFO
# Destination for log
message. Can be a file name, '
stdout', or 'syslog'
#logdest = syslog
# log file size in bytes. Used
when logdest is a file
#logfilesize = 512000
# maximum number of log files
stored on disk. Used when
logdest is a file
#logbackupcount = 14
# The file to write the pid.
If using monit, this must
match the one
# in the vxsnd.rc
#pidfile = /var/run/vxsnd.pid
# The file name for the unix
domain socket used for mgmt.
#udsfile = /var/run/vxsnd.sock
# UDP port for vxfld control
messages
#vxfld_port = 10001
# This is the address to which
registration daemons send
control messages for
# registration and/or BUM
packets for replication
svcnode_ip = 10.10.10.10
# Holdtime (in seconds) for
soft state. It is used when
sending a
# register msg to peers in
response to learning a <vni,
addr> from a
# VXLAN data pkt
#holdtime = 90
# Local IP address to bind to f
or receiving inter-vxsnd
control traffic
src_ip = 10.2.1.4
```

```
[vxsnd]
# Space separated list of IP
addresses of vxsnd to share
state with
svcnode_peers = 10.2.1.4
# When set to true, the
service node will listen for
vxlan data traffic
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#enable_vxlan_listen = true
# When set to true, the
svcnode_ip will be installed
on the loopback
# interface, and it will be
withdrawn when the vxsnd is no
longer in
# service. If set to true,
the svcnode_ip configuration
# variable must be defined.
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#install_svcnode_ip = false
# Seconds to wait before
checking the database to age
out stale entries
#age_check = 90
```

```
[vxsnd]
# Space separated list of IP
addresses of vxsnd to share
state with
svcnode_peers = 10.2.1.3
# When set to true, the
service node will listen for
vxlan data traffic
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#enable_vxlan_listen = true
# When set to true, the
svcnode_ip will be installed
on the loopback
# interface, and it will be
withdrawn when the vxsnd is no
longer in
# service. If set to true,
the svcnode_ip configuration
# variable must be defined.
# Note: Use 1, yes, true, or
on, for True and 0, no, false,
or off,
# for False
#install_svcnode_ip = false
# Seconds to wait before
checking the database to age
out stale entries
#age_check = 90
```

## See Also

- <https://tools.ietf.org/html/rfc7348>
- <http://en.wikipedia.org/wiki/Anycast>
- Detailed LNV Configuration Guide (see page 302)
- Cumulus Networks Training

## Static MAC Bindings with VXLAN

Cumulus Linux includes native Linux VXLAN kernel support.

## Contents

(Click to expand)

- [Contents \(see page 370\)](#)

- Requirements (see page 371)
- Example VXLAN Configuration (see page 371)
- Configuring the Static MAC Bindings VXLAN (see page 372)
- Troubleshooting VXLANs in Cumulus Linux (see page 375)

## Requirements

A VXLAN configuration requires a switch with a Tomahawk, Trident II+ or Trident II chipset running Cumulus Linux 2.0 or later.

For a basic VXLAN configuration, you should ensure that:

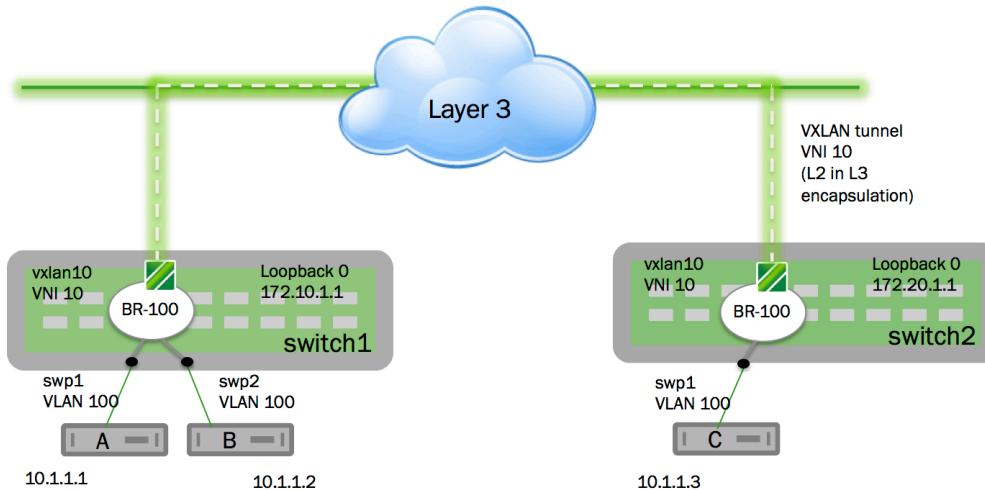
- The VXLAN has a network identifier (VNI); do not use 0 or 16777215 as the VNI ID, as they are reserved values under Cumulus Linux.
- The VXLAN link and local interfaces are added to bridge to create the association between port, VLAN and VXLAN instance.
- Each bridge on the switch has only one VXLAN interface. Cumulus Linux does not support more than one VXLAN link in a bridge; however a switch can have multiple bridges.



VXLANs are not supported on switches with [Spectrum ASICs](#).

## Example VXLAN Configuration

Consider the following example:



Preconfiguring remote MAC addresses does not scale. A better solution is to use the Cumulus Networks [Lightweight Network Virtualization](#) feature, or a controller-based option like [Midokura MidoNet](#) and [OpenStack](#) or [VMware NSX](#).

## Configuring the Static MAC Bindings VXLAN

To configure the example illustrated above, edit `/etc/network/interfaces` with a text editor like vi, nano or zile.

Add the following configuration to the `/etc/network/interfaces` file on switch1:

```
auto vtep1000
iface vtep1000
  vxlan-id 1000
  vxlan-local-tunnelip 172.10.1.1

auto br-100
iface br-100
  bridge-ports swp1.100 swp2.100 vtep1000
  post-up bridge fdb add 0:00:10:00:00:0C dev vtep1000 dst 172.20.1.1 vni 1000
```

Add the following configuration to the `/etc/network/interfaces` file on switch2:

```
auto vtep1000
iface vtep1000
  vxlan-id 1000
  vxlan-local-tunnelip 172.20.1.1

auto br-100
iface br-100
  bridge-ports swp1.100 swp2.100 vtep1000
  post-up bridge fdb add 00:00:10:00:00:0A dev vtep1000 dst 172.10.1.1 vni 1000
  post-up bridge fdb add 00:00:10:00:00:0B dev vtep1000 dst 172.10.1.1 vni 1000
```

### Runtime Configuration (Advanced)



A runtime configuration is non-persistent, which means the configuration you create here does not persist after you reboot the switch.

In general, to configure a VXLAN in Cumulus Linux without a controller, run the following commands in a terminal connected to the switch:

1. Create a VXLAN link:

```
cumulus@switch1:~$ sudo ip link add <name> type vxlan id <vni> local
<ip addr> [group <mcast group address>] [no] nolearning [ttl] [tos]
[dev] [port MIN MAX] [ageing <value>] remote [replicator addr]
```



If you are specifying `ageing`, you **must** specify the replicator (`remote`) .

2. Add a VXLAN link to a bridge:

```
cumulus@switch1:~$ sudo brctl addif br-vxlan <name>
```

3. Install a static MAC binding to a remote tunnel IP:

```
cumulus@switch1:~$ sudo bridge fdb add <mac addr> dev <device> dst <ip addr> vni <vni> port <port> via <device>
```

4. Show VXLAN link and FDB:

```
cumulus@switch1:~$ ip -d link show
```

```
cumulus@switch1:~$ bridge fdb show
```

To create a runtime configuration that matches the image above, do the following:

1. Configure hosts A and B as part of the same tenant as C (VNI 10) on switch1. Hosts A and B are part of VLAN 100. To configure the VTEP interface with VNI 10, run the following commands in a terminal connected to switch1 running Cumulus Linux:

```
cumulus@switch1:~$ sudo ip link add link swp1 name swp1.100 type vlan id 100
cumulus@switch1:~$ sudo ip link add link swp2 name swp2.100 type vlan id 100
cumulus@switch1:~$ sudo ip link add vtep1000 type vxlan id 10 local 172.10.1.1 nolearning
cumulus@switch1:~$ sudo ip link set swp1 up
cumulus@switch1:~$ sudo ip link set swp2 up
cumulus@switch1:~$ sudo ip link set vtep1000 up
```

2. Configure VLAN 100 and VTEP 1000 to be part of the same bridge br-100 on switch1:

```
cumulus@switch1:~$ sudo brctl addbr br-100
cumulus@switch1:~$ sudo ip link set br-100 up
```

```
cumulus@switch1:~$ sudo brctl addif br-100 swp1.100 swp2.100
cumulus@switch1:~$ sudo brctl addif br-100 vtep1000
```

3. Install a static MAC binding to a remote tunnel IP, assuming the MAC address for host C is 00:00:10:00:00:0C:

```
cumulus@switch1:~$ sudo bridge fdb add 00:00:10:00:00:0C dev vtep1000
dst 172.20.1.1
```

4. Configure host C as part of the same tenant as hosts A and B on switch2:

```
cumulus@switch2:~$ sudo ip link add link swp1 name swp1.100 type vlan
id 100
cumulus@switch2:~$ sudo ip link add name vtep1000 type vxlan id 10
local 172.20.1.1 nolearning
cumulus@switch2:~$ sudo ip link set swp1 up
cumulus@switch2:~$ sudo ip link set vtep1000 up
```

5. Configure VLAN 100 and VTEP 1000 to be part of the same bridge br-100 on switch2:

```
cumulus@switch2:~$ sudo brctl addbr br-100
cumulus@switch2:~$ sudo ip link set br-100 up
cumulus@switch2:~$ sudo brctl addif br-100 swp1.100
cumulus@switch2:~$ sudo brctl addif br-100 vtep1000
```

6. Install a static MAC binding to a remote tunnel IP on switch2, assuming the MAC address for host A is 00:00:10:00:00:0A and the MAC address for host B is 00:00:10:00:00:0B:

```
cumulus@switch2:~$ sudo bridge fdb add 00:00:10:00:00:0A dev vtep1000
dst 172.10.1.1
cumulus@switch2:~$ sudo bridge fdb add 00:00:10:00:00:0B dev vtep1000
dst 172.10.1.1
```

7. Verify the configuration on switch1, then on switch2:

```
cumulus@switch1:~$ ip -d link show
cumulus@switch1:~$ bridge fdb show
```

```
cumulus@switch2:~$ ip -d link show
cumulus@switch2:~$ bridge fdb show
```

- Set the static arp for hosts B and C on host A:

```
root@hostA:~# sudo arp -s 10.1.1.3 00:00:10:00:00:0C
```

- Set the static arp for hosts A and C on host B:

```
root@hostB:~# sudo arp -s 10.1.1.3 00:00:10:00:00:0C
```

- Set the static arp for hosts A and B on host C:

```
root@hostC:~# arp -s 10.1.1.1 00:00:10:00:00:0A
root@hostC:~# arp -s 10.1.1.2 00:00:10:00:00:0B
```

## Troubleshooting VXLANS in Cumulus Linux

Use the following commands to troubleshoot issues on the switch:

- brctl show:** Verifies the VXLAN configuration in a bridge:

```
cumulus@switch:~$ brctl show
bridge name      bridge id          STP enabled
interfaces
br-vxln100      8000.44383900480d    no
swp2s0.100
00
                                         swp2s1.1
                                         vxln100
```

- bridge fdb show:** Displays the list of MAC addresses in an FDB:

```
cumulus@switch1:~$ bridge fdb show
52:54:00:ae:2a:e0 dev vxln100 dst 172.16.21.150 self permanent
d2:ca:78:bb:7c:9b dev vxln100 permanent
90:e2:ba:3f:ce:34 dev swp2s1.100
90:e2:ba:3f:ce:35 dev swp2s0.100
44:38:39:00:48:0e dev swp2s1.100 permanent
44:38:39:00:48:0d dev swp2s0.100 permanent
```

- `ip -d link show`: Displays information about the VXLAN link:

```
cumulus@switch1:~$ ip -d link show vxln100
71: vxln100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    master br-vxln100 state UP
        mode DEFAULT group default
        link/ether d2:ca:78:bb:7c:9b brd ff:ff:ff:ff:ff:ff
    promiscuity 1
        vxlan id 100 local 36.0.0.11 srcport 0 0 dstport 4789 ageing
        300
        bridge_slave state forwarding priority 32 cost 100 hairpin
        off guard off root_block off
        fastleave off learning on flood on port_id 0x8001 port_no 0x1
        designated_port 32769
        designated_cost 0 designated_bridge 8000.c4:54:44:bd:1:71 des
        ignated_root 8000.c4:54:44:bd:1:71
        hold_timer      0.00 message_age_timer      0.00
        forward_delay_timer      0.00
        topology_change_ack 0 config_pending 0 proxy_arp off
        proxy_arp_wifi off mcast_router 1
        mcast_fast_leave off addrgenmode eui64
```

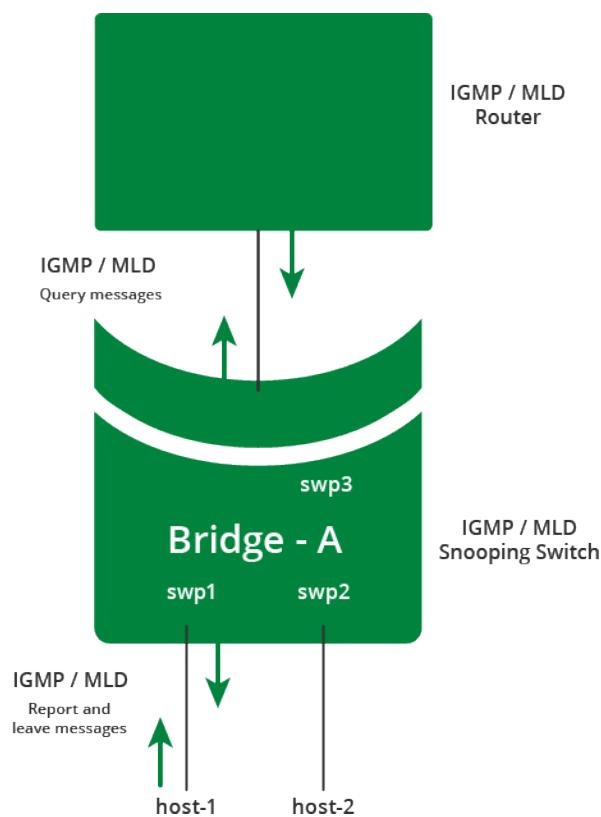
## IGMP and MLD Snooping

IGMP (Internet Group Management Protocol) and MLD (Multicast Listener Discovery) snooping functionality is implemented in the bridge driver in the Cumulus Linux kernel and are enabled by default. IGMP snooping processes IGMP v1/v2/v3 reports received on a bridge port in a bridge to identify the hosts which would like to receive multicast traffic destined to that group.

When an IGMPv2 leave message is received, a group specific query is sent to identify if there are any other hosts interested in that group, before the group is deleted.

An IGMP query message received on a port is used to identify the port that is connected to a router and is interested in receiving multicast traffic.

MLD snooping processes MLD v1/v2 reports, queries and v1 done messages for IPv6 groups. If IGMP or MLD snooping is disabled, multicast traffic will be flooded to all the bridge ports in the bridge. The multicast group IP address is mapped to a multicast MAC address and a forwarding entry is created with a list of ports interested in receiving multicast traffic destined to that group.



## Contents

This chapter covers ...

- Commands (see page 377)
- Configuring IGMP/MLD Querier (see page 377)
- Disabling IGMP and MLD Snooping (see page 378)
- Debugging IGMP/MLD Snooping (see page 379)
- Configuration Files (see page 380)
- Man Pages (see page 380)
- Useful Links (see page 381)

## Commands

- brctl
- bridge

## Configuring IGMP/MLD Querier

If no multicast router is sending queries to configure IGMP/MLD querier on the switch, you can add a configuration similar to the following in `/etc/network/interfaces`. To enable IGMP and MLD snooping for a bridge, set `bridge-mcquerier` to 1 in the bridge stanza. By default, the source IP address of IGMP queries is 0.0.0.0. To set the source IP address of the queries to be the bridge IP address, configure `bridge-mcqifaddr 1`.

For an explanation of the relevant parameters, see the `ifupdown-addons-interfaces` man page.

For a bridge in [traditional mode](#), use a configuration like the following:

```
auto br0
iface br0
    address 192.0.2.10/24
    bridge-ports swp1 swp2 swp3
    bridge-vlan-aware no
    bridge-mcquerier 1
    bridge-mcqifaddr 1
```

For a [VLAN-aware bridge](#), use a configuration like the following:

```
auto br0.100
vlan br0.100
    bridge-igmp-querier-src 123.1.1.1

auto br0
iface br0
    bridge-ports swp1 swp2 swp3
    bridge-vlan-aware yes
    bridge-vids 100 200
    bridge-pvid 1
    bridge-mcquerier 1
```

For a VLAN-aware bridge, like br0 in the above example, to enable querier functionality for VLAN 100 in the bridge, set `bridge-mcquerier` to 1 in the bridge stanza and set `bridge-igmp-querier-src` to 123.1.1.1 in the br0.100 stanza.

You can specify a range of VLANs as well. For example:

```
auto bridge.[1-200]
vlan bridge.[1-200]
    bridge-igmp-querier-src 123.1.1.1
```

## ***Disabling IGMP and MLD Snooping***

To disable IGMP and MLD snooping, set `bridge-mcsnoop` to 0 in `/etc/network/interfaces`.

```
auto br0
iface br0
    bridge-ports swp1 swp2 swp3
    bridge-mcsnoop 0
    bridge-vlan-aware yes
    bridge-vids 100 200
    bridge-pvid 1
    bridge-mcquerier 1
```

## Debugging IGMP/MLD Snooping

To get the IGMP/MLD snooping bridge state, run `brctl showstp <bridge>`:

```
cumulus@switch:~$ sudo brctl showstp br0
br0
bridge id          8000.7072cf8c272c
designated root    8000.7072cf8c272c
root port          0
cost               0
max age            20.00
age                20.00
hello time         2.00
time               2.00
forward delay      15.00
delay               15.00
ageing time        300.00
hello timer        0.00
timer               0.00
topology change timer 0.00
timer               263.70
hash elasticity    4096
max                 4096
mc last member count 2
count               2
mc router           1
snooping             1
mc last member timer 1.00
timer               260.00
mc querier timer   255.00
interval            125.00
mc response interval 10.00
interval            31.25
mc querier           0
ifaddr               0
flags

swp1 (1)
port id            8001
forwarding
designated root    8000.7072cf8c272c
cost               2
designated bridge  8000.7072cf8c272c
timer               0.00
designated port    8001
timer               0.00
designated cost     0
timer               0.00
path
message age
forward delay
hold
```

```

mc router          1           mc fast
leave             0
flags

swp2 (2)
  port id        8002         state
forwarding
  designated root 8000.7072cf8c272c path
cost              2
  designated bridge 8000.7072cf8c272c message age
timer            0.00
  designated port 8002       forward delay
timer            0.00
  designated cost 0           hold
timer            0.00
  mc router       1           mc fast
leave             0
flags

swp3 (3)
  port id        8003         state
forwarding
  designated root 8000.7072cf8c272c path
cost              2
  designated bridge 8000.7072cf8c272c message age
timer            0.00
  designated port 8003       forward delay
timer            8.98
  designated cost 0           hold
timer            0.00
  mc router       1           mc fast
leave             0
flags

```

To get the groups and bridge port state, use the `bridge mdb show` command. To display router ports and group information use the `bridge -d -s mdb show` command:

```

cumulus@switch:~$ sudo bridge -d -s mdb show
dev br0 port swp2 grp 234.10.10.10 temp 241.67
dev br0 port swp1 grp 238.39.20.86 permanent 0.00
dev br0 port swp1 grp 234.1.1.1 temp 235.43
dev br0 port swp2 grp ff1a::9 permanent 0.00
router ports on br0: swp3

```

## Configuration Files

- /etc/network/interfaces

## ***Man Pages***

- brctl(8)
- bridge(8)
- ifupdown-addons-interfaces(5)

## ***Useful Links***

- <http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge#Snooping>
- <https://tools.ietf.org/html/rfc4541>
- [http://en.wikipedia.org/wiki/IGMP\\_snooping](http://en.wikipedia.org/wiki/IGMP_snooping)
- <http://tools.ietf.org/rfc/rfc2236.txt>
- <http://tools.ietf.org/html/rfc3376>
- <http://tools.ietf.org/search/rfc2710>
- <http://tools.ietf.org/html/rfc3810>

# Layer 3 Features

## Routing

This chapter discusses routing on switches running Cumulus Linux.

### Contents

(Click to expand)

- [Contents \(see page 382\)](#)
- [Commands \(see page 382\)](#)
- [Static Routing via ip route \(see page 382\)
  - \[Persistently Adding a Static Route \\(see page 384\\)\]\(#\)](#)
- [Static Routing via quagga \(see page 384\)
  - \[Persistent Configuration \\(see page 386\\)\]\(#\)](#)
- [Supported Route Table Entries \(see page 387\)](#)
- [Configuration Files \(see page 387\)](#)
- [Useful Links \(see page 387\)](#)
- [Caveats and Errata \(see page 387\)](#)

### Commands

- [ip route](#)

### Static Routing via ip route

The `ip route` command allows manipulating the kernel routing table directly from the Linux shell. See `man ip(8)` for details. `quagga` monitors the kernel routing table changes and updates its own routing table accordingly.

To display the routing table:

```
cumulus@switch:~$ ip route show
default via 10.0.1.2 dev eth0
10.0.1.0/24 dev eth0  proto kernel  scope link  src 10.0.1.52
192.0.2.0/24 dev swp1  proto kernel  scope link  src 192.0.2.12
192.0.2.10/24 via 192.0.2.1 dev swp1  proto zebra  metric 20
192.0.2.20/24  proto zebra  metric 20
    nexthop via 192.0.2.1  dev swp1 weight 1
    nexthop via 192.0.2.2  dev swp2 weight 1
192.0.2.30/24 via 192.0.2.1 dev swp1  proto zebra  metric 20
192.0.2.40/24 dev swp2  proto kernel  scope link  src 192.0.2.42
```

```

192.0.2.50/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.60/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.70/24 proto zebra metric 30
    nexthop via 192.0.2.1 dev swp1 weight 1
    nexthop via 192.0.2.2 dev swp2 weight 1
198.51.100.0/24 dev swp3 proto kernel scope link src 198.51.100.1
198.51.100.10/24 dev swp4 proto kernel scope link src 198.51.100.11
198.51.100.20/24 dev br0 proto kernel scope link src 198.51.100.21

```

To add a static route (does not persist across reboots):

```

cumulus@switch:~$ sudo ip route add 203.0.113.0/24 via 198.51.100.2
cumulus@switch:~$ ip route
default via 10.0.1.2 dev eth0
10.0.1.0/24 dev eth0 proto kernel scope link src 10.0.1.52
192.0.2.0/24 dev swp1 proto kernel scope link src 192.0.2.12
192.0.2.10/24 via 192.0.2.1 dev swp1 proto zebra metric 20
192.0.2.20/24 proto zebra metric 20
    nexthop via 192.0.2.1 dev swp1 weight 1
    nexthop via 192.0.2.2 dev swp2 weight 1
192.0.2.30/24 via 192.0.2.1 dev swp1 proto zebra metric 20
192.0.2.40/24 dev swp2 proto kernel scope link src 192.0.2.42
192.0.2.50/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.60/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.70/24 proto zebra metric 30
    nexthop via 192.0.2.1 dev swp1 weight 1
    nexthop via 192.0.2.2 dev swp2 weight 1
198.51.100.0/24 dev swp3 proto kernel scope link src 198.51.100.1
198.51.100.10/24 dev swp4 proto kernel scope link src 198.51.100.11
198.51.100.20/24 dev br0 proto kernel scope link src 198.51.100.21
203.0.113.0/24 via 198.51.100.2 dev swp3

```

To delete a static route (does not persist across reboots):

```

cumulus@switch:~$ sudo ip route del 203.0.113.0/24
cumulus@switch:~$ ip route
default via 10.0.1.2 dev eth0
10.0.1.0/24 dev eth0 proto kernel scope link src 10.0.1.52
192.0.2.0/24 dev swp1 proto kernel scope link src 192.0.2.12
192.0.2.10/24 via 192.0.2.1 dev swp1 proto zebra metric 20
192.0.2.20/24 proto zebra metric 20
    nexthop via 192.0.2.1 dev swp1 weight 1

```

```

nexthop via 192.0.2.2 dev swp2 weight 1
192.0.2.30/24 via 192.0.2.1 dev swp1 proto zebra metric 20
192.0.2.40/24 dev swp2 proto kernel scope link src 192.0.2.42
192.0.2.50/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.60/24 via 192.0.2.2 dev swp2 proto zebra metric 20
192.0.2.70/24 proto zebra metric 30
    nexthop via 192.0.2.1 dev swp1 weight 1
    nexthop via 192.0.2.2 dev swp2 weight 1
198.51.100.0/24 dev swp3 proto kernel scope link src 198.51.100.1
198.51.100.10/24 dev swp4 proto kernel scope link src 198.51.100.11
198.51.100.20/24 dev br0 proto kernel scope link src 198.51.100.21

```

## Persistently Adding a Static Route

A static route can be persistently added by adding `up ip route add ..` into `/etc/network/interfaces`. For example:

```

cumulus@switch:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

auto swp3
iface swp3
    address 198.51.100.1/24
    up ip route add 203.0.113.0/24 via 198.51.100.2

```



Notice the simpler configuration of `swp3` due to `ifupdown2`. For more information, see [Configuring Network Interfaces with ifupdown](#) (see page 134).

## Static Routing via quagga

Static routes can also be managed via the `quagga` CLI. The routes are added to the `quagga` routing table, and then will be updated into the kernel routing table as well.

To add a static route (does not persist across reboot):

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

switch# conf t
switch(config)# ip route 203.0.113.0/24 198.51.100.2
switch(config)# end
switch# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.0.1.2, eth0
C>* 10.0.1.0/24 is directly connected, eth0
O 192.0.2.0/24 [110/10] is directly connected, swp1, 00:13:25
C>* 192.0.2.0/24 is directly connected, swp1
O>* 192.0.2.10/24 [110/20] via 192.0.2.1, swp1, 00:13:09
O>* 192.0.2.20/24 [110/20] via 192.0.2.1, swp1, 00:13:09
      *           via 192.0.2.41, swp2, 00:13:09
O>* 192.0.2.30/24 [110/20] via 192.0.2.1, swp1, 00:13:09
O 192.0.2.40/24 [110/10] is directly connected, swp2, 00:13:25
C>* 192.0.2.40/24 is directly connected, swp2
O>* 192.0.2.50/24 [110/20] via 192.0.2.41, swp2, 00:13:09
O>* 192.0.2.60/24 [110/20] via 192.0.2.41, swp2, 00:13:09
O>* 192.0.2.70/24 [110/30] via 192.0.2.1, swp1, 00:13:09
      *           via 192.0.2.41, swp2, 00:13:09
O 198.51.100.0/24 [110/10] is directly connected, swp3, 00:13:22
C>* 198.51.100.0/24 is directly connected, swp3
O 198.51.100.10/24 [110/10] is directly connected, swp4, 00:13:22
C>* 198.51.100.10/24 is directly connected, swp4
O 198.51.100.20/24 [110/10] is directly connected, br0, 00:13:22
C>* 198.51.100.20/24 is directly connected, br0
S>* 203.0.113.0/24 [1/0] via 198.51.100.2, swp3
C>* 127.0.0.0/8 is directly connected, lo
```

To delete a static route (does not persist across reboot):

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
switch# conf t
switch(config)# no ip route 203.0.113.0/24 198.51.100.2
switch(config)# end
switch# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.0.1.2, eth0
C>* 10.0.1.0/24 is directly connected, eth0
O   192.0.2.0/24 [110/10] is directly connected, swp1, 00:13:55
C>* 192.0.2.0/24 is directly connected, swp1
O>* 192.0.2.10/24 [110/20] via 11.0.0.1, swp1, 00:13:39
O>* 192.0.2.20/24 [110/20] via 11.0.0.1, swp1, 00:13:39
      *           via 11.0.4.1, swp2, 00:13:39
O>* 192.0.2.30/24 [110/20] via 11.0.0.1, swp1, 00:13:39
O   192.0.2.40/24 [110/10] is directly connected, swp2, 00:13:55
C>* 192.0.2.40/24 is directly connected, swp2
O>* 192.0.2.50/24 [110/20] via 11.0.4.1, swp2, 00:13:39
O>* 192.0.2.60/24 [110/20] via 11.0.4.1, swp2, 00:13:39
O>* 192.0.2.70/24 [110/30] via 11.0.0.1, swp1, 00:13:39
      *           via 11.0.4.1, swp2, 00:13:39
O   198.51.100.0/24 [110/10] is directly connected, swp3, 00:13:52
C>* 198.51.100.0/24 is directly connected, swp3
O   198.51.100.10/24 [110/10] is directly connected, swp4, 00:13:52
C>* 198.51.100.10/24 is directly connected, swp4
O   198.51.100.20/24 [110/10] is directly connected, br0, 00:13:52
C>* 198.51.100.20/24 is directly connected, br0
C>* 127.0.0.0/8 is directly connected, lo
switch#
```

## Persistent Configuration

From the quagga CLI, the running configuration can be saved so it persists between reboots:

```
switch# write mem
Configuration saved to /etc/quagga/zebra.conf
switch# end
```

## ***Supported Route Table Entries***

Cumulus Linux supports different numbers of route entries, depending upon your switch platform (such as Spectrum, Tomahawk or Trident II; see the [HCL](#)) and whether the routes are IPv4 or IPv6.

In addition, switches on the Tomahawk, Trident II+ and Trident II platforms are configured to manage route table entries using Algorithm Longest Prefix Match (ALPM). In ALPM mode, the hardware can store significantly more route entries.

Following are the number of routes supported on Spectrum switches:

- 256K IPv4 routes
- 128K IPv6 routes
- 256K total routes (both IPv4 and IPv6)

Following are the number of routes supported on Tomahawk switches with ALPM:

- 32K IPv4 routes
- 8K IPv6 routes
- 32K total routes (both IPv4 and IPv6)

Following are the number of routes supported on Trident II+ and Trident II switches with ALPM:

- 32K IPv4 routes
- 16K IPv6 routes
- 32K total routes (both IPv4 and IPv6)

Following are the number of routes supported on Trident and Trident+ switches:

- 16K IPv4 routes
- 8K IPv6 routes
- 16K total routes (both IPv4 and IPv6)

## ***Configuration Files***

- /etc/network/interfaces
- /etc/quagga/zebra.conf

## ***Useful Links***

- <http://linux-ip.net/html/tools-ip-route.html>
- <http://www.nongnu.org/quagga/docs/docs-info.html#Static-Route-Commands>

## ***Caveats and Errata***

- Static routes added via **quagga** can be deleted via Linux shell. This operation, while possible, should be avoided. Routes added by **quagga** should only be deleted by **quagga**, otherwise **quagga** might not be able to clean up all its internal state completely and incorrect routing can occur as a result.

## ***Introduction to Routing Protocols***

This chapter discusses the various routing protocols, and how to configure them.

## Contents

(Click to expand)

- [Contents \(see page 388\)](#)
- [Defining Routing Protocols \(see page 388\)](#)
- [Configuring Routing Protocols \(see page 388\)](#)
- [Protocol Tuning \(see page 388\)](#)
- [Configuration Files \(see page 389\)](#)

## Defining Routing Protocols

A *routing protocol* dynamically computes reachability between various end points. This enables communication to work around link and node failures, and additions and withdrawals of various addresses.

*IP routing protocols* are typically distributed; that is, an instance of the routing protocol runs on each of the routers in a network.



Cumulus Linux does **not** support running multiple instances of the same protocol on a router.

*Distributed routing protocols* compute reachability between end points by disseminating relevant information and running a routing algorithm on this information to determine the routes to each end station. To scale the amount of information that needs to be exchanged, routes are computed on address prefixes rather than on every end point address.

## Configuring Routing Protocols

A routing protocol needs to know three pieces of information, at a minimum:

- Who am I (my identity)
- To whom to disseminate information
- What to disseminate

Most routing protocols use the concept of a router ID to identify a node. Different routing protocols answer the last two questions differently.

The way they answer these questions affects the network design and thereby configuration. For example, in a link-state protocol such as OSPF (see [Open Shortest Path First \(OSPF\) Protocol \(see page 406\)](#)) or IS-IS, complete local information (links and attached address prefixes) about a node is disseminated to every other node in the network. Since the state that a node has to keep grows rapidly in such a case, link-state protocols typically limit the number of nodes that communicate this way. They allow for bigger networks to be built by breaking up a network into a set of smaller subnetworks (which are called areas or levels), and by advertising summarized information about an area to other areas.

Besides the two critical pieces of information mentioned above, protocols have other parameters that can be configured. These are usually specific to each protocol.

## Protocol Tuning

Most protocols provide certain tunable parameters that are specific to convergence during changes.

Wikipedia defines [convergence](#) as the “state of a set of routers that have the same topological information about the network in which they operate”. It is imperative that the routers in a network have the same topological state for the proper functioning of a network. Without this, traffic can be blackholed, and thus not reach its destination. It is normal for different routers to have differing topological states during changes, but this difference should vanish as the routers exchange information about the change and recompute the forwarding paths. Different protocols converge at different speeds in the presence of changes.

A key factor that governs how quickly a routing protocol converges is the time it takes to detect the change. For example, how quickly can a routing protocol be expected to act when there is a link failure. Routing protocols classify changes into two kinds: hard changes such as link failures, and soft changes such as a peer dying silently. They’re classified differently because protocols provide different mechanisms for dealing with these failures.

It is important to configure the protocols to be notified immediately on link changes. This is also true when a node goes down, causing all of its links to go down.

Even if a link doesn’t fail, a routing peer can crash. This causes that router to usually delete the routes it has computed or worse, it makes that router impervious to changes in the network, causing it to go out of sync with the other routers in the network because it no longer shares the same topological information as its peers.

The most common way to detect a protocol peer dying is to detect the absence of a heartbeat. All routing protocols send a heartbeat (or “hello”) packet periodically. When a node does not see a consecutive set of these hello packets from a peer, it declares its peer dead and informs other routers in the network about this. The period of each heartbeat and the number of heartbeats that need to be missed before a peer is declared dead are two popular configurable parameters.

If you configure these timers very low, the network can quickly descend into instability under stressful conditions when a router is not able to keep sending the heartbeats quickly as it is busy computing routing state; or the traffic is so much that the hellos get lost. Alternately, configuring this timer to very high values also causes blackholing of communication because it takes much longer to detect peer failures. Usually, the default values initialized within each protocol are good enough for most networks. Cumulus Networks recommends you do not adjust these settings.

## Configuration Files

- /etc/quagga/daemons

## Network Topology

In computer networks, *topology* refers to the structure of interconnecting various nodes. Some commonly used topologies in networks are star, hub and spoke, leaf and spine, and broadcast.

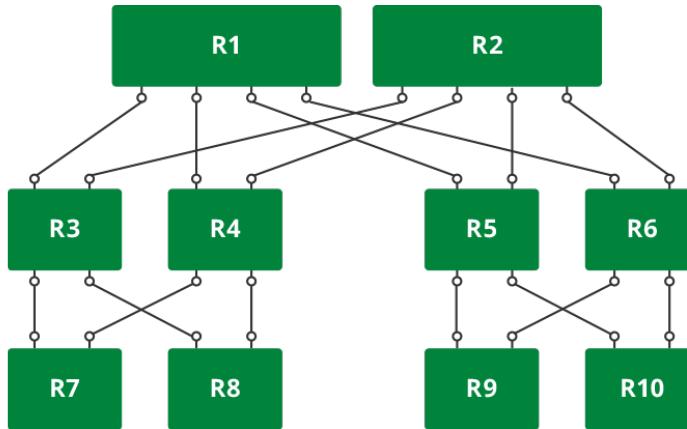
## Contents

(Click to expand)

- [Contents \(see page 389\)](#)
- [Clos Topologies \(see page 389\)](#)
- [Over-Subscribed and Non-Blocking Configurations \(see page 390\)](#)
- [Containing the Failure Domain \(see page 390\)](#)
- [Load Balancing \(see page 391\)](#)

## Clos Topologies

In the vast majority of modern data centers, **Clos or fat tree topology** is very popular. This topology is shown in the figure below. It is also commonly referred to as leaf-spine topology. We shall use this topology throughout the routing protocol guide.



This topology allows the building of networks of varying size using nodes of different port counts and/or by increasing the tiers. The picture above is a three-tiered Clos network. We number the tiers from the bottom to the top. Thus, in the picture, the lowermost layer is called tier 1 and the topmost tier is called tier 3.

The number of end stations (such as servers) that can be attached to such a network is determined by a very simple mathematical formula.

In a 2-tier network, if each node is made up of  $m$  ports, then the total number of end stations that can be connected is  $m^2/2$ . In more general terms, if tier-1 nodes are  $m$ -port nodes and tier-2 nodes are  $n$ -port nodes, then the total number of end stations that can be connected are  $(m*n)/2$ . In a three tier network, where tier-3 nodes are  $o$ -port nodes, the total number of end stations that can be connected are  $(m*n*o)/2^{(\text{number of tiers}-1)}$ .

Let's consider some practical examples. In many data centers, it is typical to connect 40 servers to a top-of-rack (ToR) switch. The ToRs are all connected via a set of spine switches. If a ToR switch has 64 ports, then after hooking up 40 ports to the servers, the remaining 24 ports can be hooked up to 24 spine switches of the same link speed or to a smaller number of higher link speed switches. For example, if the servers are all hooked up as 10GE links, then the ToRs can connect to the spine switches via 40G links. So, instead of connecting to 24 spine switches with 10G links, the ToRs can connect to 6 spine switches with each link being 40G. If the spine switches are also 64-port switches, then the total number of end stations that can be connected is 2560 ( $40*64$ ) stations.

In a three tier network of 64-port switches, the total number of servers that can be connected are  $(40*64*64)/2 = 81920$ . As you can see, this kind of topology can serve quite a large network with three tiers.

## Over-Subscribed and Non-Blocking Configurations

In the above example, the network is **over-subscribed**; that is, 400G of bandwidth from end stations (40 servers \* 10GE links) is serviced by only 240G of inter-rack bandwidth. The over-subscription ratio is 0.6 ( $240/400$ ).

This can lead to congestion in the network and hot spots. Instead, if network operators connected 32 servers per rack, then 32 ports are left to be connected to spine switches. Now, the network is said to be **rerrangably non-blocking**. Now any server in a rack can talk to any other server in any other rack without necessarily blocking traffic between other servers.

In such a network, the total number of servers that can be connected are  $(64*64)/2 = 2048$ . Similarly, a three-tier version of the same can serve up to  $(64*64*64)/4 = 65536$  servers.

## Containing the Failure Domain

Traditional data centers were built using just two spine switches. This means that if one of those switches fails, the network bandwidth is cut in half, thereby greatly increasing network congestion and adversely affecting many applications. To avoid this, vendors typically try and make the spine switches resilient to failures by providing such features as dual control line cards and attempting to make the software highly available. However, as Douglas Adams famously noted, ">>>". In many cases, HA is among the top two or three causes of software failure (and thereby switch failure).

To support a fairly large network with just two spine switches also means that these switches have a large port count. This can make the switches quite expensive.

If the number of spine switches were to be merely doubled, the effect of a single switch failure is halved. With 8 spine switches, the effect of a single switch failure only causes a 12% reduction in available bandwidth.

So, in modern data centers, people build networks with anywhere from 4 to 32 spine switches.

## Load Balancing

In a Clos network, traffic is load balanced across the multiple links using equal cost multi-pathing (ECMP).

Routing algorithms compute shortest paths between two end stations where shortest is typically the lowest path cost. Each link is assigned a metric or cost. By default, a link's cost is a function of the link speed. The higher the link speed, the lower its cost. A 10G link has a higher cost than a 40G or 100G link, but a lower cost than a 1G link. Thus, the link cost is a measure of its traffic carrying capacity.

In the modern data center, the links between tiers of the network are homogeneous; that is, they have the same characteristics (same speed and therefore link cost) as the other links. As a result, the first hop router can pick any of the spine switches to forward a packet to its destination (assuming that there is no link failure between the spine and the destination switch). Most routing protocols recognize that there are multiple equal-cost paths to a destination and enable any of them to be selected for a given traffic flow.

## Quagga Overview

Cumulus Linux uses **quagga**, an open source routing software suite, to provide the routing protocols for dynamic routing. Cumulus Linux supports the latest Quagga version, 0.99.23.1+cl3.0 . Quagga is a fork of the **GNU Zebra** project.

Quagga provides many routing protocols, of which Cumulus Linux supports the following:

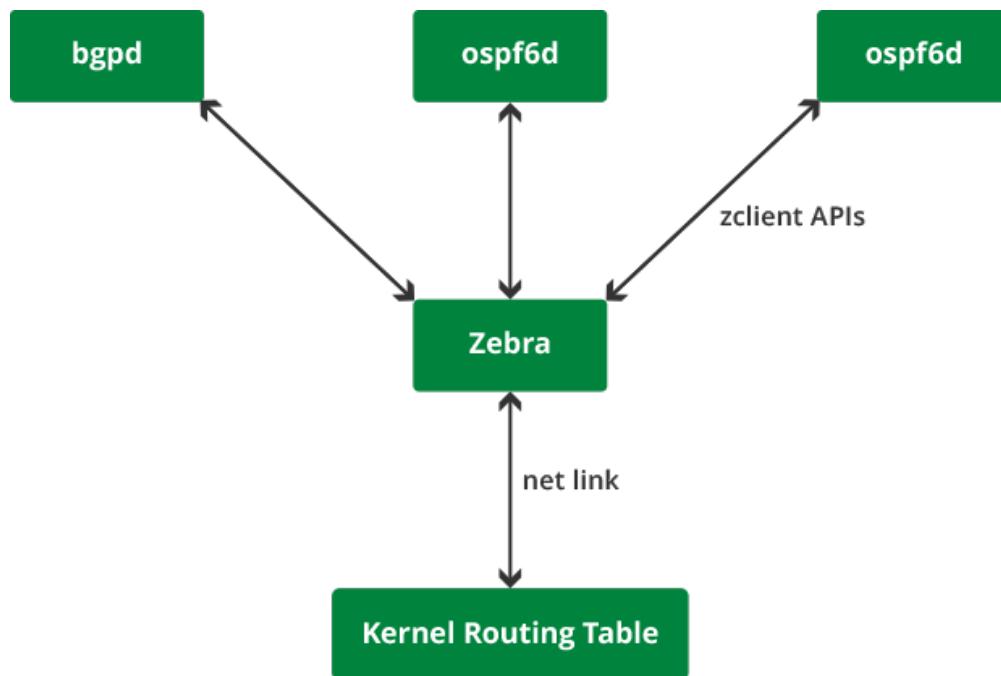
- Open Shortest Path First ([v2 \(see page 406\)](#) and [v3 \(see page 417\)](#))
- Border Gateway Protocol ([see page 419](#))

## Contents

(Click to expand)

- [Contents \(see page 391\)](#)
- [Architecture \(see page 391\)](#)
- [Zebra \(see page 392\)](#)
- [Configuration Files \(see page 392\)](#)
- [Useful Links \(see page 393\)](#)

## Architecture



As shown in the figure above, the Quagga routing suite consists of various protocol-specific daemons and a protocol-independent daemon called **zebra**. Each of the protocol-specific daemons are responsible for running the relevant protocol and building the routing table based on the information exchanged.

It is not uncommon to have more than one protocol daemon running at the same time. For example, at the edge of an enterprise, protocols internal to an enterprise (called IGP for Interior Gateway Protocol) such as [OSPF \(see page 406\)](#) or RIP run alongside the protocols that connect an enterprise to the rest of the world (called EGP or Exterior Gateway Protocol) such as [BGP \(see page 419\)](#).

**zebra** is the daemon that resolves the routes provided by multiple protocols (including static routes specified by the user) and programs these routes in the Linux kernel via **netlink** (in Linux). **zebra** does more than this, of course.

## Zebra

The [Quagga documentation](#) defines **zebra** as the IP routing manager for **quagga** that "provides kernel routing table updates, interface lookups, and redistribution of routes between different routing protocols."

## Configuration Files

- /etc/quagga/bgpd.conf
- /etc/quagga/debian.conf
- /etc/quagga/ospf6d.conf
- /etc/quagga/ospfd.conf
- /etc/quagga/vtysh.conf
- /etc/quagga/zebra.conf

## Useful Links

- <http://www.quagga.net/>
- <http://packages.debian.org/quagga>

## Configuring Quagga

This section provides an overview of configuring Cumulus Quagga, the Cumulus Networks-enhanced version of the Quagga routing software package that provides a suite of routing protocols so you can configure routing on your switch.

## Contents

This chapter covers ...

- Configuration Files (see page 393)
- Configuring Cumulus Quagga (see page 394)
  - Enabling and Starting Cumulus Quagga (see page 394)
  - Understanding Integrated Configurations (see page 395)
  - Restoring the Default Configuration (see page 396)
- Interface IP Addresses and VRFs (see page 396)
- Using the vtysh Modal CLI (see page 396)
- Using the Cumulus Linux Non-Modal CLI (see page 401)
- Reloading the Quagga Configuration (see page 401)
  - Debugging (see page 402)
- Useful Links (see page 402)

## Configuration Files

At startup, Quagga reads a set of files to determine the startup configuration. The files and what they contain are specified below:

File	Description
Quagga.conf	The default, integrated, single configuration file for all <code>quagga</code> daemons.
daemons	Contains the list of <code>quagga</code> daemons that must be started.
zebra.conf	Configuration file for the <code>zebra</code> daemon.
ospfd.conf	Configuration file for the OSPFv2 daemon.
ospf6d.conf	Configuration file for the OSPFv3 daemon.

File	Description
bgpd.conf	Configuration file for the BGP daemon.
ripd.conf	Configuration file for the RIP daemon. Cumulus Networks has not tested RIP.
ripngd.conf	Configuration file for the IPv6 RIP daemon. Cumulus Networks has not tested RIPv6.
isisd.conf	Configuration file for the IS-IS daemon. Cumulus Networks has not tested IS-IS.



The individual configuration files are not present unless you disable `integrated-vtysh-config`; see below (see page 395) for details.

## Configuring Cumulus Quagga

Cumulus Quagga does not start by default in Cumulus Linux. Before you run Cumulus Quagga, make sure all you have enabled relevant daemons that you intend to use — `zebra`, `bgpd`, `ospfd`, `ospf6d`, `ripd`, `ripngd`, `isisd` — in the `/etc/quagga/daemons` file.

The `zebra` daemon must always be enabled. The others you can enable according to how you plan to route your network — using [BGP \(see page 419\)](#) for example, instead of [OSPF \(see page 406\)](#).

Before you start Cumulus Quagga, you need to enable the corresponding daemons. Edit the `/etc/quagga/daemons` file and set to yes each daemon you are enabling.

```
zebra=yes (* this one is mandatory to bring the others up)
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
babeld=no
```

## Enabling and Starting Cumulus Quagga

Once you enable the appropriate daemons, then you need to enable and start the Cumulus Quagga service.

```
cumulus@switch:~$ sudo systemctl enable quagga.service
cumulus@switch:~$ sudo systemctl start quagga.service
```



All the routing protocol daemons (`bgpd`, `ospfd`, `ospf6d`, `ripd`, `ripngd` and `isisd`) are dependent on `zebra`. When you start `quagga`, `systemd` determines whether `zebra` is running; if `zebra` is not running, it starts `zebra`, then starts the dependent service, such as `bgpd`.

In general, if you restart a service, its dependent services also get restarted. For example, running `systemctl restart quagga.service` restarts any of the routing protocol daemons that are enabled and running.

For more information on the `systemctl` command and changing the state of daemons, read [Managing Application Daemons \(see page 122\)](#).

## ***Understanding Integrated Configurations***

By default in Cumulus Linux, Quagga saves the configuration of all daemons in a single integrated configuration file, `Quagga.conf`.

You can disable this mode by running:

```
switch(config)# no service integrated-vtysh-config
```

To enable the integrated configuration file mode again, run:

```
switch(config)# service integrated-vtysh-config
```

If you disable the integrated configuration mode, Quagga saves each daemon-specific configuration file in a separate file. At a minimum for a daemon to start, that daemon must be enabled and its daemon-specific configuration file must be present, even if that file is empty.

You can save the current configuration by running:

```
switch# write mem
Building Configuration...
Integrated configuration saved to /etc/quagga/Quagga.conf
[OK]
```



You can use `write file` instead of `write mem`.

When the integrated configuration mode disabled, the output looks like this:

```
switch# write mem
Building Configuration...
Configuration saved to /etc/quagga/zebra.conf
```

```
Configuration saved to /etc/quagga/bgpd.conf  
[OK]
```

## Restoring the Default Configuration

If you need to restore the Quagga configuration to the default running configuration, you need to delete the `Quagga.conf` file and restart the `quagga` service. You should back up `Quagga.conf` (or any configuration files you may remove, see the note below) before proceeding.

1. Confirm `service integrated-vtysh-config` is enabled:

```
cumulus@switch$ sudo cl-rctl running-config |grep integrated  
service integrated-vtysh-config
```

2. Remove `/etc/quagga/Quagga.conf`:

```
cumulus@switch$ sudo rm /etc/quagga/Quagga.conf
```

3. Restart Quagga:

```
cumulus@switch$ sudo systemctl restart quagga.service
```



If for some reason `service integrated-vtysh-config` is not configured, then you should remove all the configuration files (such as `zebra.conf` or `ospf6d.conf`) instead of `Quagga.conf` in step 2 above.

## Interface IP Addresses and VRFs

Quagga inherits the IP addresses and any associated routing tables for the network interfaces from the `/etc/network/interfaces` file. This is the recommended way to define the addresses; do **not** create interfaces using Quagga. For more information, see [Configuring IP Addresses](#) (see page 138) and [Virtual Routing and Forwarding - VRF](#) (see page 469).

## Using the vtysh Modal CLI

Quagga provides a CLI – `vtysh` – for configuring and displaying the state of the protocols. It is invoked by running:

```
cumulus@switch:~$ sudo vtysh
```

```
Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
switch#
```

`vtysh` provides a Cisco-like modal CLI, and many of the commands are similar to Cisco IOS commands. By modal CLI, we mean that there are different modes to the CLI, and certain commands are only available within a specific mode. Configuration is available with the `configure terminal` command, which is invoked thus:

```
switch# configure terminal
switch(config)#
```

The prompt displays the mode the CLI is in. For example, when the interface-specific commands are invoked, the prompt changes to:

```
switch(config)# interface swp1
switch(config-if)#
```

When the routing protocol specific commands are invoked, the prompt changes to:

```
switch(config)# router ospf
switch(config-router)#
```

At any level, "?" displays the list of available top-level commands at that level:

```
switch(config-if)# ?
babel      Babel interface commands
bandwidth   Set bandwidth informational parameter
description Interface specific description
end        End current mode and change to enable mode
exit        Exit current mode and down to previous mode
ip          Interface Internet Protocol config commands
ipv6       Interface IPv6 config commands
isis        IS-IS commands
link-detect Enable link detection on interface
list        Print command list
mpls-te    MPLS-TE specific commands
multicast   Set multicast flag to interface
no         Negate a command or set its defaults
```

```

ospf      OSPF interface commands
quit      Exit current mode and down to previous mode
shutdown  Shutdown the selected interface
  
```

?-based completion is also available to see the parameters that a command takes:

```

switch(config-if)# bandwidth ?
<1-10000000> Bandwidth in kilobits
switch(config-if)# ip ?
address  Set the IP address of an interface
irdp     Alter ICMP Router discovery preference this interface
ospf     OSPF interface commands
rip      Routing Information Protocol
router   IP router interface commands
  
```

Displaying state can be done at any level, including the top level. For example, to see the routing table as seen by **zebra**, you use:

```

switch# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.0.2, eth0
C>* 192.0.2.11/24 is directly connected, swp1
C>* 192.0.2.12/24 is directly connected, swp2
B>* 203.0.113.30/24 [200/0] via 192.0.2.2, swp1, 10:43:05
B>* 203.0.113.31/24 [200/0] via 192.0.2.2, swp1, 10:43:05
B>* 203.0.113.32/24 [200/0] via 192.0.2.2, swp1, 10:43:05
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.0/24 is directly connected, eth0
  
```

To run the same command at a config level, you prepend **do** to it:

```

switch(config-router)# do show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.0.2, eth0
C>* 192.0.2.11/24 is directly connected, swp1
  
```

```
C>* 192.0.2.12/24 is directly connected, swp2
B>* 203.0.113.30/24 [200/0] via 192.0.2.2, swp1, 10:43:05
B>* 203.0.113.31/24 [200/0] via 192.0.2.2, swp1, 10:43:05
B>* 203.0.113.32/24 [200/0] via 192.0.2.2, swp1, 10:43:05
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.0/24 is directly connected, eth0
```

Running single commands with `vtysh` is possible using the `-c` option of `vtysh`:

```
cumulus@switch:~$ sudo vtysh -c 'sh ip route'
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.0.2, eth0
C>* 192.0.2.11/24 is directly connected, swp1
C>* 192.0.2.12/24 is directly connected, swp2
B>* 203.0.113.30/24 [200/0] via 192.0.2.2, swp1, 11:05:10
B>* 203.0.113.31/24 [200/0] via 192.0.2.2, swp1, 11:05:10
B>* 203.0.113.32/24 [200/0] via 192.0.2.2, swp1, 11:05:10
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.0/24 is directly connected, eth0
```

Running a command multiple levels down is done thus:

```
cumulus@switch:~$ sudo vtysh -c 'configure terminal' -c 'router ospf' -c
'area 0.0.0.1 range 10.10.10.0/24'
```

Notice that the commands also take a partial command name (for example, `sh ip route` above) as long as the partial command name is not aliased:

```
cumulus@switch:~$ sudo vtysh -c 'sh ip r'
% Ambiguous command.
```

A command or feature can be disabled by prepending the command with `no`. For example:

```
switch(config-router)# no area 0.0.0.1 range 10.10.10.0/24
```

The current state of the configuration can be viewed using the `show running-config` command:

```
switch# show running-config
Building configuration...

Current configuration:
!
hostname quagga
log file /media/node/zebra.log
log file /media/node/bgpd.log
log timestamp precision 6
!
service integrated-vtysh-config
!
password xxxxxxxx
enable password xxxxxxxx
!
interface eth0
ipv6 nd suppress-ra
link-detect
!
interface lo
link-detect
!
interface swp1
ipv6 nd suppress-ra
link-detect
!
interface swp2
ipv6 nd suppress-ra
link-detect
!
router bgp 65000
bgp router-id 0.0.0.9
bgp log-neighbor-changes
bgp scan-time 20
network 29.0.1.0/24
timers bgp 30 90
neighbor tier-2 peer-group
neighbor 192.0.2.2 remote-as 65000
neighbor 192.0.2.2 ttl-security hops 1
neighbor 192.0.2.2 advertisement-interval 30
neighbor 192.0.2.2 timers 30 90
neighbor 192.0.2.2 timers connect 30
neighbor 192.0.2.2 next-hop-self
```

```
neighbor 192.0.2.12 remote-as 65000
neighbor 192.0.2.12 next-hop-self
neighbor 203.0.113.1 remote-as 65000
!
ip forwarding
ipv6 forwarding
!
line vty
exec-timeout 0 0
!
end
```



If you attempt to configure a routing protocol that has not been started, `vtysh` silently ignores those commands.

## Using the Cumulus Linux Non-Modal CLI

Alternately, if you do not want to use a modal CLI to configure Quagga, you can use a suite of Cumulus Linux-specific commands (see page 402) instead.

## Reloading the Quagga Configuration

If you make a change to your routing configuration, you need to reload Quagga so your changes take place. `Quagga reload` enables you to apply only the modifications you make to your Quagga configuration, synchronizing its running state with the configuration in `/etc/quagga/Quagga.conf`. This is useful for optimizing automation of Quagga in your environment or to apply changes made at runtime.



Quagga reload only applies to an integrated service configuration, where your Quagga configuration is stored in a single `Quagga.conf` file instead of one configuration file per Quagga daemon (like `zebra` or `bgpd`).

To reload your Quagga configuration after you've modified `/etc/quagga/Quagga.conf`, run:

```
cumulus@switch:~$ sudo systemctl reload quagga.service
```

Examine the running configuration and verify that it matches the config in `/etc/quagga/Quagga.conf`:

```
cumulus@switch:~$ sudo cl-rctl running-config
```

## Debugging

If the running configuration is not what you expected, please [submit a support request](#) and supply the following information:

- The current running configuration (run `sudo cl-rctl running-config` and output the contents to a file)
- The contents of `/etc/quagga/Quagga.conf`
- The contents of `/var/log/quagga/quagga-reload.log`

## Useful Links

- [www.nongnu.org/quagga/docs/docs-info.html#BGP](http://www.nongnu.org/quagga/docs/docs-info.html#BGP)
- [www.nongnu.org/quagga/docs/docs-info.html#IPv6-Support](http://www.nongnu.org/quagga/docs/docs-info.html#IPv6-Support)
- [www.nongnu.org/quagga/docs/docs-info.html#Zebra](http://www.nongnu.org/quagga/docs/docs-info.html#Zebra)

## Cumulus Linux Quagga Commands

Using the `vtysh` modal CLI is the primary way to [configure Quagga](#) (see page 393) in Cumulus Linux. However, an alternative exists in the form of a non-modal CLI containing a suite of Cumulus Linux-specific commands, structured similar to the Linux `ip` command. The available commands are as follows:

Command	Description
cl-bgp	BGP (see page 419) commands. See <code>man cl-bgp</code> for details.
cl-ospf	OSPFv2 (see page 406) commands. For example: <code>cumulus@switch:~\$ sudo cl-ospf area 0.0.0.1 range 10.10.10.0/24</code>
cl-ospf6	OSPFv3 (see page 417) commands.
cl-ra	Route advertisement commands. See <code>man cl-ra</code> for details.
cl-rctl	Zebra and non-routing protocol-specific commands. See <code>man cl-rctl</code> for details.

## Comparing vtysh and Cumulus Linux Commands

This section describes how you can use the various Cumulus Linux CLI commands to configure Quagga, without using `vtysh`.

## Displaying the Routing Table

To display the routing table under Quagga, you would run:

```
switch# show ip route
```

To display the routing table with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-rctl route
```

## ***Creating a New Neighbor***

To create a new neighbor under Quagga, you would run:

```
switch(config)# router bgp 65002
switch(config-router)# neighbor 14.0.0.22 remote-as 65007
```

To create a new neighbor with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-bgp as 65002 neighbor add 14.0.0.22 remote-as
65007
```

## ***Redistributing Routing Information***

To redistribute routing information from static route entries into RIP tables under Quagga, you would run:

```
switch(config)# router bgp 65002
switch(config-router)# redistribute static
```

To redistribute routing information from static route entries into RIP tables with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-bgp as 65002 redistribute add static
```

## ***Defining a Static Route***

If you intend to use static routes, you only need to enable the `zebra` daemon.

To define a static route under Quagga, you would run:

```
switch(config)# ip route 155.1.2.20/24 br2 45
```

To define a static route with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-rctl ip route add 175.0.0.0/28 interface br1  
distance 25
```

## Configuring an IPv6 Interface

To configure an IPv6 address under Quagga, you would run:

```
switch(config)# int br3  
switch(config-if)# ipv6 address 3002:2123:1234:1abc::21/64
```

To configure an IPv6 address with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-rctl interface add swp3 ipv6 address 3002:2123:  
abcd:2120::41/64
```

## Enabling PTM

To enable topology checking (PTM) under Quagga, you would run:

```
switch(config)# ptm-enable
```

To enable topology checking (PTM) with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-rctl ptm-enable set
```

## Configuring MTU in IPv6 Network Discovery

To configure [MTU](#) (see page 152) in IPv6 network discovery for an interface under Quagga, you would run:

```
switch(config)# int swp3  
switch(config-if)# ipv6 nd mtu 9000
```

To configure MTU in IPv6 network discovery for an interface with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ra interface swp3 set mtu 9000
```

## ***Logging OSPF Adjacency Changes***

To log adjacency of OSPF changes under Quagga, you would run:

```
switch(config)# router ospf
switch(config-router)# router-id 2.0.0.21
switch(config-router)# log-adjacency-changes
```

To log adjacency changes of OSPF with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ospf log-adjacency-changes set
cumulus@switch:~$ sudo cl-ospf router-id set 3.0.0.21
```

## ***Setting OSPF Interface Priority***

To set the OSPF interface priority under Quagga, you would run:

```
switch(config)# int swp3
switch(config-if)# ip ospf priority 120
```

To set the OSPF interface priority with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ospf interface set swp3 priority 120
```

## ***Configuring Timing for OSPF SPF Calculations***

To configure timing for OSPF SPF calculations under Quagga, you would run:

```
switch(config)# router ospf6
switch(config-ospf6)# timer throttle spf 40 50 60
```

To configure timing for OSPF SPF calculations with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ospf6 timer add throttle spf 40 50 60
```

## ***Configuring Hello Packet Intervals***

To configure the OSPF Hello packet interval in number of seconds for an interface under Quagga, you would run:

```
switch(config)# int swp4
switch(config-if)# ipv6 ospf6 hello-interval 60
```

To configure the OSPF Hello packet interval in number of seconds for an interface with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ospf6 interface set swp4 hello-interval 60
```

## Displaying OSPF Debugging Status

To display OSPF debugging status under Quagga, you would run:

```
switch# show debugging ospf
```

To display OSPF debugging status with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-ospf debug show
```

## Displaying BGP Information

To display BGP information under Quagga, you would run:

```
switch# show ip bgp summary
```

To display BGP information with the Cumulus Linux CLI, run:

```
cumulus@switch:~$ sudo cl-bgp summary
```

# Open Shortest Path First - OSPF - Protocol

OSPFv2 is a [link-state routing protocol](#) for IPv4. OSPF maintains the view of the network topology conceptually as a directed graph. Each router represents a vertex in the graph. Each link between neighboring routers represents a unidirectional edge. Each link has an associated weight (called cost) that is either automatically derived from its bandwidth or administratively assigned. Using the weighted topology graph, each router computes a shortest path tree (SPT) with itself as the root, and applies the results to build its forwarding table. The computation is generally referred to as *SPF computation* and the resultant tree as the *SPF tree*.

An LSA (*link-state advertisement*) is the fundamental quantum of information that OSPF routers exchange with each other. It seeds the graph building process on the node and triggers SPF computation. LSAs originated by a node are distributed to all the other nodes in the network through a mechanism called *flooding*. Flooding is done hop-by-hop. OSPF ensures reliability by using link state acknowledgement packets. The set of LSAs in a router's memory is termed *link-state database* (LSDB), a representation of the network graph. Thus, OSPF ensures a consistent view of LSDB on each node in the network in a distributed fashion (eventual consistency model); this is key to the protocol's correctness.

## Contents

This chapter covers ...

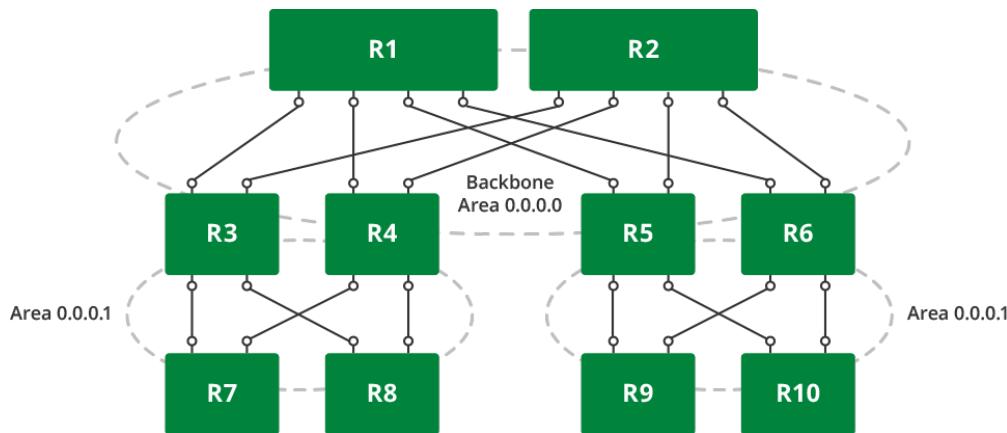
- Scalability and Areas (see page 407)
- Configuring OSPFv2 (see page 408)
  - Enabling the OSPF and Zebra Daemons (see page 408)
  - Configuring OSPF (see page 408)
  - Defining (Custom) OSPF Parameters on the Interfaces (see page 410)
- Scaling Tip: Summarization (see page 411)
- Scaling Tip: Stub Areas (see page 412)
- Configuration Tip: Unnumbered Interfaces (see page 413)
- ECMP (see page 414)
- Topology Changes and OSPF Reconvergence (see page 414)
  - Example Configurations (see page 414)
- Debugging OSPF (see page 415)
- Configuration Files (see page 416)
- Supported RFCs (see page 416)
- Useful Links (see page 416)

## Scalability and Areas

An increase in the number of nodes affects OSPF scalability in the following ways:

- Memory footprint to hold the entire network topology,
- Flooding performance,
- SPF computation efficiency.

The OSPF protocol advocates hierarchy as a *divide and conquer* approach to achieve high scale. The topology may be divided into areas, resulting in a two-level hierarchy. Area 0 (or 0.0.0.0), called the backbone area, is the top level of the hierarchy. Packets traveling from one non-zero area to another must go via the backbone area. As an example, the leaf-spine topology we have been referring to in the routing section can be divided into areas as follows:



Here are some points to note about areas and OSPF behavior:

- Routers that have links to multiple areas are called *area border routers* (ABR). For example, routers R3, R4, R5, R6 are ABRs in the diagram. An ABR performs a set of specialized tasks, such as SPF computation per area and summarization of routes across areas.
- Most of the LSAs have an area-level flooding scope. These include router LSA, network LSA, and summary LSA.



In the diagram, we reused the same non-zero area address. This is fine since the area address is only a scoping parameter provided to all routers within that area. It has no meaning outside the area. Thus, in the cases where ABRs do not connect to multiple non-zero areas, the same area address can be used, thus reducing the operational headache of coming up with area addresses.

## Configuring OSPFv2

Configuring OSPF involves the following tasks:

- Enabling the OSPF daemon
- Enabling OSPF
- Defining (Custom) OSPF parameters on the interfaces

### Enabling the OSPF and Zebra Daemons

To enable OSPF, enable the `zebra` and `ospf` daemons, as described in [Configuring Quagga](#) (see page 393), then start the Quagga service:

```
cumulus@switch:~$ sudo systemctl start quagga.service
```

## Configuring OSPF

As we discussed in [Introduction to Routing Protocols](#) (see page 387), there are three steps to the configuration:

1. Identifying the router with the router ID.

2. With whom should the router communicate?
3. What information (most notably the prefix reachability) to advertise?

There are two ways to achieve (2) and (3) in the Quagga OSPF:

1. The **network** statement under **router ospf** does both. The statement is specified with an IP subnet prefix and an area address. All the interfaces on the router whose IP address matches the **network** subnet are put into the specified area. OSPF process starts bringing up peering adjacency on those interfaces. It also advertises the interface IP addresses formatted into LSAs (of various types) to the neighbors for proper reachability.

From the Cumulus Linux shell:

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R3# configure terminal
R3(config)# router ospf
R3(config-router)# router-id 0.0.0.1
R3(config-router)# log-adjacency-changes detail
R3(config-router)# network 10.0.0.0/16 area 0.0.0.0
R3(config-router)# network 192.0.2.0/16 area 0.0.0.1
R3(config-router)#

```

Or through **cl-ospf**, from the Cumulus Linux shell:

```
cumulus@switch:~$ sudo cl-ospf router set id 0.0.0.1
cumulus@switch:~$ sudo cl-ospf router set log-adjacency-changes detail
cumulus@switch:~$ sudo cl-ospf router set network 10.0.0.0/16 area
0.0.0.0
cumulus@switch:~$ sudo cl-ospf router set network 192.0.2.0/16 area
0.0.0.1

```

The subnets in the **network** subnet can be as coarse as possible to cover the most number of interfaces on the router that should run OSPF.

There may be interfaces where it's undesirable to bring up OSPF adjacency. For example, in a data center topology, the host-facing interfaces need not run OSPF; however the corresponding IP addresses should still be advertised to neighbors. This can be achieved using the **passive-interface** construct.

From the vtysh/quagga CLI:

```
R3# configure terminal  
R3(config)# router ospf  
R3(config-router)# passive-interface swp10  
R3(config-router)# passive-interface swp11
```

Or use the `passive-interface default` command to put all interfaces as passive and selectively remove certain interfaces to bring up protocol adjacency:

```
R3# configure terminal  
R3(config)# router ospf  
R3(config-router)# passive-interface default  
R3(config-router)# no passive-interface swp1
```

2. Explicitly enable OSPF for each interface by configuring it under the interface configuration mode:

```
R3# configure terminal  
R3(config)# interface swp1  
R3(config-if)# ip ospf area 0.0.0.0
```

If OSPF adjacency bringup is not desired, you should configure the corresponding interfaces as passive as explained above.

This model of configuration is required for unnumbered interfaces as discussed later in this guide.

For achieving step (3) alone, the `quagga` configuration provides another method: *redistribution*. For example:

```
R3# configure terminal  
R3(config)# router ospf  
R3(config-router)# redistribute connected
```

Redistribution, however, unnecessarily loads the database with type-5 LSAs and should be limited to generating real external prefixes (for example, prefixes learned from BGP). In general, it is a good practice to generate local prefixes using `network` and/or `passive-interface` statements.

## **Defining (Custom) OSPF Parameters on the Interfaces**

1. Network type, such as point-to-point, broadcast.
2. Timer tuning, like hello interval.
3. For unnumbered interfaces (see below), enable OSPF.

Using Quagga's vtysh:

```
R3(config)# interface swp1
R3(config-if)# ospf network point-to-point
R3(config-if)# ospf hello-interval 5
```

Or through cl-ospf, from the Cumulus Linux shell:

```
cumulus@switch:~$ sudo cl-ospf interface swp1 set network point-to-point
cumulus@switch:~$ sudo cl-ospf interface swp1 set hello-interval 5
```

The OSPF configuration is saved in `/etc/quagga/ospfd.conf`.

### ***Scaling Tip: Summarization***

By default, an ABR creates a summary (type-3) LSA for each route in an area and advertises it in adjacent areas. Prefix range configuration optimizes this behavior by creating and advertising one summary LSA for multiple routes.

To configure a range:

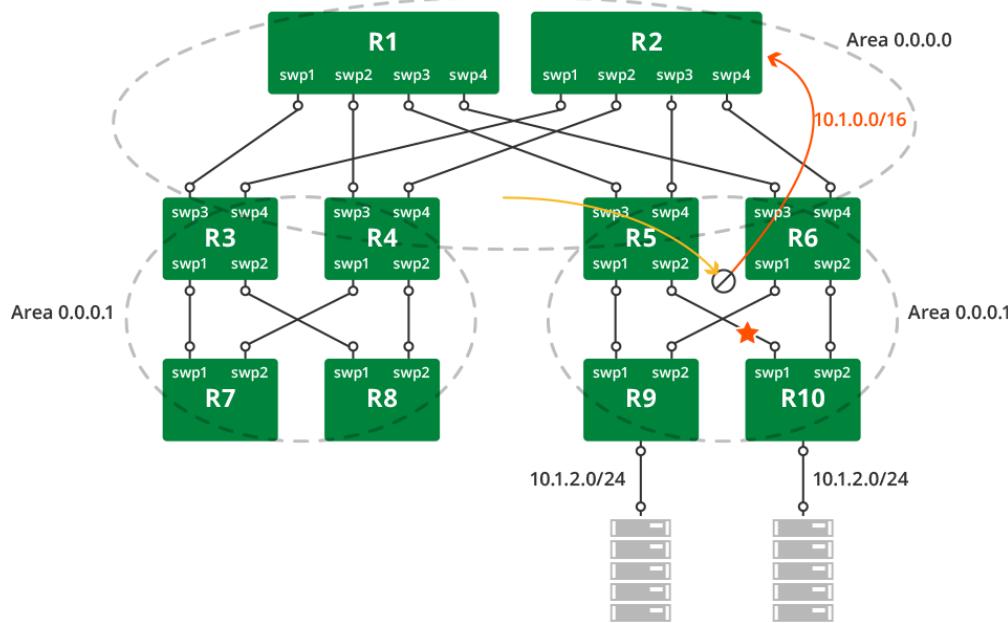
```
R3(config)# router ospf
R3(config-router)# area 0.0.0.1 range 30.0.0.0/8
```



Summarize in the direction to the backbone. The backbone receives summarized routes and injects them to other areas already summarized.



Summarization can cause non-optimal forwarding of packets during failures. Here is an example scenario:



As shown in the diagram, the ABRs in the right non-zero area summarize the host prefixes as 10.1.0.0/16. When the link between R5 and R10 fails, R5 will send a worse metric for the summary route (metric for the summary route is the maximum of the metrics of intra-area routes that are covered by the summary route. Upon failure of the R5-R10 link, the metric for 10.1.2.0/24 goes higher at R5 as the path is R5-R9-R6-R10). As a result, other backbone routers shift traffic destined to 10.1.0.0/16 towards R6. This breaks ECMP and is an under-utilization of network capacity for traffic destined to 10.1.1.0/24.

## Scaling Tip: Stub Areas

Nodes in an area receive and store intra-area routing information and summarized information about other areas from the ABRs. In particular, a good summarization practice about inter-area routes through prefix range configuration helps scale the routers and keeps the network stable.

Then there are external routes. External routes are the routes redistributed into OSPF from another protocol. They have an AS-wide flooding scope. In many cases, external link states make up a large percentage of the LSDB.

*Stub areas* alleviate this scaling problem. A stub area is an area that does not receive external route advertisements.

To configure a stub area:

```
R3(config)# router ospf
R3(config-router)# area 0.0.0.1 stub
```

Stub areas still receive information about networks that belong to other areas of the same OSPF domain. Especially, if summarization is not configured (or is not comprehensive), the information can be overwhelming for the nodes. *Totally stubby areas* address this issue. Routers in totally stubby areas keep in their LSDB information about routing within their area, plus the default route.

To configure a totally stubby area:

```
R3(config)# router ospf
R3(config-router)# area 0.0.0.1 stub no-summary
```

Here is a brief tabular summary of the area type differences:

Type	Behavior
Normal non- zero area	LSA types 1, 2, 3, 4 area-scoped, type 5 externals, inter-area routes summarized
Stub area	LSA types 1, 2, 3, 4 area-scoped, No type 5 externals, inter-area routes summarized
Totally stubby area	LSA types 1, 2 area-scoped, default summary, No type 3, 4, 5 LSA types allowed

### Configuration Tip: Unnumbered Interfaces

Unnumbered interfaces are interfaces without unique IP addresses. In OSPFv2, configuring unnumbered interfaces reduces the links between routers into pure topological elements, which dramatically simplifies network configuration and reconfiguration. In addition, the routing database contains only the real networks, so the memory footprint is reduced and SPF is faster.



Unnumbered is usable for point-to-point interfaces only.



If there is a `network <network number>/<mask> area <area ID>` command present in the Quagga configuration, the `ip ospf area <area ID>` command is rejected with the error "Please remove network command first." This prevents you from configuring other areas on some of the unnumbered interfaces. You can use either the `network area` command or the `ospf area` command in the configuration, but not both.



Unless the Ethernet media is intended to be used as a LAN with multiple connected routers, we recommend configuring the interface as point-to-point. It has the additional advantage of a simplified adjacency state machine; there is no need for DR/BDR election and *LSA reflection*. See [RFC5309](#) for a more detailed discussion.

To configure an unnumbered interface, take the IP address of another interface (called the *anchor*) and use that as the IP address of the unnumbered interface:

```
auto lo
iface lo inet loopback
    address 192.0.2.1/32
```

```
auto swp1
iface swp1
  address 192.0.2.1/32

auto swp2
iface swp2
  address 192.0.2.1/32
```

To enable OSPF on an unnumbered interface from within Quagga's **vtysh**:

```
R3(config)# interface swp1
R3(config-if)# ip ospf area 0.0.0.1
```

## ECMP

During SPF computation for an area, if OSPF finds multiple paths with equal cost (metric), all those paths are used for forwarding. For example, in the reference topology diagram, R8 uses both R3 and R4 as next hops to reach a destination attached to R9.

## Topology Changes and OSPF Reconvergence

Topology changes usually occur due to one of four events:

1. Maintenance of a router node
2. Maintenance of a link
3. Failure of a router node
4. Failure of a link

For the maintenance events, operators typically raise the OSPF administrative weight of the link(s) to ensure that all traffic is diverted from the link or the node (referred to as *costing out*). The speed of reconvergence does not matter. Indeed, changing the OSPF cost causes LSAs to be reissued, but the links remain in service during the SPF computation process of all routers in the network.

For the failure events, traffic may be lost during reconvergence; that is, until SPF on all nodes computes an alternative path around the failed link or node to each of the destinations. The reconvergence depends on layer 1 failure detection capabilities and at the worst case *DeadInterval* OSPF timer.

## Example Configurations

Example configuration for event 1, using **vtysh**:

```
R3(config)# router ospf
R3(config-router)# max-metric router-lsa administrative
```

Or, with the non-modal shell command approach:

```
cumulus@switch:~$ sudo cl-ospf router set max-metric router-lsa
administrative
```

Example configuration for event 2, using vtysh:

```
R3(config)# interface swp1
R3(config-if)# ospf cost 65535
```

Or, with the non-modal shell command approach:

```
cumulus@switch:~$ sudo cl-ospf interface swp1 set cost 65535
```

## Debugging OSPF

[OperState](#) lists all the commands to view the operational state of OSPF.

The three most important states while troubleshooting the protocol are:

1. Neighbors, with `show ip ospf neighbor`. This is the starting point to debug neighbor states (also see `tcpdump` below).
2. Database, with `show ip ospf database`. This is the starting point to verify that the LSDB is, in fact, synchronized across all routers in the network. For example, sweeping through the output of `show ip ospf database router` taken from all routers in an area will ensure if the topology graph building process is complete; that is, every node has seen all the other nodes in the area.
3. Routes, with `show ip ospf route`. This is the outcome of SPF computation that gets downloaded to the forwarding table, and is the starting point to debug, for example, why an OSPF route is not being forwarded correctly.



Compare the route output with kernel by using `show ip route | grep zebra` and with the hardware entries using `cl-route-check -V`.

Using `cl-ospf`:

```
cumulus@switch:~$ sudo cl-ospf neighbor show [all | detail]

cumulus@switch:~$ sudo cl-ospf database show [asbr-summary | network |
opaque-area |
                    opaque-link | summary | external |
                    nssa-external | opaque-as | router]
```

```
cumulus@switch:~$ sudo cl-ospf route show
```

Debugging-OSPF lists all of the OSPF debug options.

Using cl-ospf:

```
Usage: cl-ospf debug { COMMAND | help }

COMMANDS
  { set | clear } (all | event | ism | ism [OBJECT] | lsa | lsa
  [OBJECT] |
    nsm | nsm [OBJECT] | nssa | packet | packet [OBJECT] |
    zebra [OBJECT] | zebra all)
```

Using zebra under vtysh:

```
cumulus@switch:~$ sudo vtysh
R3# show [zebra]

IOBJECT := { events | status | timers }
OOBJECT := { interface | redistribute }
POBJECT := { all | dd | hello | ls-ack | ls-request | ls-update }
ZOBJECT := { all | events | kernel | packet | rib |
```

Using tcpdump to capture OSPF packets:

```
cumulus@switch:~$ sudo tcpdump -v -i swp1 ip proto ospf
```

## Configuration Files

- /etc/quagga/ospfd.conf

## Supported RFCs

- RFC2328
- RFC3137
- RFC5309

## Useful Links

- Bidirectional forwarding detection (see page 446) (BFD) and OSPF

- [http://en.wikipedia.org/wiki/Open\\_Shortest\\_Path\\_First](http://en.wikipedia.org/wiki/Open_Shortest_Path_First)
- <http://www.nongnu.org/quagga/docs/docs-info.html#OSPFv2>
- Perlman, Radia (1999). Interconnections: Bridges, Routers, Switches, and Internetworking Protocols (2 ed.). Addison-Wesley.
- Moy, John T. OSPF: Anatomy of an Internet Routing Protocol. Addison-Wesley.

## Open Shortest Path First v3 - OSPFv3 - Protocol

OSPFv3 is a revised version of OSPFv2 to support the IPv6 address family. Refer to [Open Shortest Path First \(OSPF\) Protocol \(see page 406\)](#) for a discussion on the basic concepts, which remain the same between the two versions.

OSPFv3 has changed the formatting in some of the packets and LSAs either as a necessity to support IPv6 or to improve the protocol behavior based on OSPFv2 experience. Most notably, v3 defines a new LSA, called intra-area prefix LSA to separate out the advertisement of stub networks attached to a router from the router LSA. It is a clear separation of node topology from prefix reachability and lends itself well to an optimized SPF computation.



IETF has defined extensions to OSPFv3 to support multiple address families (that is, both IPv6 and IPv4). Quagga (see page 391) does not support it yet.

## Contents

This chapter covers ...

- Configuring OSPFv3 (see page 417)
- Unnumbered Interfaces (see page 419)
- Debugging OSPF (see page 419)
- Configuration Files (see page 419)
- Supported RFCs (see page 419)
- Useful Links (see page 419)

## Configuring OSPFv3

Configuring OSPFv3 involves the following tasks:

1. Enabling the `zebra` and `ospf6` daemons, as described in [Configuring Quagga \(see page 393\)](#), then start the Quagga service:

```
cumulus@switch:~$ sudo systemctl start quagga.service
```

2. Enabling OSPF6 and map interfaces to areas. From Quagga's `vtysh` shell:

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

switch# conf t
switch# configure terminal
switch(config)# router ospf6
switch(config-router)# router-id 0.0.1
switch(config-router)# log-adjacency-changes detail
switch(config-router)# interface swp1 area 0.0.0.0
switch(config-router)# interface swp2 area 0.0.0.1
switch(config-router)#

```

Or through **cl-ospf6**, from the Cumulus Linux shell:

```
cumulus@switch:~$ sudo cl-ospf6 router set id 0.0.0.1
cumulus@switch:~$ sudo cl-ospf6 router set log-adjacency-changes detail
cumulus@switch:~$ sudo cl-ospf6 interface swp1 set area 0.0.0.0
cumulus@switch:~$ sudo cl-ospf6 interface swp2 set area 0.0.0.1

```

3. Defining (custom) OSPF6 parameters on the interfaces:
  - a. Network type (such as point-to-point, broadcast)
  - b. Timer tuning (for example, hello interval)

Using Quagga's **vtysh**:

```
switch(config)# interface swp1
switch(config-if)# ipv6 ospf6 network point-to-point
switch(config-if)# ipv6 ospf6 hello-interval 5

```

Or through **cl-ospf6**, from the Cumulus Linux shell:

```
cumulus@switch:~$ sudo cl-ospf6 interface swp1 set network point-to-
point
cumulus@switch:~$ sudo cl-ospf6 interface swp1 set hello-interval 5

```

The OSPFv3 configuration is saved in `/etc/quagga/ospf6d.conf`.

## Unnumbered Interfaces

Unlike OSPFv2, OSPFv3 intrinsically supports unnumbered interfaces. Forwarding to the next hop router is done entirely using IPv6 link local addresses. Therefore, you are not required to configure any global IPv6 address to interfaces between routers.

## Debugging OSPF

See [Debugging OSPF \(see page 415\)](#) for OSPFv2 for the troubleshooting discussion. The equivalent commands are:

```
cumulus@switch:~$ sudo vtysh
switch# show ipv6 ospf6 neighbor
switch# show ipv6 ospf6 database [detail | dump | internal |
                                as-external | group-membership |
                                inter-prefix | inter-router |
                                intra-prefix | link | network |
                                router | type-7 | * | adv-router |
                                linkstate-id | self-originated]
switch# show ip ospf route
```

Another helpful command is `show ipv6 ospf6 [area <id>] spf tree`. It dumps the node topology as computed by SPF to help visualize the network view.

## Configuration Files

- /etc/quagga/ospf6d.conf

## Supported RFCs

- RFC5340
- RFC3137

## Useful Links

- Bidirectional forwarding detection (see page 446) (BFD) and OSPF
- [http://en.wikipedia.org/wiki/Open\\_Shortest\\_Path\\_First](http://en.wikipedia.org/wiki/Open_Shortest_Path_First)
- <http://www.nongnu.org/quagga/docs/docs-info.html#OSPFv3>

## Border Gateway Protocol - BGP

BGP is the routing protocol that runs the Internet. It is an increasingly popular protocol for use in the data center as it lends itself well to the rich interconnections in a Clos topology. Specifically:

- It does not require routing state to be periodically refreshed unlike OSPF.
- It is less chatty than its link-state siblings. For example, a link or node transition can result in a bestpath change, causing BGP to send updates.

- It is multi-protocol and extensible.
- There are many robust vendor implementations.
- The protocol is very mature and comes with many years of operational experience.

This IETF draft provides further details of the use of BGP within the data center.

## Contents

This chapter covers ...

- Commands (see page 421)
- Autonomous System Number (ASN) (see page 421)
- eBGP and iBGP (see page 421)
- Route Reflectors (see page 422)
- ECMP with BGP (see page 422)
  - Maximum Paths (see page 422)
  - Multipath Relax (see page 423)
- BGP for both IPv4 and IPv6 (see page 423)
- Configuring BGP (see page 423)
- Using BGP Unnumbered Interfaces (see page 425)
  - BGP and Extended Next-hop Encoding (see page 426)
  - Configuring BGP Unnumbered Interfaces (see page 426)
  - Managing Unnumbered Interfaces (see page 426)
  - How traceroute Interacts with BGP Unnumbered Interfaces (see page 428)
  - Advanced: Understanding How Next-hop Fields Are Set (see page 428)
  - Limitations (see page 429)
- BGP add-path (see page 430)
  - BGP add-path RX (see page 430)
  - BGP add-path TX (see page 432)
- Fast Convergence Design Considerations (see page 434)
  - Specifying the Interface Name in the neighbor Command (see page 434)
- Configuring BGP Peering Relationships across Switches (see page 435)
- Configuration Tips (see page 436)
  - Using peer-group to Simplify Configuration (see page 436)
  - Preserving the AS\_PATH Setting (see page 437)
  - Utilizing Multiple Routing Tables and Forwarding (see page 437)
- Troubleshooting (see page 437)
  - Debugging Tip: Logging Neighbor State Changes (see page 440)
  - Troubleshooting Link-local Addresses (see page 440)
- Enabling Read-only Mode (see page 441)
- Applying a Route Map for Route Updates (see page 442)

- Protocol Tuning (see page 442)
  - Converging Quickly On Link Failures (see page 442)
  - Converging Quickly On Soft Failures (see page 443)
  - Reconnecting Quickly (see page 444)
  - Advertisement Interval (see page 444)
- Configuration Files (see page 445)
- Useful Links (see page 445)
- Caveats and Errata (see page 445)
  - ttl-security Issue (see page 445)

## Commands

Cumulus Linux:

- bgp
- vtysh

Quagga:

- bgp
- neighbor
- router
- show

## Autonomous System Number (ASN)

One of the key concepts in BGP is an *autonomous system number* or ASN. An **autonomous system** is defined as a set of routers under a common administration. Since BGP was originally designed to peer between independently managed enterprises and/or service providers, each such enterprise is treated as an autonomous system, responsible for a set of network addresses. Each such autonomous system is given a unique number called its ASN. ASNs are handed out by a central authority (ICANN). However, ASNs between 64512 and 65535 are reserved for private use. Using BGP within the data center relies on either using this number space or else using the single ASN you own.

The ASN is central to how BGP builds a forwarding topology. A BGP route advertisement carries with it not only the originator's ASN, but also the list of ASNs that this route advertisement has passed through. When forwarding a route advertisement, a BGP speaker adds itself to this list. This list of ASNs is called the *AS path*. BGP uses the AS path to detect and avoid loops.

ASNs were originally 16-bit numbers, but were later modified to be 32-bit. Quagga supports both 16-bit and 32-bit ASNs, but most implementations still run with 16-bit ASNs.

## eBGP and iBGP

When BGP is used to peer between autonomous systems, the peering is referred to as *external BGP* or eBGP. When BGP is used within an autonomous system, the peering used is referred to as *internal BGP* or iBGP. eBGP peers have different ASNs while iBGP peers have the same ASN.

While the heart of the protocol is the same when used as eBGP or iBGP, there is a key difference in the protocol behavior between use as eBGP and iBGP: an iBGP node does not forward routing information learned from one iBGP peer to another iBGP peer. It expects the originating iBGP peer to send this information to all iBGP peers.

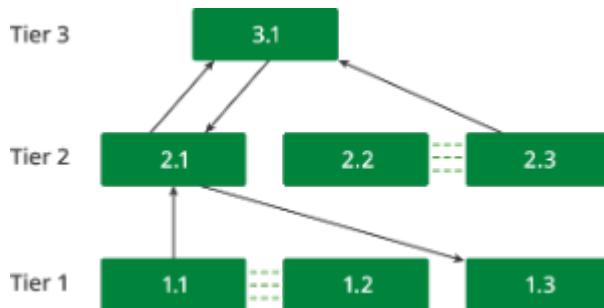
This implies that iBGP peers are all connected to each other. In a large network, this requirement can quickly become unscalable. The most popular method to avoid this problem is to introduce a *route reflector*.

## Route Reflectors

Route reflectors are quite easy to understand in a Clos topology. In a two-tier Clos network, the leaf (or tier 1) switches are the only ones connected to end stations. Subsequently, this means that the spines themselves do not have any routes to announce. They're merely **reflecting** the routes announced by one leaf to the other leaves. Thus, the spine switches function as route reflectors while the leaf switches serve as route reflector clients.

In a three-tier network, the tier 2 nodes (or mid-tier spines) act as both route reflector servers and route reflector clients. They act as route reflectors because they announce the routes learned from the tier 1 nodes to other tier 1 nodes and to tier 3 nodes. They also act as route reflector clients to the tier 3 nodes, receiving routes learned from other tier 2 nodes. Tier 3 nodes act only as route reflectors.

In the following illustration, tier 2 node 2.1 is acting as a route reflector server, announcing the routes between tier 1 nodes 1.1 and 1.2 to tier 1 node 1.3. It is also a route reflector client, learning the routes between tier 2 nodes 2.2 and 2.3 from the tier 3 node, 3.1.



## ECMP with BGP

If a BGP node hears a prefix **p** from multiple peers, it has all the information necessary to program the routing table to forward traffic for that prefix **p** through all of these peers. Thus, BGP supports equal-cost multipathing (ECMP).

In order to perform ECMP in BGP, you may need to configure two parameters: *maximum paths* (if you're using iBGP) and, *multipath relax* (if you're using eBGP).

## Maximum Paths

In Cumulus Linux, the BGP `maximum-paths` setting is enabled by default, so multiple routes are already installed.

However, if you're using iBGP, use the `maximum-paths ibgp` command as shown below:

```

leaf1# conf t
leaf1(config)# router bgp 65000
leaf1(config-router)# maximum-paths
<1-255> Number of paths
ibgp      iBGP-multipath
leaf1(config-router)# maximum-paths ibgp
<1-255> Number of paths
  
```

## Multipath Relax

If your data center uses eBGP, you can to configure a second parameter for proper ECMP: the `bestpath as-path multipath-relax no-as-set` command. You configure it under the BGP routing process.

```
leaf1# conf t
leaf1(config)# router bgp 65000
leaf1(config-router)# bgp bestpath
as-path          compare-routerid  med
leaf1(config-router)# bgp bestpath as-path
confed          ignore           multipath-relax
leaf1(config-router)# bgp bestpath as-path multipath-relax
<cr>
no-as-set  Do not generate an AS_SET
leaf1(config-router)# bgp bestpath as-path multipath-relax no-as-set
```

To see more information on the `no-as-set` option, read [the AS\\_PATH section below \(see page 437\)](#).

## BGP for both IPv4 and IPv6

Unlike OSPF, which has separate versions of the protocol to announce IPv4 and IPv6 routes, BGP is a multi-protocol routing engine, capable of announcing both IPv4 and IPv6 prefixes. It supports announcing IPv4 prefixes over an IPv4 session and IPv6 prefixes over an IPv6 session. It also supports announcing prefixes of both these address families over a single IPv4 session or over a single IPv6 session.

## Configuring BGP

1. Enable the BGP and Zebra daemons:

- Enable the `zebra` and `bgpd` daemons as described in [Configuring Quagga \(see page 393\)](#), then start Quagga:

```
cumulus@switch:~$ sudo systemctl restart quagga.service
```

- Touch an empty `bgpd` configuration file:

```
cumulus@switch:~$ sudo touch /etc/quagga/bgpd.conf
```

A slightly more useful configuration file would contain the following lines:

```
hostname R7
password *****
enable password *****
log timestamp precision 6
```

```

log file /var/log/quagga/bgpd.log
!
line vty
  exec-timeout 0 0
!

```

The most important information here is the specification of the location of the log file, where the BGP process can log debugging and other useful information. A common convention is to store the log files under `/var/log/quagga`.

You must restart the `bgpd` daemon when you update the BGP configuration:

```

cumulus@switch:~$ sudo /usr/lib/quagga/quagga restart bgpd.
service

```

2. Identify the BGP node by assigning an ASN and `router-id`:

```

cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R7# configure terminal
R7(config)# router bgp 65000
R7(config-router)# bgp router-id 0.0.0.1

```

3. Specify to whom it must disseminate routing information:

```
R7(config-router)# neighbor 10.0.0.2 remote-as 65001
```

If it is an iBGP session, the `remote-as` is the same as the local AS:

```
R7(config-router)# neighbor 10.0.0.2 remote-as 65000
```

Specifying the peer's IP address allows BGP to set up a TCP socket with this peer, but it doesn't distribute any prefixes to it, unless it is explicitly told that it must via the `activate` command:

```
R7(config-router)# address-family ipv4 unicast
R7(config-router-af)# neighbor 10.0.0.2 activate
```

```
R7(config-router-af)# exit
R7(config-router)# address-family ipv6
R7(config-router-af)# neighbor 2001:db8:0002::0a00:0002 activate
R7(config-router-af)# exit
```

As you can see, `activate` has to be specified for each address family that is being announced by the BGP session.

- Specify some properties of the BGP session:

```
R7(config-router)# neighbor 10.0.0.2 next-hop-self
R7(config-router)# address-family ipv4 unicast
R7(config-router-af)# maximum-paths 64
```

For iBGP, the `maximum-paths` is selected by typing:

```
R7(config-router-af)# maximum-paths ibgp 64
```

If this is a route-reflector client, it can be specified as follows:

```
R3(config-router-af)# neighbor 10.0.0.1 route-reflector-client
```



It is node R3, the route reflector, on which the peer is specified as a client.

- Specify what prefixes to originate:

```
R7(config-router)# address-family ipv4 unicast
R7(config-router-af)# network 192.0.2.0/24
R7(config-router-af)# network 203.0.113.1/24
```

## **Using BGP Unnumbered Interfaces**

Unnumbered interfaces are interfaces without unique IP addresses. In BGP, you configure unnumbered interfaces using *extended next-hop encoding* (ENHE), which is defined by [RFC 5549](#). BGP unnumbered interfaces provide a means of advertising an IPv4 route with an IPv6 next-hop. Prior to RFC 5549, an IPv4 route could be advertised only with an IPv4 next-hop.

BGP unnumbered interfaces are particularly useful in deployments where IPv4 prefixes are advertised through BGP over a section without any IPv4 address configuration on links. As a result, the routing entries are also IPv4 for destination lookup and have IPv6 next-hops for forwarding purposes.

## BGP and Extended Next-hop Encoding

Once enabled and active, BGP makes use of the available IPv6 next-hops for advertising any IPv4 prefixes. BGP learns the prefixes, calculates the routes and installs them in IPv4 AFI to IPv6 AFI format. However, ENHE in Cumulus Linux does not install routes into the kernel in IPv4 prefix to IPv6 next-hop format. For link-local peerings enabled by dynamically learning the other end's link-local address using IPv6 neighbor discovery router advertisements, an IPv6 next-hop is converted into an IPv4 link-local address and a static neighbor entry is installed for this IPv4 link-local address with the MAC address derived from the link-local address of the other end.



It is assumed that the IPv6 implementation on the peering device will use the MAC address as the interface ID when assigning the IPv6 link-local address, as suggested by RFC 4291.

## Configuring BGP Unnumbered Interfaces

Configuring a BGP unnumbered interface requires enabling IPv6 neighbor discovery router advertisements. The `interval` you specify is measured in seconds, and defaults to 600 seconds. Extended next-hop encoding is sent only for the link-local address peerings:

```
interface swp1
  ipv6 nd ra-interval 5
!
router bgp 10
  neighbor swp1 interface
  neighbor swp1 remote-as 20
  neighbor swp1 capability extended-nexthop
!
```

## Managing Unnumbered Interfaces

All the relevant BGP commands are now capable of showing IPv6 next-hops and/or the interface name for any IPv4 prefix:

```
# show ip bgp
BGP table version is 66, local router ID is 192.0.2.5
Status codes: s suppressed, d damped, h history, * valid, > best, =
multipath,
          i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete
      Network          Next Hop            Metric LocPrf Weight Path
*> 192.0.2.5/32    0.0.0.0                  0        32768   ??
*= 192.0.2.6/32    swp2                   0  65534   64503   ??
*=                   swp6                   0  65002   64503   ??
*=                   swp5                   0  65001   64503   ??
```

```
*=
*=          swp4          0 65534 64503 ?
*>         swp3          0 65534 64503 ?

# show ip bgp 192.0.2.14/32
BGP routing table entry for 192.0.2.14/32
Paths: (1 available, best #1, table Default-IP-Routing-Table)
    Advertised to non peer-group peers:
        swp1 swp2 swp3 swp4 swp5 swp6
        65534
            fe80::202:ff:fe00:3d from swp2 (192.0.2.14)
            (fe80::202:ff:fe00:3d) (used)
                Origin incomplete, metric 0, localpref 100, valid, external, best
                Last update: Tue May 12 17:18:41 2015
```

Quagga RIB commands are also modified:

```
# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel, T - Table,
       > - selected route, * - FIB route
K>* 0.0.0.0/0 via 192.168.0.2, eth0
C>* 192.0.2.5/32 is directly connected, lo
B>* 192.0.2.6/32 [20/0] via fe80::202:ff:fe00:45, swp3, 00:46:12
    *
    *           via fe80::202:ff:fe00:35, swp1, 00:46:12
    *           via fe80::202:ff:fe00:3d, swp2, 00:46:12
    *           via fe80::202:ff:fe00:4d, swp4, 00:46:12
    *           via fe80::202:ff:fe00:55, swp5, 00:46:12
    *           via fe80::202:ff:fe00:5a, swp6, 00:46:12
```

The following commands show how the IPv4 link-local address 169.254.0.1 is used to install the route and static neighbor entry to facilitate proper forwarding without having to install an IPv4 prefix with IPv6 next-hop in the kernel:

```
# ip route show 192.0.2.6
192.0.2.6 proto zebra metric 20
    nexthop via 169.254.0.1 dev swp3 weight 1 onlink
    nexthop via 169.254.0.1 dev swp1 weight 1 onlink
    nexthop via 169.254.0.1 dev swp2 weight 1 onlink
    nexthop via 169.254.0.1 dev swp4 weight 1 onlink
    nexthop via 169.254.0.1 dev swp5 weight 1 onlink
    nexthop via 169.254.0.1 dev swp6 weight 1 onlink
```

```
# ip neigh
fe80::202:ff:fe00:35 dev swp1 lladdr 00:02:00:00:00:35 router REACHABLE
fe80::202:ff:fe00:5a dev swp6 lladdr 00:02:00:00:00:5a router REACHABLE
fe80::202:ff:fe00:3d dev swp2 lladdr 00:02:00:00:00:3d router REACHABLE
fe80::202:ff:fe00:55 dev swp5 lladdr 00:02:00:00:00:55 router REACHABLE
fe80::202:ff:fe00:45 dev swp3 lladdr 00:02:00:00:00:45 router REACHABLE
fe80::202:ff:fe00:4d dev swp4 lladdr 00:02:00:00:00:4d router REACHABLE
169.254.0.1 dev swp5 lladdr 00:02:00:00:00:55 PERMANENT
192.168.0.2 dev eth0 lladdr 52:55:c0:a8:00:02 REACHABLE
169.254.0.1 dev swp3 lladdr 00:02:00:00:00:45 PERMANENT
169.254.0.1 dev swp1 lladdr 00:02:00:00:00:35 PERMANENT
169.254.0.1 dev swp4 lladdr 00:02:00:00:00:4d PERMANENT
169.254.0.1 dev swp6 lladdr 00:02:00:00:00:5a PERMANENT
169.254.0.1 dev swp2 lladdr 00:02:00:00:00:3d PERMANENT
```

## How traceroute Interacts with BGP Unnumbered Interfaces

Every router or end host must have an IPv4 address in order to complete a `traceroute` of IPv4 addresses. In this case, the IPv4 address used is that of the loopback device.

Even if ENHE is not used in the data center, link addresses are not typically advertised. This is because:

- Link addresses take up valuable FIB resources. In a large Clos environment, the number of such addresses can be quite large.
- Link addresses expose an additional attack vector for intruders to use to either break in or engage in DDOS attacks.

Therefore, assigning an IP address to the loopback device is essential.

## Advanced: Understanding How Next-hop Fields Are Set

This section describes how the IPv6 next-hops are set in the MP\_REACH\_NLRI ([multiprotocol reachable NLRI](#)) initiated by the system, which applies whether IPv6 prefixes or IPv4 prefixes are exchanged with ENHE. There are two main aspects to determine — how many IPv6 next-hops are included in the MP\_REACH\_NLRI (since the RFC allows either one or two next-hops) and the values of the next-hop(s). This section also describes how a received MP\_REACH\_NLRI is handled as far as processing IPv6 next-hops.

- Whether peering to a global IPv6 address or link-local IPv6 address, the determination whether to send one or two next-hops is as follows:
  1. If reflecting the route, two next-hops are sent only if the peer has `nexthop-local unchanged` configured and the attribute of the received route has an IPv6 link-local next-hop; otherwise, only one next-hop is sent.
  2. Otherwise (if it's not reflecting the route), two next-hops are sent if explicitly configured (`nexthop-local unchanged`) or the peer is directly connected (that is, either peering is on link-local address or the global IPv4 or IPv6 address is *directly connected*) and the route is either a local/self-originated route or the peer is an eBGP peer.
  3. In all other cases, only one next-hop gets sent, unless an outbound route map adds another next-hop.

- `route-map` can impose two next-hops in scenarios where Cumulus Linux would only send one next-hop — by specifying `set ipv6 nexthop link-local`.
- For all routes to eBGP peers and self-originated routes to iBGP peers, the global next-hop (first value) is the peering address of the local system. If the peering is on the link-local address, this is the global IPv6 address on the peering interface, if present; otherwise, it is the link-local IPv6 address on the peering interface.
- For other routes to iBGP peers (eBGP to iBGP or reflected), the global next-hop will be the global next-hop in the received attribute.



If this address were a link-local IPv6 address, it would get reset so that the link-local IPv6 address of the eBGP peer is not passed along to an iBGP peer, which most likely may be on a different link.

- `route-map` and/or the peer configuration can change the above behavior. For example, `route-map` can set the global IPv6 next-hop or the peer configuration can set it to `self` — which is relevant for *iBGP* peers. The route map or peer configuration can also set the next-hop to unchanged, which ensures the source IPv6 global next-hop is passed around — which is relevant for *eBGP* peers.
- Whenever two next-hops are being sent, the link-local next-hop (the second value of the two) is the link-local IPv6 address on the peering interface unless it is due to `nh-local-unchanged` or `route-map` has set the link-local next-hop.
- Network administrators cannot set **martian values** for IPv6 next-hops in `route-map`. Also, global and link-local next-hops are validated to ensure they match the respective address types.
- In a received update, a martian check is imposed for the IPv6 global next-hop. If the check fails, it gets treated as an implicit withdraw.
- If two next-hops are received in an update and the second next-hop is not a link-local address, it gets ignored and the update is treated as if only one next-hop was received.
- Whenever two next-hops are received in an update, the second next-hop is used to install the route into `zebra`. As per the previous point, it is already assured that this is a link-local IPv6 address. Currently, this is assumed to be reachable and is not registered with NHT.
- When `route-map` specifies the next-hop as `peer-address`, the global IPv6 next-hop as well as the link-local IPv6 next-hop (if it's being sent) is set to the *peering address*. If the peering is on a link-local address, the former could be the link-local address on the peering interface, unless there is a global IPv6 address present on this interface.

The above rules imply that there are scenarios where a generated update has two IPv6 next-hops, and both of them are the IPv6 link-local address of the peering interface on the local system. If you are peering with a switch or router that is not running Cumulus Linux and expects the first next-hop to be a global IPv6 address, a route map can be used on the sender to specify a global IPv6 address. This conforms with the recommendations in the Internet draft [draft-kato-bgp-ipv6-link-local-00.txt](#), "BGP4+ Peering Using IPv6 Link-local Address".

## ***Limitations***

- Interface-based peering with separate IPv4 and IPv6 sessions is not supported.
- ENHE is sent for IPv6 link-local peerings only.
- If a IPv4 /30 or /31 IP address is assigned to the interface IPv4 peering will be used over IPv6 link-local peering.

## BGP add-path

In Cumulus Linux 3.0, BGP and static routing (IPv4 and IPv6) are supported within a VRF context. For more information, refer to [Virtual Routing and Forwarding - VRF \(see page 469\)](#).

## BGP add-path RX

*BGP add-path RX* allows BGP to receive multiple paths for the same prefix. A path identifier is used so that additional paths do not override previously advertised paths. No additional configuration is required for BGP add-path RX.



The Add-Path RX capability is advertised by Cumulus Linux BGP by default. Add-Path TX requires the administrator to enable it. Enabling TX will reset the session.

To view the existing capabilities, run `show ip bgp neighbors`. They can be seen listed in the subsection *Add Path*; below *Neighbor capabilities*:

```
r7# show ip bgp neighbors
BGP neighbor is 10.7.8.2, remote AS 800, local AS 700, external link
Hostname: r8

...
Neighbor capabilities:
  4 Byte AS: advertised and received
AddPath:
  IPv4 Unicast: TX advertised IPv4 Unicast
  IPv4 Unicast: RX advertised IPv4 Unicast and received
  Route refresh: advertised and received(old & new)
  Address family IPv4 Unicast: advertised and received
  Hostname Capability: advertised and received
  Graceful Restart Capabilty: advertised and received
  Remote Restart timer is 120 seconds
  Address families by peer:
    none
...
```

The example output above shows that additional BGP paths can be sent and received (TX and RX are advertised). It also shows that the BGP neighbor, `10.7.8.2`, supports both.

To view the current additional paths, run `show ip bgp <network>`:

```
r7# show ip bgp 1.1.1.1/32
BGP routing table entry for 1.1.1.1/32
Paths: (6 available, best #6, table Default-IP-Routing-Table)
      500
```

```

10.7.6.2 from r6(10.7.6.2) (10.0.0.6)
    Origin IGP, metric 0, localpref 100, valid, external
    Community: 6:6
    AddPath ID: RX 0, TX 7
    Last update: Thu Jun 2 00:57:16 2016

500
10.7.5.2 from r5(10.7.5.2) (10.0.0.5)
    Origin IGP, metric 0, localpref 100, valid, external, bestpath-from-
AS 500
    Community: 5:5
    AddPath ID: RX 0, TX 6
    Advertised to: r8(10.7.8.2)
    Last update: Thu Jun 2 00:57:16 2016

300
10.7.4.2 from r4(10.7.4.2) (10.0.0.4)
    Origin IGP, metric 0, localpref 100, valid, external
    Community: 4:4
    AddPath ID: RX 0, TX 5
    Last update: Thu Jun 2 00:57:16 2016

300
10.7.3.2 from r3(10.7.3.2) (10.0.0.3)
    Origin IGP, metric 0, localpref 100, valid, external, bestpath-from-
AS 300
    Community: 3:3
    AddPath ID: RX 0, TX 4
    Advertised to: r8(10.7.8.2)
    Last update: Thu Jun 2 00:57:16 2016

100
10.7.2.2 from r2(10.7.2.2) (10.0.0.2)
    Origin IGP, metric 0, localpref 100, valid, external, multipath
    Community: 2:2
    AddPath ID: RX 0, TX 3
    Last update: Thu Jun 2 00:57:16 2016

100
10.7.1.2 from r1(10.7.1.2) (10.0.0.1)
    Origin IGP, metric 0, localpref 100, valid, external, multipath,
bestpath-from-AS 100, best
    Community: 1:1
    AddPath ID: RX 0, TX 2
    Advertised to: r1(10.7.1.2) r2(10.7.2.2) r3(10.7.3.2) r4(10.7.4.2) r5

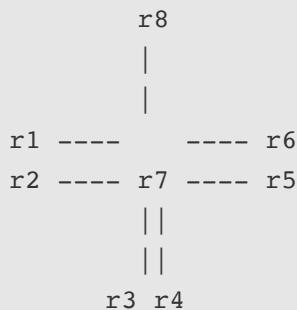
```

```
(10.7.5.2) r6(10.7.6.2) r8(10.7.8.2)
Last update: Thu Jun 2 00:57:16 2016
```

The example output above shows five additional paths that have been added by the TX node for receiving. All the paths have a unique AddPath ID.

## BGP add-path TX

AddPath TX allows BGP to advertise more than just the bestpath for a prefix. Consider the following topology:



In this topology:

- r1 and r2 are in AS 100
- r3 and r4 are in AS 300
- r5 and r6 are in AS 500
- r7 is in AS 700
- r8 is in AS 800
- r7 learns 1.1.1.1/32 from r1, r2, r3, r4, r5, and r6. Among these r7 picks the path from r1 as the bestpath for 1.1.1.1/32

The example below configures the r7 session to advertise the bestpath learned from each AS. In this case, this means a path from AS 100, a path from AS 300, and a path from AS 500. The `show ip bgp 1.1.1.1/32` from r7 has "bestpath-from-AS 100" so the user can see what the bestpath is from each AS:

```
r7# conf t
r7(config)# router bgp 700
r7(config-router)# neighbor 192.0.2.2 addpath-tx-bestpath-per-AS
r7(config-router)#

```

The output below shows the result on r8:

```
r8# show ip bgp 1.1.1.1/32
BGP routing table entry for 1.1.1.1/32
```

```

Paths: (3 available, best #3, table Default-IP-Routing-Table)
Advertised to non peer-group peers:
r7(10.7.8.1)
700 100
  10.7.8.1 from r7(10.7.8.1) (10.0.0.7)
    Origin IGP, localpref 100, valid, external
    Community: 1:1
    AddPath ID: RX 2, TX 4
    Last update: Thu Jun 2 00:57:14 2016

700 300
  10.7.8.1 from r7(10.7.8.1) (10.0.0.7)
    Origin IGP, localpref 100, valid, external
    Community: 3:3
    AddPath ID: RX 4, TX 3
    Last update: Thu Jun 2 00:57:14 2016

700 500
  10.7.8.1 from r7(10.7.8.1) (10.0.0.7)
    Origin IGP, localpref 100, valid, external, bestpath-from-AS 700, best
    Community: 5:5
    AddPath ID: RX 6, TX 2
    Last update: Thu Jun 2 00:57:14 2016

r8#

```

The example below shows the results if r7 is configured to advertise all paths to r8:

```

r7# conf t
r7(config)# router bgp 700
r7(config-router)# neighbor 192.0.2.2 addpath-tx-all-paths
r7(config-router)#

```

The output below shows the result on r8:

```

r8# show ip bgp 1.1.1.1/32
BGP routing table entry for 1.1.1.1/32
Paths: (3 available, best #3, table Default-IP-Routing-Table)
Advertised to non peer-group peers:
r7(10.7.8.1)
700 100
  10.7.8.1 from r7(10.7.8.1) (10.0.0.7)

```

```

Origin IGP, localpref 100, valid, external
Community: 1:1
AddPath ID: RX 2, TX 4
Last update: Thu Jun 2 00:57:14 2016

700 300
10.7.8.1 from r7(10.7.8.1) (10.0.0.7)
Origin IGP, localpref 100, valid, external
Community: 3:3
AddPath ID: RX 4, TX 3
Last update: Thu Jun 2 00:57:14 2016

700 500
10.7.8.1 from r7(10.7.8.1) (10.0.0.7)
Origin IGP, localpref 100, valid, external, bestpath-from-AS 700, best
Community: 5:5
AddPath ID: RX 6, TX 2
Last update: Thu Jun 2 00:57:14 2016
  
```

r8#

## ***Fast Convergence Design Considerations***

Without getting into the why (see the IETF draft cited in Useful Links below that talks about BGP use within the data center), we strongly recommend the following use of addresses in the design of a BGP-based data center network:

- Use of interface addresses: Set up BGP sessions only using interface-scoped addresses. This allows BGP to react quickly to link failures.
- Use of next-hop-self: Every BGP node says that it knows how to forward traffic to the prefixes it is announcing. This reduces the requirement to announce interface-specific addresses and thereby reduces the size of the forwarding table.

## ***Specifying the Interface Name in the neighbor Command***

When you are configuring BGP for the neighbors of a given interface, you can specify the interface name instead of its IP address. All the other **neighbor** command options remain the same.

This is equivalent to BGP peering to the link-local IPv6 address of the neighbor on the given interface. The link-local address is learned via IPv6 neighbor discovery router advertisements.

Consider the following example configuration:

```

router bgp 65000
  bgp router-id 0.0.0.1
  neighbor swp1 interface
  neighbor swp1 remote-as 65000
  
```

```

neighbor swp1 next-hop-self
!
address-family ipv6
neighbor swp1 activate
exit-address-family

```



By default, Cumulus Linux sends IPv6 neighbor discovery router advertisements. Cumulus Networks recommends you adjust the router advertisement's interval to a shorter value (`ipv6 nd ra-interval <interval>`) to address scenarios when nodes come up and miss router advertisement processing to relay the neighbor's link-local address to BGP. The `interval` is measured in seconds and defaults to 600 seconds.

## Configuring BGP Peering Relationships across Switches

A BGP peering relationship is typically initiated with the `neighbor x.x.x.x remote-as <AS number>` command. In order to simplify configuration across multiple switches, you can specify the *internal* or *external* keyword to the configuration instead of the AS number.

Specifying *internal* signifies an iBGP peering; that is, the neighbor will only create or accept a connection with the specified neighbor if the remote peer AS number matches this BGP's AS number.

Specifying *external* signifies an eBGP peering; that is, the neighbor will only create a connection with the neighbor if the remote peer AS number does **not** match this BGP AS number.

You can make this distinction using the `neighbor` command or the `peer-group` command.

In general, use the following syntax with the `neighbor` command:

```

neighbor (ipv4 addr|ipv6 addr|WORD) remote-as (<1-
4294967295>|internal|external)

```

Some example configurations follow.

To connect to **the same AS** using the `neighbor` command, modify your configuration similar to the following:

```

router bgp 500
neighbor 192.168.1.2 remote-as internal

```

To connect to a **different AS** using the `neighbor` command, modify your configuration similar to the following:

```

router bgp 500
neighbor 192.168.1.2 remote-as external

```

To connect to **the same AS** using the `peer-group` command, modify your configuration similar to the following:

```
router bgp 500
neighbor swp1 interface
neighbor IBGP peer-group
neighbor IBGP remote-as internal
neighbor swp1 peer-group IBGP
neighbor 192.0.2.3 peer-group IBGP
neighbor 192.0.2.4 peer-group IBGP
```

To connect to a **different AS** using the `peer-group` command, modify your configuration similar to the following:

```
router bgp 500
neighbor swp2 interface
neighbor EBGP peer-group
neighbor EBGP remote-as external
neighbor 192.0.2.2 peer-group EBGP
neighbor swp2 peer-group EBGP
neighbor 192.0.2.4 peer-group EBGP
```

## Configuration Tips

### Using `peer-group` to Simplify Configuration

When there are many peers to connect to, the amount of redundant configuration becomes overwhelming. For example, repeating the `activate` and `next-hop-self` commands for even 60 neighbors makes for a very long configuration file. Using `peer-group` addresses this problem.

Instead of specifying properties of each individual peer, Quagga allows for defining one or more peer-groups and associating all the attributes common to that peer session to a peer-group.

After doing this, the only task is to associate an IP address with a peer-group. Here is an example of defining and using peer-groups:

```
R7(config-router)# neighbor tier-2 peer-group
R7(config-router)# neighbor tier-2 remote-as 65000
R7(config-router)# address-family ipv4 unicast
R7(config-router-af)# neighbor tier-2 activate
R7(config-router-af)# neighbor tier-2 next-hop-self
R7(config-router-af)# maximum-paths ibgp 64
R7(config-router-af)# exit
```

```
R7(config-router)# neighbor 10.0.0.2 peer-group tier-2
R7(config-router)# neighbor 192.0.2.2 peer-group tier-2
```

If you're using eBGP, besides specifying the neighbor's IP address, you also have to specify the neighbor's ASN, since it is different for each neighbor. In such a case, you wouldn't specify the `remote-as` for the peer-group.

## ***Preserving the AS\_PATH Setting***

If you plan to use multipathing with the `multipath-relax` option, Quagga generates an AS\_SET in place of the current AS\_PATH for the bestpath. This helps to prevent loops but is unusual behavior. To preserve the AS\_PATH setting, use the `no-as-set` option when configuring bestpath:

```
R7(config-router)# bgp bestpath as-path multipath-relax no-as-set
```

## ***Utilizing Multiple Routing Tables and Forwarding***

You can run multiple routing tables (one for in-band/data plane traffic and one for out-of-band/management plane traffic) on the same switch using [management VRF \(see page 489\)](#) (multiple routing tables and forwarding).

## ***Troubleshooting***

The most common starting point for troubleshooting BGP is to view the summary of neighbors connected to and some information about these connections. A sample output of this command is as follows:

```
R7# show ip bgp summary
BGP router identifier 0.0.0.9, local AS number 65000
RIB entries 7, using 672 bytes of memory
Peers 2, using 9120 bytes of memory

Neighbor          V     AS MsgRcvd MsgSent      TblVer  InQ OutQ Up/Down  State
/PfxRcd
10.0.0.2          4 65000       11       10          0     0     0 00:06:38      3
192.0.2.2          4 65000       11       10          0     0     0 00:06:38      3

Total number of neighbors 2
```

(Pop quiz: Are these iBGP or eBGP sessions? Hint: Look at the ASNs.)

It is also useful to view the routing table as defined by BGP:

```
R7# show ip bgp
BGP table version is 0, local router ID is 0.0.0.9
```

Status codes: s suppressed, d damped, h history, \* valid, > best, i - internal,

r RIB-failure, S Stale, R Removed

Origin codes: i - IGP, e - EGP, ? - incomplete

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 192.0.2.29/24	0.0.0.0	0		32768	i
*>i192.0.2.30/24	10.0.0.2	0	100	0	i
* i	192.0.2.2	0	100	0	i
*>i192.0.2.31/24	10.0.0.2	0	100	0	i
* i	192.0.2.2	0	100	0	i
*>i192.0.2.32/24	10.0.0.2	0	100	0	i
* i	192.0.2.2	0	100	0	i

Total number of prefixes 4

A more detailed breakdown of a specific neighbor can be obtained using `show ip bgp neighbor <neighbor ip address>`:

```
R7# show ip bgp neighbor 10.0.0.2
BGP neighbor is 10.0.0.2, remote AS 65000, local AS 65000, internal link
BGP version 4, remote router ID 0.0.0.5
BGP state = Established, up for 00:14:03
Last read 14:52:31, hold time is 180, keepalive interval is 60 seconds
Neighbor capabilities:
```

- 4 Byte AS: advertised and received
- Route refresh: advertised and received(old & new)
- Address family IPv4 Unicast: advertised and received

Message statistics:

- Inq depth is 0
- Outq depth is 0

	Sent	Rcvd
Opens:	1	1
Notifications:	0	0
Updates:	1	3
Keepalives:	16	15
Route Refresh:	0	0
Capability:	0	0
Total:	18	19

Minimum time between advertisement runs is 5 seconds

For address family: IPv4 Unicast  
 NEXT\_HOP is always this router

```

Community attribute sent to this neighbor(both)
3 accepted prefixes

Connections established 1; dropped 0
Last reset never
Local host: 10.0.0.1, Local port: 35258
Foreign host: 10.0.0.2, Foreign port: 179
Nexthop: 10.0.0.1
Nexthop global: fe80::202:ff:fe00:19
Nexthop local: ::

BGP connection: non shared network
Read thread: on Write thread: off

```

To see the details of a specific route such as from whom it was received, to whom it was sent, and so forth, use the `show ip bgp <ip address/prefix>` command:

```

R7# show ip bgp 192.0.2.0
BGP routing table entry for 192.0.2.0/24
Paths: (2 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  Local
    10.0.0.2 (metric 1) from 10.0.0.2 (0.0.0.10)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Originator: 0.0.0.10, Cluster list: 0.0.0.5
      Last update: Mon Jul  8 10:12:17 2013
  Local
    192.0.2.2 (metric 1) from 192.0.2.2 (0.0.0.10)
      Origin IGP, metric 0, localpref 100, valid, internal
      Originator: 0.0.0.10, Cluster list: 0.0.0.6
      Last update: Mon Jul  8 10:12:17 2013

```

This shows that the routing table prefix seen by BGP is 192.0.2.0/24, that this route was not advertised to any neighbor, and that it was heard by two neighbors, 10.0.0.2 and 192.0.2.2.

Here is another output of the same command, on a different node in the network:

```

cumulus@switch:~$ sudo vtysh -c 'sh ip bgp 192.0.2.0'
BGP routing table entry for 192.0.2.0/24
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
    10.0.0.1 192.0.2.21 192.0.2.22
  Local, (Received from a RR-client)
    203.0.113.1 (metric 1) from 203.0.113.1 (0.0.0.10)

```

```
Origin IGP, metric 0, localpref 100, valid, internal, best
Last update: Mon Jul  8 09:07:41 2013
```

## **Debugging Tip: Logging Neighbor State Changes**

It is very useful to log the changes that a neighbor goes through to troubleshoot any issues associated with that neighbor. This is done using the `log-neighbor-changes` command:

```
R7(config-router)# bgp log-neighbor-changes
```

The output is sent to the specified log file, usually `/var/log/quagga/bgpd.log`, and looks like this:

```
2013/07/08 10:12:06.572827 BGP: %NOTIFICATION: sent to neighbor 10.0.0.2 6
/3 (Cease/Peer Unconfigured) 0 bytes
2013/07/08 10:12:06.572954 BGP: Notification sent to neighbor 10.0.0.2:
type 6/3
2013/07/08 10:12:16.682071 BGP: %ADJCHANGE: neighbor 192.0.2.2 Up
2013/07/08 10:12:16.682660 BGP: %ADJCHANGE: neighbor 10.0.0.2 Up
```

## **Troubleshooting Link-local Addresses**

To verify that `quagga` learned the neighboring link-local IPv6 address via the IPv6 neighbor discovery router advertisements on a given interface, use the `show interface <if-name>` command. If `ipv6 nd suppress-ra` isn't enabled on both ends of the interface, then `Neighbor address(s):` should have the other end's link-local address. That is the address that BGP would use when BGP is enabled on that interface.

Use `vtysh` to run `quagga`, then verify the configuration:

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R7# show interface swp1
Interface swp1 is up, line protocol is up
  PTM status: disabled
  Description: rut
  index 3 metric 1 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:09
  inet 11.0.0.1/24 broadcast 11.0.0.255
  inet6 fe80::202:ff:fe00:9/64
```

```

ND advertised reachable time is 0 milliseconds
ND advertised retransmit interval is 0 milliseconds
ND router advertisements are sent every 600 seconds
ND router advertisements lifetime tracks ra-interval
ND router advertisement default router preference is medium
Hosts use stateless autoconfig for addresses.

Neighbor address(s):
inet6 fe80::4638:39ff:fe00:129b/128

```

Instead of the IPv6 address, the peering interface name is displayed in the `show ip bgp summary` command and wherever else applicable:

```

R7# show ip bgp summary
BGP router identifier 0.0.0.1, local AS number 65000
RIB entries 1, using 112 bytes of memory
Peers 1, using 8712 bytes of memory

Neighbor          V     AS MsgRcvd MsgSent    TblVer  InQ OutQ Up/Down  State
/PfxRcd
swp1            4  65000      161       170        0      0    0 00:02:28        0

```

Most of the show commands can take the interface name instead of the IP address, if that level of specificity is needed:

```

R7# show ip bgp neighbors
<cr>
A.B.C.D  Neighbor to display information about
WORD      Neighbor on bgp configured interface
X:X::X:X  Neighbor to display information about
R7# show ip bgp neighbors swp1

```

## ***Enabling Read-only Mode***

You can enable read-only mode for when the BGP process restarts or when the BGP process is cleared using `clear ip bgp *`. When enabled, read-only mode begins as soon as the first peer reaches its *established* state and a timer for `<max-delay>` seconds is started.

While in read-only mode, BGP doesn't run best-path or generate any updates to its peers. This mode continues until:

- All the configured peers, except the shutdown peers, have sent an explicit EOR (End-Of-RIB) or an implicit EOR. The first keep-alive after BGP has reached the established state is considered an implicit EOR. If the `<establish-wait>` option is specified, then BGP will wait for peers to reach the established state from the start of the `update-delay` until the `<establish-wait>` period is over; that is, the minimum set of established peers for which EOR is expected would be peers established during the `establish-wait` window, not necessarily all the configured neighbors.
- The `max-delay` period is over.

Upon reaching either of these two conditions, BGP resumes the decision process and generates updates to its peers.

To enable read-only mode:

```
cumulus@switch:$ sudo bgp update-delay <max-delay in seconds> [<establish-
wait in seconds>]
```

The default `<max-delay>` is 0 — the feature is off by default.

Use output from `show ip bgp summary` for information about the state of the update delay.

This feature can be useful in reducing CPU/network usage as BGP restarts/clears. It's particularly useful in topologies where BGP learns a prefix from many peers. Intermediate best paths are possible for the same prefix as peers get established and start receiving updates at different times. This feature is also valuable if the network has a high number of such prefixes.

## **Applying a Route Map for Route Updates**

You can apply a route map on route updates from BGP to Zebra. All the applicable match operations are allowed, such as match on prefix, next-hop, communities, and so forth. Set operations for this attach-point are limited to metric and next-hop only. Any operation of this feature does not affect BGPs internal RIB.

Both IPv4 and IPv6 address families are supported. Route maps work on multi-paths as well. However, the metric setting is based on the best path only.

To apply a route map for route updates:

```
cumulus@switch:$ sudo cl-bgp table-map <route-map-name>
```

## **Protocol Tuning**

### **Converging Quickly On Link Failures**

In the Clos topology, we recommend that you only use interface addresses to set up peering sessions. This means that when the link fails, the BGP session is torn down immediately, triggering route updates to propagate through the network quickly. This requires the following commands be enabled for all links: `link-detect` and `ttl-security hops <hops>`. `ttl-security hops` specifies how many hops away the neighbor is. For example, in a Clos topology, every peer is at most 1 hop away.



See Caveats and Errata below for information regarding `ttl-security hops`.

Here is an example:

```
cumulus@switch:~$ sudo vtysh

Hello, this is Quagga (version 0.99.23.1+cl3.0).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R7# configure terminal
R7(config)# interface swp1
R7(config-if)# link-detect
R7(config-if)# exit
R7(config)# router bgp 65000
R7(config-router)# neighbor 10.0.0.2 ttl-security hops 1
```

## **Converging Quickly On Soft Failures**

It is possible that the link is up, but the neighboring BGP process is hung or has crashed. If a BGP process crashes, Quagga's **watchquagga** daemon, which monitors the various **quagga** daemons, will attempt to restart it. If the process is also hung, **watchquagga** will attempt to restart the process. BGP itself has a keepalive timer that is exchanged between neighbors. By default, this keepalive timer is set to 60 seconds. This time can be reduced to a lower number, but this has the disadvantage of increasing the CPU load, especially in the presence of a lot of neighbors. **keepalive-time** is the periodicity with which the keepalive message is sent. **hold-time** specifies how many keepalive messages can be lost before the connection is considered invalid. It is usually set to 3 times the keepalive time. Here is an example of reducing these timers:

```
R7(config-router)# neighbor 10.0.0.2 timers 30 90
```

We can make these the default for all BGP neighbors using a different command:

```
R7(config-router)# timers bgp 30 90
```

The following display snippet shows that the default values have been modified for this neighbor:

```
R7(config-router)# do show ip bgp neighbor 10.0.0.2
BGP neighbor is 10.0.0.2, remote AS 65000, local AS 65000, internal link
  BGP version 4, remote router ID 0.0.0.5
  BGP state = Established, up for 05:53:59
  Last read 14:53:25, hold time is 180, keepalive interval is 60 seconds
  Configured hold time is 90, keepalive interval is 30 seconds
  ....
```



When you're in a configuration mode, such as when you're configuring BGP parameters, you can run any show command by adding `do` to the original command. For example, `do show ip bgp neighbor` was shown above. Under a non-configuration mode, you'd simply run:

```
show ip bgp neighbor 10.0.0.2
```

## Reconnecting Quickly

A BGP process attempts to connect to a peer after a failure (or on startup) every `connect-time` seconds. By default, this is 120 seconds. To modify this value, use:

```
R7(config-router)# neighbor 10.0.0.2 timers connect 30
```

This command has to be specified per each neighbor, peer-group doesn't support this option in `quagga`.

## Advertisement Interval

BGP by default chooses stability over fast convergence. This is very useful when routing for the Internet. For example, unlike link-state protocols, BGP typically waits for a duration of `advertisement-interval` seconds between sending consecutive updates to a neighbor. This ensures that an unstable neighbor flapping routes won't be propagated throughout the network. By default, this is set to 30 seconds for an eBGP session and 5 seconds for an iBGP session. For very fast convergence, set the timer to 0 seconds. You can modify this as follows:

```
R7(config-router)# neighbor 10.0.0.2 advertisement-interval 0
```

The following output shows the modified value:

```
R7(config-router)# do show ip bgp neighbor 10.0.0.2
BGP neighbor is 10.0.0.2, remote AS 65000, local AS 65000, internal link
  BGP version 4, remote router ID 0.0.0.5
  BGP state = Established, up for 06:01:49
  Last read 14:53:15, hold time is 180, keepalive interval is 60 seconds
  Configured hold time is 90, keepalive interval is 30 seconds
Neighbor capabilities:
  4 Byte AS: advertised and received
  Route refresh: advertised and received(old & new)
  Address family IPv4 Unicast: advertised and received
```

```

Message statistics:
  Inq depth is 0
  Outq depth is 0
      Sent          Rcvd
  Opens:           1           1
  Notifications:  0           0
  Updates:        1           3
  Keepalives:     363         362
  Route Refresh:  0           0
  Capability:    0           0
  Total:          365         366
Minimum time between advertisement runs is 0 seconds
.....

```



This command is not supported with peer-groups.

See this [IETF draft](#) for more details on the use of this value.

## Configuration Files

- /etc/quagga/bgpd.conf

## Useful Links

- Bidirectional forwarding detection (see page 446) (BFD) and BGP
- Wikipedia entry for BGP (includes list of useful RFCs)
- Quagga online documentation for BGP (may not be up to date)
- IETF draft discussing BGP use within data centers

## Caveats and Errata

### **ttl-security Issue**

Enabling `ttl-security` does not cause the hardware to be programmed with the relevant information. This means that frames will come up to the CPU and be dropped there. It is recommended that you use the `cl-acltool` command to explicitly add the relevant entry to hardware.

For example, you can configure a file, like `/etc/cumulus/acl/policy.d/01control_plane_bgp.rules`, with a rule like this for TTL:

```

INGRESS_INTF = swp1
INGRESS_CHAIN = INPUT, FORWARD

[iptables]

```

```
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --dport bgp -m
  ttl --ttl 255 POLICE --set-mode pkt --set-rate 2000 --set-burst 1000
-A $INGRESS_CHAIN --in-interface $INGRESS_INTF -p tcp --dport bgp DROP
```



For more information about ACLs and `cl-acltool`, see [Netfilter \(ACLs\)](#) (see page 95).

## Bidirectional Forwarding Detection - BFD

*Bidirectional Forwarding Detection* (BFD) provides low overhead and rapid detection of failures in the paths between two network devices. It provides a unified mechanism for link detection over all media and protocol layers. Use BFD to detect failures for IPv4 and IPv6 single or multihop paths between any two network devices, including unidirectional path failure detection.



Cumulus Linux does not support demand mode in BFD.

### Using BFD Multihop Routed Paths

BFD multihop sessions are built over arbitrary paths between two systems, which results in some complexity that does not exist for single hop sessions. Here are some best practices for using multihop paths:

- **Spoofing:** To avoid spoofing with multihop paths, configure `max_hop_cnt` (maximum hop count) for each peer, which limits the number of hops for a BFD session. All BFD packets exceeding the max hop count will be dropped.
- **Demultiplexing:** Since multihop BFD sessions can take arbitrary paths, demultiplex the initial BFD packet based on the source/destination IP address pair. Use Quagga, which monitors connectivity to the peer, to determine the source/destination IP address pairs.

Multihop BFD sessions are supported for both IPv4 and IPv6 peers. See below for more details.

### BFD Parameters

You can configure the following BFD parameters for both IPv4 and IPv6 sessions:

- The required minimum interval between the received BFD control packets.
- The minimum interval for transmitting BFD control packets.
- The detection time multiplier.

### Configuring BFD

You configure BFD one of two ways: by specifying the configuration in the `PTM topology.dot` file (see page 198), or using [Quagga](#) (see page 391).

The Quagga CLI can track IPv4 and IPv6 peer connectivity — both single hop and multihop, and both link-local IPv6 peers and global IPv6 peers — using BFD sessions without needing the `topology.dot` file. Use Quagga to register multihop peers with PTM and BFD as well as for monitoring the connectivity to the

remote BGP multihop peer. Quagga can dynamically register and unregister both IPv4 and IPv6 peers with BFD when the BFD-enabled peer connectivity is established or de-established, respectively. Also, you can configure BFD parameters for each BGP or OSPF peer using Quagga.



The BFD parameter configured in the topology file is given higher precedence over the client-configured BFD parameters for a BFD session that has been created by both topology file and client (Quagga).

## BFD in BGP

For Quagga when using **BGP**, neighbors are registered and de-registered with PTM (see page 198) dynamically when you enable BFD in BGP:

```
quagga(config)# router bgp X
quagga(config-router)# neighbor <neighbor ip> bfd
```

You can configure BFD parameters for each BGP neighbor. For example:

### BFD in BGP

```
quagga(config-router)# neighbor <neighbor ip> bfd
<2-255> Detect Multiplier
<cr>
quagga(config-router)# neighbor <neighbor ip> bfd 4
<50-60000> Required min receive interval
quagga(config-router)# neighbor <neighbor ip> bfd 4 400
<50-60000> Desired min transmit interval
quagga(config-router)# neighbor <neighbor ip> bfd 4 400 400
<cr>
quagga(config-router)# neighbor <neighbor ip> bfd 4 400 400
```

To see neighbor information in BGP, including BFD status, run `show bgp neighbors <IP address>`.

### Show BGP Neighbor

```
quagga# show bgp neighbors 12.12.12.1
BGP neighbor is 12.12.12.1, remote AS 65001, local AS 65000, external
link
Hostname: r1
    BGP version 4, remote router ID 0.0.0.1
    BGP state = Established, up for 00:01:39
    Last read 00:00:39, Last write 00:01:09
    Hold time is 180, keepalive interval is 60 seconds
    Neighbor capabilities:
        4 Byte AS: advertised and received
```

```

AddPath:
  IPv4 Unicast: RX advertised and received
  Route refresh: advertised and received(old & new)
  Address family IPv4 Unicast: advertised and received
  Hostname Capability: advertised and received
  Graceful Restart Capability: advertised and received
    Remote Restart timer is 120 seconds
  Address families by peer:
    none
Graceful restart informations:
  End-of-RIB send: IPv4 Unicast
  End-of-RIB received: IPv4 Unicast
Message statistics:
  Inq depth is 0
  Outq depth is 0
      Sent          Rcvd
  Opens:           1           1
  Notifications: 0           0
  Updates:        2           2
  Keepalives:     2           1
  Route Refresh:  0           0
  Capability:    0           0
  Total:          5           4
Minimum time between advertisement runs is 30 seconds
Update source is 12.12.12.7

For address family: IPv4 Unicast
  Update group 1, subgroup 1
  Packet Queue length 0
  NEXT_HOP is always this router
  Community attribute sent to this neighbor(both)
  1 accepted prefixes
  Connections established 1; dropped 0
  Last reset never
  External BGP neighbor may be up to 2 hops away.
Local host: 12.12.12.7, Local port: 34274
Foreign host: 12.12.12.1, Foreign port: 179
Nexthop: 12.12.12.7
Nexthop global: ::
Nexthop local: ::
BGP connection: non shared network
Read thread: on Write thread: off
  BFD: Type: multi hop
    Detect Mul: 3, Min Rx interval: 300, Min Tx interval: 300
    Status: Down, Last update: 0:00:00:13

```

## BFD in OSPF

For Quagga using **OSPF**, neighbors are registered and de-registered dynamically with [PTM](#) (see page 198) when you enable or disable BFD in OSPF. A neighbor is registered with BFD when two-way adjacency is established and deregistered when adjacency goes down if the BFD is enabled on the interface. The BFD configuration is per interface and any IPv4 and IPv6 neighbors discovered on that interface inherit the configuration.

### BFD in OSPF

```
quagga(config)# interface X
quagga(config-if)# ipv6 ospf6 bfd
<2-255> Detect Multiplier
<cr>
quagga(config-if)# ipv6 ospf6 bfd 5
<50-60000> Required min receive interval
quagga(config-if)# ipv6 ospf6 bfd 5 500
<50-60000> Desired min transmit interval
quagga(config-if)# ipv6 ospf6 bfd 5 500 500
<cr>
quagga(config-if)# ipv6 ospf6 bfd 5 500 500
```

## OSPF Show Commands

The BFD lines at the end of each code block shows the corresponding IPv6 or IPv4 OSPF interface or neighbor information.

### Show IPv6 OSPF Interface

```
quagga# show ipv6 ospf6 interface swp2s0
swp2s0 is up, type BROADCAST
  Interface ID: 4
  Internet Address:
    inet : 11.0.0.21/30
    inet6: fe80::4638:39ff:fe00:6c8e/64
  Instance ID 0, Interface MTU 1500 (autodetect: 1500)
  MTU mismatch detection: enabled
  Area ID 0.0.0.0, Cost 10
  State PointToPoint, Transmit Delay 1 sec, Priority 1
  Timer intervals configured:
    Hello 10, Dead 40, Retransmit 5
  DR: 0.0.0.0 BDR: 0.0.0.0
  Number of I/F scoped LSAs is 2
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
  BFD: Detect Mul: 3, Min Rx interval: 300, Min Tx interval: 300
```

### Show IPv6 OSPF Neighbor

```
quagga# show ipv6 ospf6 neighbor detail
Neighbor 0.0.0.4%swp2s0
  Area 0.0.0.0 via interface swp2s0 (ifindex 4)
  His IfIndex: 3 Link-local address: fe80::202:ff:fe00:a
  State Full for a duration of 02:32:33
  His choice of DR/BDR 0.0.0.0/0.0.0.0, Priority 1
  DbDesc status: Slave SeqNum: 0x76000000
  Summary-List: 0 LSAs
  Request-List: 0 LSAs
  Retrans-List: 0 LSAs
  0 Pending LSAs for DbDesc in Time 00:00:00 [thread off]
  0 Pending LSAs for LSReq in Time 00:00:00 [thread off]
  0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
  0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
  BFD: Type: single hop
    Detect Mul: 3, Min Rx interval: 300, Min Tx interval: 300
    Status: Up, Last update: 0:00:00:20
```

### Show IPv4 OSPF Interface

```
quagga# show ip ospf interface swp2s0
swp2s0 is up
  ifindex 4, MTU 1500 bytes, BW 0 Kbit <UP,BROADCAST,RUNNING,
MULTICAST>
  Internet Address 11.0.0.21/30, Area 0.0.0.0
  MTU mismatch detection:enabled
  Router ID 0.0.0.3, Network Type POINTOPOINT, Cost: 10
  Transmit Delay is 1 sec, State Point-To-Point, Priority 1
  No designated router on this network
  No backup designated router on this network
  Multicast group memberships: OSPFAllRouters
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s,
Retransmit 5
    Hello due in 7.056s
  Neighbor Count is 1, Adjacent neighbor count is 1
  BFD: Detect Mul: 5, Min Rx interval: 500, Min Tx interval: 500
```

### Show IPv4 OSPF Neighbor

```
quagga# show ip ospf neighbor detail
Neighbor 0.0.0.4, interface address 11.0.0.22
  In the area 0.0.0.0 via interface swp2s0
  Neighbor priority is 1, State is Full, 5 state changes
  Most recent state change statistics:
    Progressive change 3h59m04s ago
```

```
DR is 0.0.0.0, BDR is 0.0.0.0
Options 2 *|-|-|-|E|*
Dead timer due in 38.501s
Database Summary List 0
Link State Request List 0
Link State Retransmission List 0
Thread Inactivity Timer on
Thread Database Description Retransmission off
Thread Link State Request Retransmission on
Thread Link State Update Retransmission on
BFD: Type: single hop
    Detect Mul: 5, Min Rx interval: 500, Min Tx interval: 500
    Status: Down, Last update: 0:00:01:29
```

## Troubleshooting BFD

To troubleshoot BFD, use `ptmctl -b`. For more information, see [Prescriptive Topology Manager - PTM](#) (see page 198).

## Equal Cost Multipath Load Sharing - Hardware ECMP

Cumulus Linux supports hardware-based equal cost multipath (ECMP) load sharing. ECMP is enabled by default in Cumulus Linux. Load sharing occurs automatically for all routes with multiple next hops installed. ECMP load sharing supports both IPv4 and IPv6 routes.

### Contents

(Click to expand)

- [Contents \(see page 451\)](#)
- [Understanding Equal Cost Routing \(see page 452\)](#)
- [Understanding ECMP Hashing \(see page 452\)
  - \[Using cl-ecmpcalc to Determine the Hash Result \\(see page 453\\)\]\(#\)
  - \[cl-ecmpcalc Limitations \\(see page 453\\)\]\(#\)
  - \[ECMP Hash Buckets \\(see page 454\\)\]\(#\)](#)
- [Resilient Hashing \(see page 455\)
  - \[Resilient Hash Buckets \\(see page 456\\)\]\(#\)
  - \[Removing Next Hops \\(see page 456\\)\]\(#\)
  - \[Adding Next Hops \\(see page 458\\)\]\(#\)
  - \[Configuring Resilient Hashing \\(see page 458\\)\]\(#\)](#)
- [Caveats \(see page 459\)](#)
- [Useful Links \(see page 459\)](#)

## ***Understanding Equal Cost Routing***

ECMP operates only on equal cost routes in the Linux routing table.

In this example, the 10.1.1.0/24 route has two possible next hops that have been installed in the routing table:

```
$ ip route show 10.1.1.0/24
10.1.1.0/24 proto zebra metric 20
nexthop via 192.168.1.1 dev swp1 weight 1 onlink
nexthop via 192.168.2.1 dev swp2 weight 1 onlink
```

For routes to be considered equal they must:

- Originate from the same routing protocol. Routes from different sources are not considered equal. For example, a static route and an OSPF route are not considered for ECMP load sharing.
- Have equal cost. If two routes from the same protocol are unequal, only the best route is installed in the routing table.



As of Cumulus Linux 3.0, the BGP `maximum-paths` setting is enabled, so multiple routes are installed by default. See the [ECMP section \(see page 422\)](#) of the BGP chapter for more information.

## ***Understanding ECMP Hashing***

Once multiple routes are installed in the routing table, a hash is used to determine which path a packet follows.

Cumulus Linux hashes on the following fields:

- IP protocol
- Ingress interface
- Source IPv4 or IPv6 address
- Destination IPv4 or IPv6 address

For TCP/UDP frames, Cumulus Linux also hashes on:

- Source port
- Destination port

ECMP Hash Fields

Source IP	Destination IP	Layer 4 Protocol	Source Port	Destination Port	Payload
-----------	----------------	------------------	-------------	------------------	---------

To prevent out of order packets, ECMP hashing is done on a per-flow basis, which means that all packets with the same source and destination IP addresses and the same source and destination ports always hash to the same next hop. ECMP hashing does not keep a record of flow states.

ECMP hashing does not keep a record of packets that have hashed to each next hop and does not guarantee that traffic sent to each next hop is equal.

## ***Using cl-ecmpcalc to Determine the Hash Result***

Since the hash is deterministic and always provides the same result for the same input, you can query the hardware and determine the hash result of a given input. This is useful when determining exactly which path a flow takes through a network.

On Cumulus Linux, use the **cl-ecmpcalc** command to determine a hardware hash result.

In order to use **cl-ecmpcalc**, all fields that are used in the hash must be provided. This includes ingress interface, layer 3 source IP, layer 3 destination IP, layer 4 source port and layer 4 destination port.

```
$ sudo cl-ecmpcalc -i swp1 -s 10.0.0.1 -d 10.0.0.1 -p tcp --sport 20000 --
dport 80
ecmpcalc: will query hardware
swp3
```

If any field is omitted, **cl-ecmpcalc** fails.

```
$ sudo cl-ecmpcalc -i swp1 -s 10.0.0.1 -d 10.0.0.1 -p tcp
ecmpcalc: will query hardware
usage: cl-ecmpcalc [-h] [-v] [-p PROTOCOL] [-s SRC] [--sport SPORT] [-d
DST]
                  [--dport DPORT] [--vid VID] [-i IN_INTERFACE]
                  [--sportid SPORTID] [--smoid SMOID] [-o OUT_INTERFACE]
                  [--dportid DPORTID] [--dmodid DMODID] [--hardware]
                  [--nohardware] [-hs HASHSEED]
                  [-hf HASHFIELDS [HASHFIELDS ...]]
                  [--hashfunction {crc16-ccitt,crc16-bisync}] [-e EGRESS]
                  [-c MCOUNT]
```

```
cl-ecmpcalc: error: --sport and --dport required for TCP and UDP frames
```

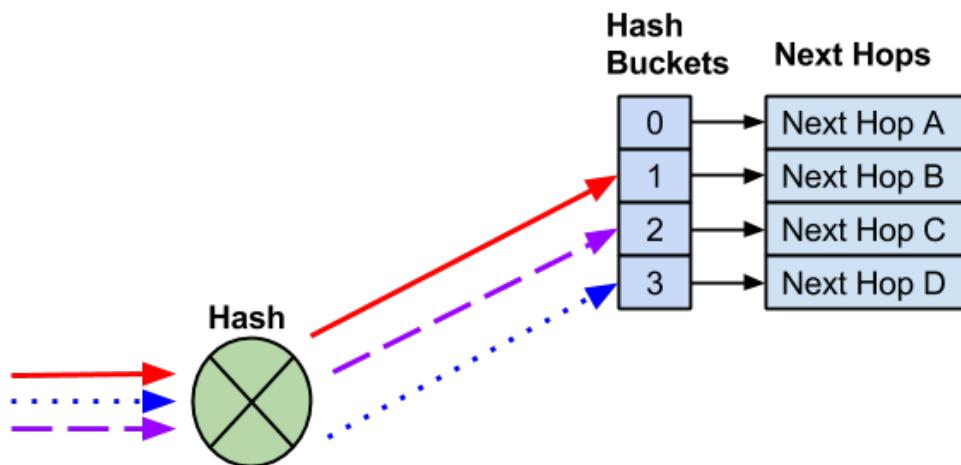
## ***cl-ecmpcalc Limitations***

**cl-ecmpcalc** can only take input interfaces that can be converted to a single physical port in the port tab file, like the physical switch ports (swp). Virtual interfaces like bridges, bonds, and subinterfaces are not supported.

## ECMP Hash Buckets

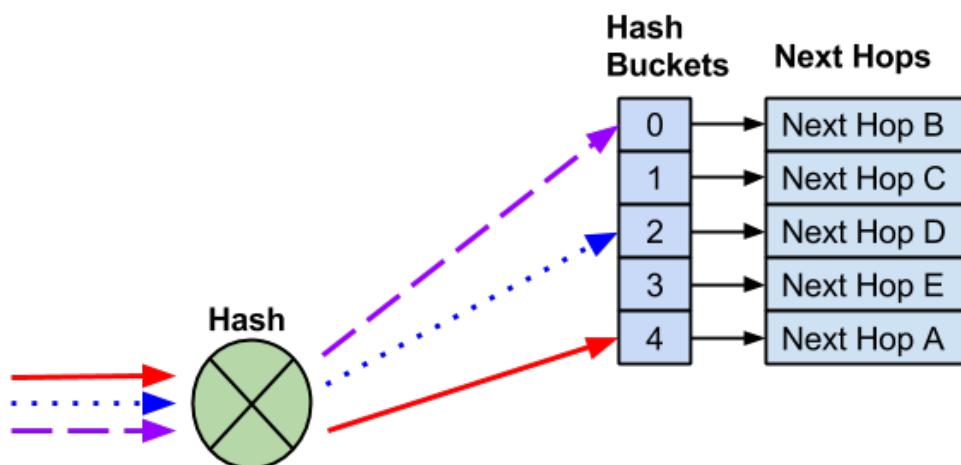
When multiple routes are installed in the routing table, each route is assigned to an ECMP *bucket*. When the ECMP hash is executed the result of the hash determines which bucket gets used.

In the following example, 4 next hops exist. Three different flows are hashed to different hash buckets. Each next hop is assigned to a unique hash bucket.



## Adding a Next Hop

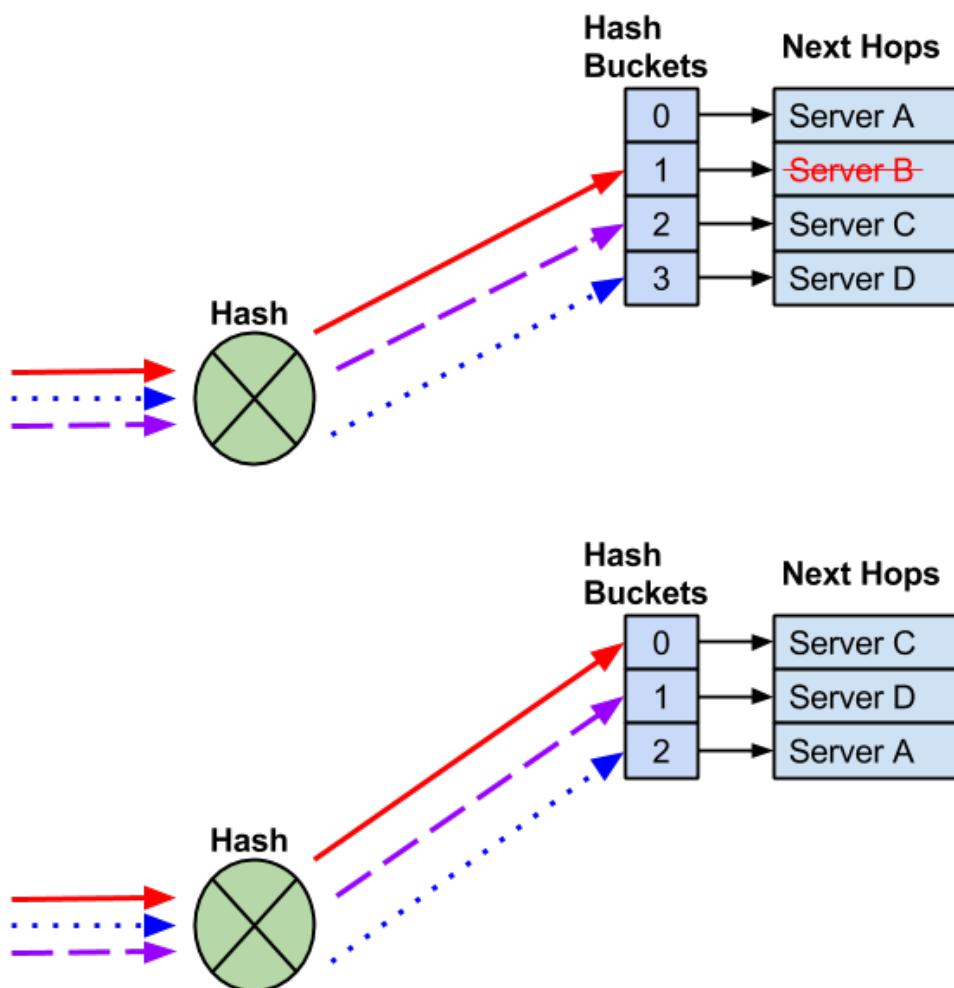
When a next hop is added, a new hash bucket is created. The assignment of next hops to hash buckets, as well as the hash result, may change when additional next hops are added.



A new next hop is added and a new hash bucket is created. As a result, the hash and hash bucket assignment changed, causing the existing flows to be sent to different next hops.

## Removing a Next Hop

When a next hop is removed, the remaining hash bucket assignments may change, again, potentially changing the next hop selected for an existing flow.



A next hop fails and the next hop and hash bucket are removed. The remaining next hops may be reassigned.

In most cases, the modification of hash buckets has no impact on traffic flows as traffic is being forward to a single end host. In deployments where multiple end hosts are using the same IP address (anycast), *resilient hashing* must be used.

## Resilient Hashing

In Cumulus Linux when a next hop fails or is removed from an ECMP pool, the hashing or hash bucket assignment can change. For deployments where there is a need for flows to always use the same next hop, like TCP anycast deployments, this can create session failures.

The ECMP hash performed with resilient hashing is exactly the same as the default hashing mode. Only the method in which next hops are assigned to hash buckets differs.

Resilient hashing supports both IPv4 and IPv6 routes.

Resilient hashing is not enabled by default. See below for steps on configuring it.



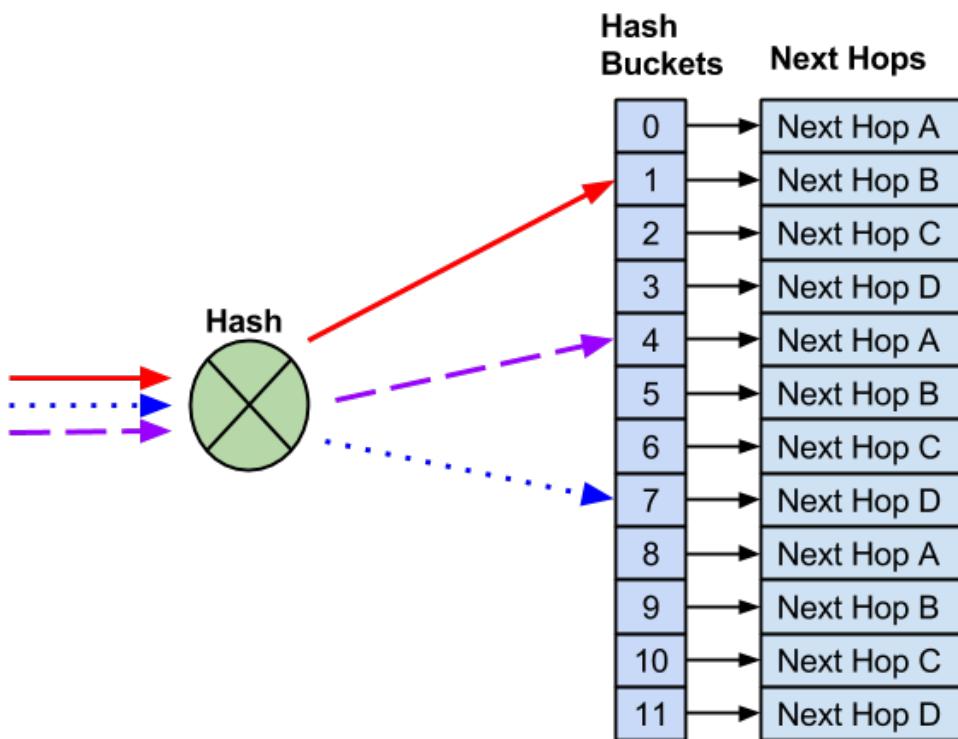
Resilient hashing is not supported on switches with [Spectrum ASICs](#).



Resilient hashing prevents disruptions when new next hops are removed. It does not prevent disruption when next hops are added.

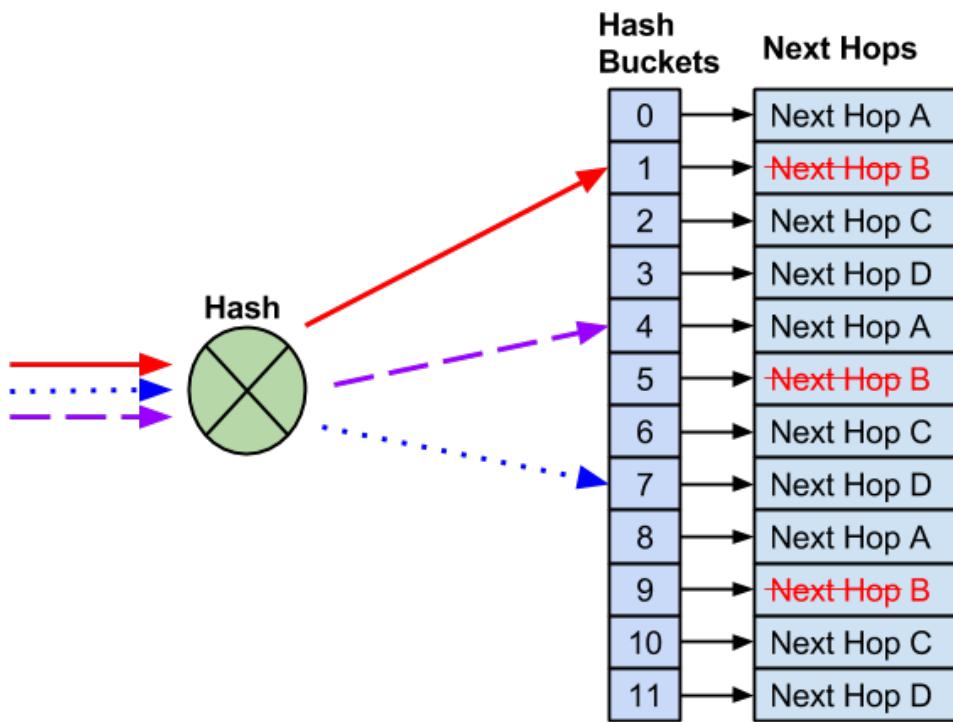
## Resilient Hash Buckets

When resilient hashing is configured, a fixed number of buckets are defined. Next hops are then assigned in round robin fashion to each of those buckets. In this example, 12 buckets are created and four next hops are assigned.

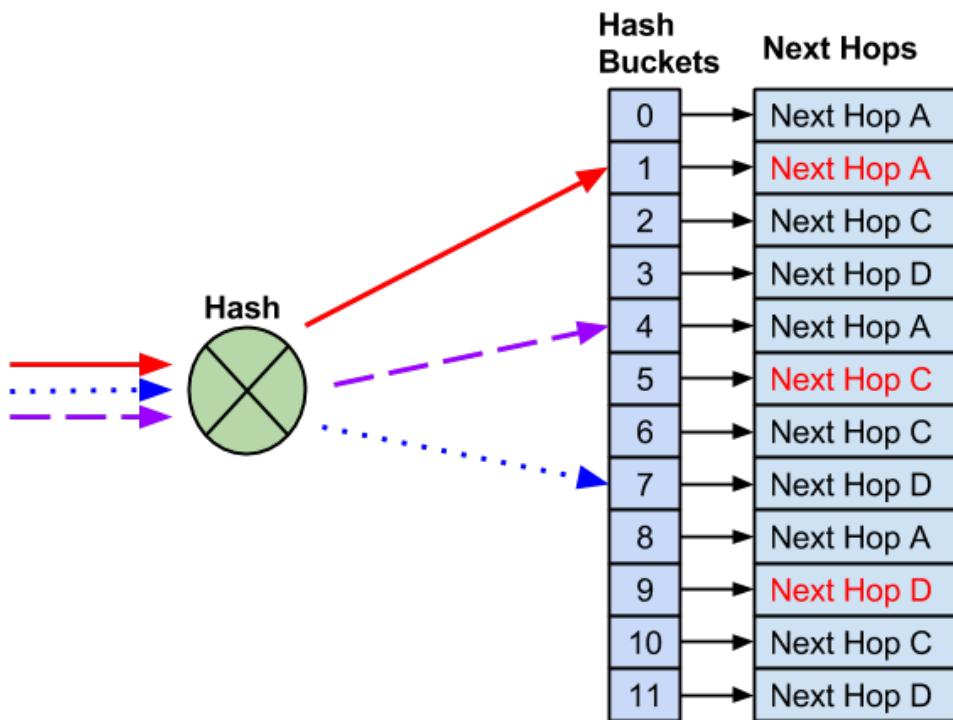


## Removing Next Hops

Unlike default ECMP hashing, when a next hop needs to be removed, the number of hash buckets does not change.



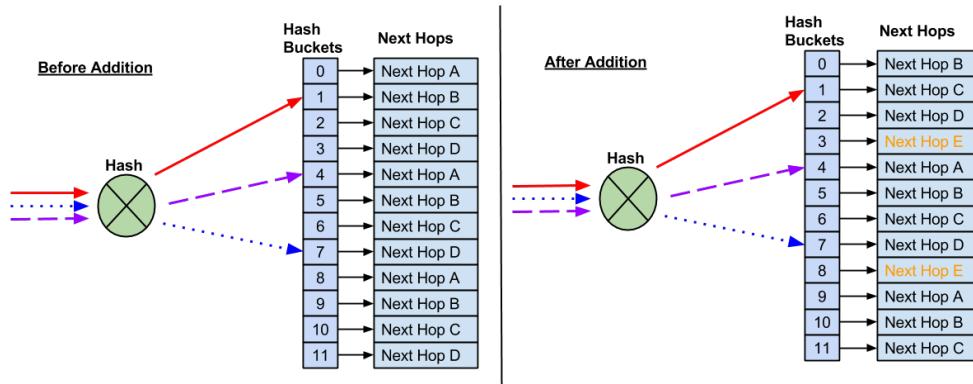
With 12 buckets assigned and four next hops, instead of reducing the number of buckets — which would impact flows to known good hosts — the remaining next hops replace the failed next hop.



After the failed next hop is removed, the remaining next hops are installed as replacements. This prevents impact to any flows that hash to working next hops.

## Adding Next Hops

Resilient hashing does not prevent possible impact to existing flows when new next hops are added. Due to the fact there are a fixed number of buckets, a new next hop requires reassigning next hops to buckets.



As a result, some flows may hash to new next hops, which can impact anycast deployments.

## Configuring Resilient Hashing

Resilient hashing is not enabled by default. When resilient hashing is enabled, 65,536 buckets are created to be shared among all ECMP routes.



An ECMP route counts as a single route with multiple next hops. The following example is considered to be a single ECMP route:

```
$ ip route show 10.1.1.0/24
10.1.1.0/24 proto zebra metric 20
  nexthop via 192.168.1.1 dev swp1 weight 1 onlink
  nexthop via 192.168.2.1 dev swp2 weight 1 onlink
```

All ECMP routes must use the same number of buckets (the number of buckets cannot be configured per ECMP route).

The number of buckets can be configured as 64, 128, 256, 512 or 1024; the default is 128:

Number of Hash Buckets	Number of Supported ECMP Routes
64	1024
<b>128</b>	<b>512</b>

Number of Hash Buckets	Number of Supported ECMP Routes
256	256
512	128
1024	64

A larger number of ECMP buckets reduces the impact on adding new next hops to an ECMP route. However, the system supports fewer ECMP routes. If the maximum number of ECMP routes have been installed, new ECMP routes log an error and are not installed.

To enable resilient hashing, edit `/etc/cumulus/datapath/traffic.conf`:

1. Enable resilient hashing:

```
# Enable resilient hashing
resilient_hash_enable = TRUE
```

2. **(Optional)** Edit the number of hash buckets:

```
# Resilient hashing flowset entries per ECMP group
# Valid values - 64, 128, 256, 512, 1024
resilient_hash_entries_ecmp = 256
```

3. Restart the `switchd` service:

```
cumulus@switch:~$ sudo systemctl restart switchd.service
```

## Caveats

Resilient hashing is only supported on switches with the [Tomahawk, Trident II+ and Trident II chipsets](#). You can run `netshow system` to determine the chipset.

## Useful Links

- [http://en.wikipedia.org/wiki/Equal-cost\\_multi-path\\_routing](http://en.wikipedia.org/wiki/Equal-cost_multi-path_routing)

## Redistribute Neighbor

*Redistribute neighbor* provides a mechanism for IP subnets to span racks without forcing the end hosts to run a routing protocol.

The fundamental premise behind redistribute neighbor is to announce individual host /32 routes in the routed fabric. Other hosts on the fabric can then use this new path to access the hosts in the fabric. If multiple equal-cost paths (ECMP) are available, traffic can load balance across the available paths natively.

The challenge is to accurately compile and update this list of reachable hosts or neighbors. Luckily, existing commonly-deployed protocols are available to solve this problem. Hosts use [ARP](#) to resolve MAC addresses when sending to an IPv4 address. A host then builds an ARP cache table of known MAC addresses: IPv4 tuples as they receive or respond to ARP requests.

In the case of a leaf switch, where the default gateway is deployed for hosts within the rack, the ARP cache table contains a list of all hosts that have ARP'd for their default gateway. In many scenarios, this table contains all the layer 3 information that's needed. This is where redistribute neighbor comes in, as it is a mechanism of formatting and syncing this table into the routing protocol.

## Contents

- Availability (see page 460)
- Target Use Cases and Best Practices (see page 460)
- How It Works (see page 461)
- Configuration Steps (see page 461)
  - Configuring the Leaf(s) (see page 462)
  - Configuring the Host(s) (see page 464)
    - Configuring a Dual-connected Host (see page 464)
    - Installing ifplugged (see page 465)
- Known Limitations (see page 466)
  - TCAM Route Scale (see page 466)
  - Possible Uneven Traffic Distribution (see page 466)
  - Silent Hosts Never Receive Traffic (see page 466)
  - Support for IPv4 Only (see page 466)
  - VRFs Are not Supported (see page 466)
- Troubleshooting (see page 466)
  - Verification (see page 468)

## Availability

Redistribute neighbor is distributed as `python-rdnbrd`.

## Target Use Cases and Best Practices

Redistribute neighbor was created with these use cases in mind:

- Virtualized clusters
- Hosts with service IP addresses that migrate between racks
- Hosts that are dual connected to two leaf nodes without using proprietary protocols such as MLAG ([see page 244](#))
- Anycast services needing dynamic advertisement from multiple hosts

Cumulus Networks recommends following these guidelines with redistribute neighbor:

- Use a single logical connection from each host to each leaf.
- A host can connect to one or more leafs. Each leaf advertises the /32 it sees in its neighbor table.
- A host-bound bridge/VLAN should be local to each switch only.
- Leaf switches with redistribute neighbor enabled should be directly connected to the hosts.
- IP addressing must be non-overlapping, as the host IPs are directly advertised into the routed fabric.
- Run redistribute neighbor on Linux-based hosts primarily; other host operating systems may work, but Cumulus Networks has not actively tested any at this stage.

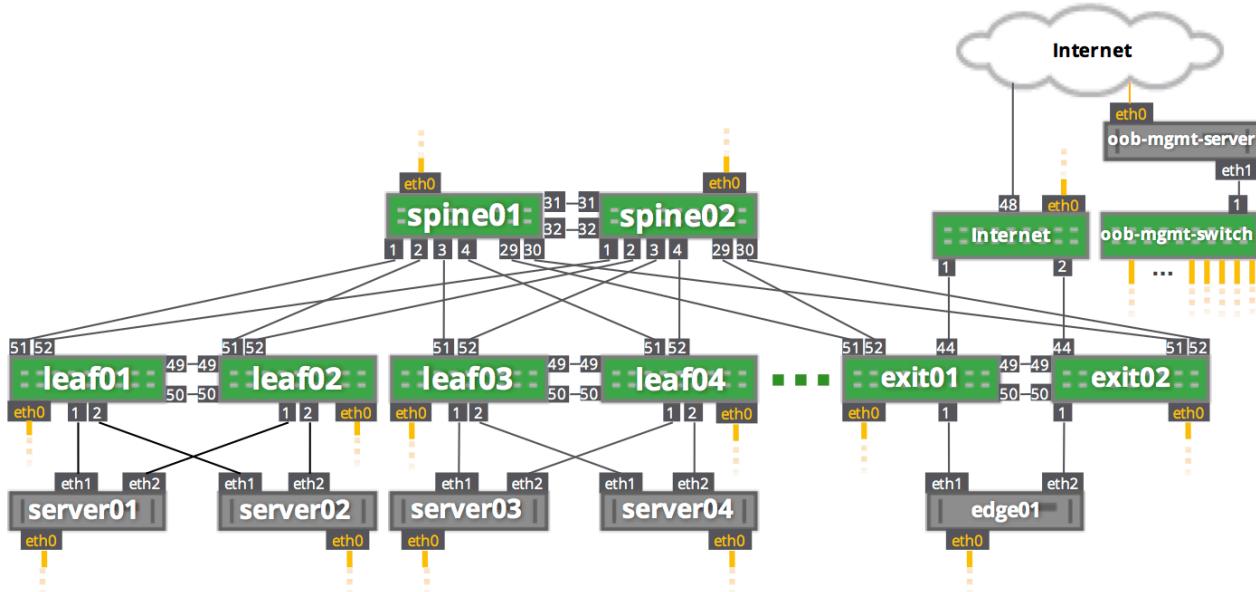
## How It Works

Redistribute neighbor works as follows:

1. The leaf/ToR switches learn about connected hosts when the host sends an ARP request or ARP reply.
2. An entry for the host is added to the kernel neighbor table of each leaf switch.
3. The redistribute neighbor daemon, `rdnbrd`, monitors the kernel neighbor table and creates a /32 route for each neighbor entry. This /32 route is created in kernel table 10.
4. Quagga is configured to import routes from kernel table 10.
5. A route-map is used to control which routes from table 10 are imported.
6. In Quagga these routes are imported as *table* routes.
7. BGP, OSPF and so forth are then configured to redistribute the table 10 routes.

## Configuration Steps

The following configuration steps are based on the [reference topology](#) set forth by Cumulus Networks. Here is a diagram of the topology:



## Configuring the Leaf(s)

The following steps demonstrate how to configure leaf01, but the same steps can be applied to any of the leafs.

1. Configure the host facing ports in `/etc/network/interfaces`. Note that you should configure the same IP address on both host-facing interfaces and you should use a /32. In this case, swp1 and swp2 are configured as they are the ports facing server01 and server02:

```
auto lo
iface lo inet loopback
    address 10.0.0.11/32

auto swp1
iface swp1
    address 10.0.0.11/32

auto swp2
iface swp2
    address 10.0.0.11/32
```

2. Install the `python-rdnbrd` package:

```
cumulus@leaf01:~$ sudo apt-get update
cumulus@leaf01:~$ sudo apt-get install python-rdnbrd
```

3. Enable the daemon so it starts at bootup:

```
cumulus@leaf01:~$ sudo systemctl enable rdnbrd.service
```

4. Start the daemon:

```
cumulus@leaf01:~$ sudo systemctl restart rdnbrd.service
```

5. Configure Quagga:

```
cumulus@leaf01:~$ sudo vtysh
Hello, this is Quagga (version 0.99.23.1+cl3u1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
leaf01# conf t
leaf01(config)#
```

- Add the table as routes into the local routing table:

```
leaf01(config)# ip import-table 10
```

- Define a route-map that matches on the host-facing interface:

```
leaf01(config)# route-map REDIST_NEIGHBOR permit 10
leaf01(config-route-map)# match interface swp1
leaf01(config-route-map)# route-map REDIST_NEIGHBOR permit 20
leaf01(config-route-map)# match interface swp2
```

- Apply that route-map to routes imported into *table*:

```
leaf01(config)# ip protocol table route-map REDIST_NEIGHBOR
```

- Redistribute the imported *table* routes in into the appropriate routing protocol.

#### BGP:

```
leaf01(config)# router bgp 65001
leaf01(config-router)# address-family ipv4 unicast
leaf01(config-router-af)# redistribute table 10
leaf01(config-router-af)# end
```

#### OSPF:

```
leaf01(config)# router ospf
leaf01(config-router)# redistribute table 10
```

Here's the contents of `/etc/quagga/Quagga.conf` from the reference topology:

```
cumulus@leaf01$ cat /etc/network/interfaces
!
interface swp51
    no ipv6 nd suppress-ra
    ipv6 nd ra-interval 3
!
interface swp52
```

```
no ipv6 nd suppress-ra
  ipv6 nd ra-interval 3
!
ip import-table 10 distance 19
route-map rdarp permit 1
match interface swp2
route-map rdarp permit 2
match interface swp1
!
ip protocol table route-map rdarp
router bgp 65011
  bgp router-id 10.0.0.11
  network 10.0.0.11/32
  maximum-paths 64
  bgp bestpath as-path multipath-relax
  bgp bestpath compare-routerid
  neighbor fabric peer-group
  neighbor fabric description Internal Fabric Network
  neighbor fabric capability extended-nexthop
  neighbor fabric advertisement-interval 0
  neighbor fabric timers 1 3
  neighbor fabric timers connect 3
  neighbor fabric remote-as external
  neighbor swp51 interface v6only
  neighbor swp51 peer-group fabric
  neighbor swp52 interface v6only
  neighbor swp52 peer-group fabric
  redistribute table 10
!
address-family ipv6 unicast
  neighbor fabric activate
  neighbor swp51 activate
  neighbor swp52 activate
exit-address-family
!
!
```

## Configuring the Host(s)

There are a few possible host configurations that range in complexity. This document only covers the basic use case: dual-connected Linux hosts with static IP addresses assigned.

Additional host configurations will be covered in future separate knowledge base articles.

## Configuring a Dual-connected Host

Configure a host with the same /32 IP address on its loopback (lo) and uplinks (in this example, eth1 and eth2). This is done so both leaf switches advertise the same /32 regardless of the interface. Cumulus Linux relies on [ECMP \(see page 451\)](#) to load balance across the interfaces southbound, and an equal cost static route (see the configuration below) for load balancing northbound.

The loopback hosts the primary service IP address(es) and to which you can bind services.

Configure the loopback and physical interfaces. Referring back to the topology diagram, server01 is connected to leaf01 via eth1 and to leaf02 via eth2. You should note:

- The loopback IP is assigned to lo, eth1 and eth2.
- The post-up ARPing is used to force the host to ARP as soon as its interface comes up. This allows the leaf to learn about the host as soon as possible.
- The post-up `ip route replace` is used to install a default route via one or both leaf nodes if both swp1 and swp2 are up.

```
cumulus@server01$ cat /etc/network/interfaces
# The loopback network interface
auto lo
iface lo inet loopback
auto lo:1
iface lo:1
    address 10.1.0.101/32
auto eth1
iface eth1
    address 10.1.0.101/32
    post-up for i in {1..3}; do arping -q -c 1 -w 0 -i eth1 10.0.0.11; sleep 1; done
    post-up ip route add 0.0.0.0/0 nexthop via 10.0.0.11 dev eth1 onlink nexthop via 10.0.0.12 dev eth2 onlink || true

auto eth2
iface eth2
    address 10.1.0.101/32
    post-up for i in {1..3}; do arping -q -c 1 -w 0 -i eth2 10.0.0.12; sleep 1; done
    post-up ip route add 0.0.0.0/0 nexthop via 10.0.0.11 dev eth1 onlink nexthop via 10.0.0.12 dev eth2 onlink || true
```

## **Installing ifplugged**

Additionally, install and use `ifplugged`. `ifplugged` modifies the behavior of the Linux routing table when an interface undergoes a link transition (carrier up/down). The Linux kernel by default leaves routes up even when the physical interface is unavailable (NO-CARRIER).

Install `ifplugged` on the host and modify the settings in `/etc/default/ifplugged`:

```
cumulus@server01:~$ sudo apt-get update
cumulus@server01:~$ sudo apt-get install ifplugged
```

Edit `/etc/default/ifplugged` as follows, where `eth1` and `eth2` are the interface names that your host uses to connect to the leaves.

```
cumulus@server01$ cat /etc/default/ifplugged
```

```
INTERFACES="eth1 eth2"
HOTPLUG_INTERFACES=""
ARGS="-q -f -u10 -d10 -w -I"
SUSPEND_ACTION="stop"
```

For full instructions on installing `ifplugd` on Ubuntu, follow [this guide](#).

## **Known Limitations**

### **TCAM Route Scale**

This feature adds each ARP entry as a /32 host route into the routing table of all switches within a summarization domain. Take care to keep the number of hosts minus fabric routes under the TCAM size of the switch. Review the [Cumulus Networks datasheets](#) for up to date scalability limits of your chosen hardware platforms. If in doubt, contact Cumulus Networks support or your Cumulus Networks CSE; they will be happy to help.

### **Possible Uneven Traffic Distribution**

Linux uses source L3 addresses only to do load balancing on most older distributions.

### **Silent Hosts Never Receive Traffic**

Freshly provisioned hosts that have never sent traffic may not ARP for their default gateways. The post-up ARPing in `/etc/network/interfaces` on the host should take care of this. If the host does not ARP, then `rdnbrd` on the leaf cannot learn about the host.

### **Support for IPv4 Only**

This release of redistribute neighbor supports IPv4 only.

### **VRFs Are not Supported**

This release of redistribute neighbor does not support VRFs (see page 469).

## **Troubleshooting**

- How do I determine if `rdnbrd` (the redistribute neighbor daemon) is running?

Use `systemctl` to check:

```
cumulus@leaf01$ systemctl status rdnbrd.service
* rdnbrd.service - Cumulus Linux Redistribute Neighbor Service
  Loaded: loaded (/lib/systemd/system/rdnbrd.service; enabled)
  Active: active (running) since Wed 2016-05-04 18:29:03 UTC; 1h
           13min ago
    Main PID: 1501 (python)
   CGroup: /system.slice/rdnbrd.service
             `--1501 /usr/bin/python /usr/sbin/rdnbrd -d
```

- **How do I change rdnbrd's default configuration?**

By editing /etc/rdnbrd.conf then running systemctl restart rdnbrd.service:

```

cumulus@leaf01$ cat /etc/rdnbrd.conf
# syslog logging level CRITICAL, ERROR, WARNING, INFO, or DEBUG
loglevel = INFO

# TX an ARP request to known hosts every keepalive seconds
keepalive = 1

# If a host does not send an ARP reply for holdtime consider the
host down
holdtime = 3

# Install /32 routes for each host into this table
route_table = 10

# Uncomment to enable ARP debugs on specific interfaces.
# Note that ARP debugs can be very chatty.
# debug_arp = swp1 swp2 swp3 br1
# If we already know the MAC for a host, unicast the ARP
request. This is
# unusual for ARP (why ARP if you know the destination MAC) but
we will be
# using ARP as a keepalive mechanism and do not want to
broadcast so many ARPs
# if we do not have to. If a host cannot handle a unicasted ARP
request, set
# the following option to False.
#
# Unicasting ARP requests is common practice (in some scenarios)
# for other
# networking operating systems so it is unlikely that you will
need to set
# this to False.
unicast_arp_requests = True
cumulus@leaf01:~$ sudo systemctl restart rdnbrd.service

```

- **What is table 10? Why was table 10 chosen?**

The Linux kernel supports multiple routing tables and has the ability to utilize 0 through 255 as table IDs. However, tables 0, 253, 254 and 255 are reserved, and 1 is usually the first one utilized, so rdnbrd only allows you to specify 2-252. The number 10 was chosen for no particular reason. Feel free to set it to any value between 2-252. You can see all the tables specified here:

```

cumulus@switch$ cat /etc/iproute2/rt_tables
#
# reserved values
#
255 local

```

```

254 main
253 default
0 unspec
#
# local
#
#1 inr.ruhep

```

Read more information on [Linux route tables](#), or you can read the [Ubuntu man pages for ip route](#).

- **How do I determine that the /32 redistribute neighbor routes are being advertised to my neighbor?**

For BGP, check the advertised routes to the neighbor.

```

cumulus@leaf01:~$ sudo vtysh
Hello, this is Quagga (version 0.99.23.1+cl3u1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
leaf01# show ip bgp neighbor swp51 advertised-routes
BGP table version is 5, local router ID is 10.0.0.11
Status codes: s suppressed, d damped, h history, * valid, >
best, = multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

      Network          Next Hop            Metric LocPrf Weight Path
*-> 10.0.0.11/32    0.0.0.0                  0        32768 i
*-> 10.0.0.12/32    ::                      0       6502
0 65012 i
*-> 10.0.0.21/32    ::                      0       6502
0 i
*-> 10.0.0.22/32    ::                      0       6502
0 i

Total number of prefixes 4

```

## Verification

The following workflow can be used to verify that the kernel routing table is being correctly populated, and that routes are being correctly imported/advertised:

1. Verify that ARP neighbour entries are being populated into the Kernel routing table 10.

```

cumulus@switch:~$ ip route show table 10
10.0.1.101 dev swp1 scope link

```

If these routes are not being generated, verify the following:

- That the `rdnbrd` daemon is running

- Check `/etc/rdnbrd.conf` to verify the correct table number is used
2. Verify that routes are being imported into Quagga from the kernel routing table 10.

```
cumulus@switch:~$ sudo vtysh
Hello, this is Quagga (version 0.99.23.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
switch# show ip route table
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel, T - Table,
       > - selected route, * - FIB route
T[10]>* 10.0.1.101/32 [19/0] is directly connected, swp1, 01:25:29
```

Both the `>` and `*` should be present so that table 10 routes are installed as preferred into the routing table. If the routes are not being installed, verify the following:

- The imported distance of the locally imported kernel routes using the `ip import 10 distance X` command, where X is **not** less than the administrative distance of the routing protocol. If the distance is too low, routes learned from the protocol may overwrite the locally imported routes.
  - The routes are in the kernel routing table.
3. Confirm that routes are in the BGP/OSPF database and being advertised.

```
switch# show ip bgp
```

## Virtual Routing and Forwarding - VRF

Cumulus Linux provides *virtual routing and forwarding* (VRF) to allow for the presence of multiple independent routing tables working simultaneously on the same router or switch. This permits multiple network paths without the need for multiple switches. Think of this feature as VLAN for layer 3, but unlike VLANs, there is no field in the IP header carrying it. Other implementations call this feature *VRF-Lite*.

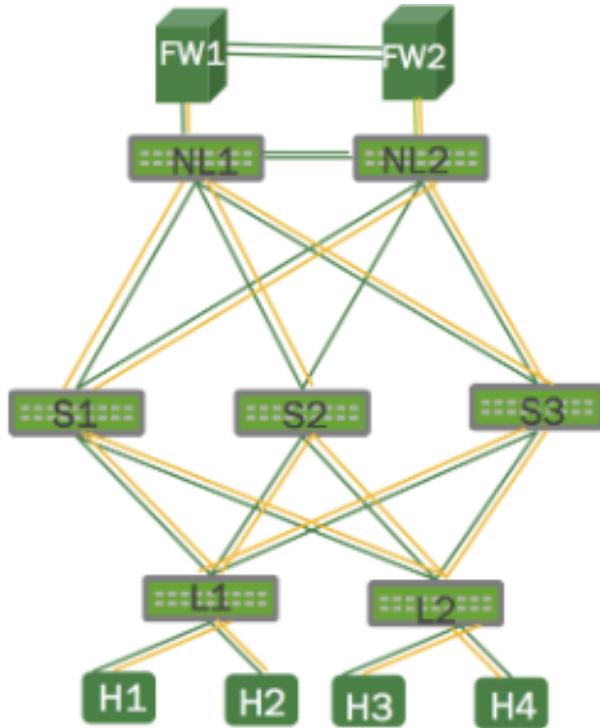
The primary use cases for VRF in a data center are similar to VLANs at layer 2: using common physical infrastructure to carry multiple isolated traffic streams for multi-tenant environments, where these streams are allowed to cross over only at configured boundary points, typically firewalls or IDS. You can also use it to burst traffic from private clouds to enterprise networks where the burst point is at layer 3. Or you can use it in an OpenStack deployment.

VRF is fully supported in the Linux kernel, so it has the following characteristics:

- The VRF is presented as a layer 3 master network device with its own associated routing table.
- The layer 3 interfaces (VLAN interfaces, bonds, switch virtual interfaces/SVIs) associated with the VRF are enslaved to that VRF; IP rules direct FIB (forwarding information base) lookups to the routing table for the VRF device.
- The VRF device can have its own IP address, known as a *VRF-local loopback*.
- Applications can use existing interfaces to operate in a VRF context — by binding sockets to the VRF device or passing the `ifindex` using `cmsg`.

- Listen sockets used by services are VRF-global by default unless the application is configured to use a more limited scope. Connected sockets (like TCP) are then bound to the VRF domain in which the connection originates.
- Connected and local routes are placed in appropriate VRF tables.
- Neighbor entries continue to be per-interface, and you can view all entries associated with the VRF device.
- A VRF does not map to its own network namespace; however, you can nest VRFs in a network namespace.
- You can use existing Linux tools to interact with it, such as `tcpdump`.

You configure VRF by associating each subset of interfaces to a VRF routing table, and configuring an instance of the routing protocol — BGP — for each routing table.



VRF is not supported on switches with [Spectrum ASICs](#).

## Contents

- [Contents \(see page 470\)](#)
- [Configuring VRF \(see page 471\)](#)
  - [Specifying a Table ID \(see page 471\)](#)
  - [Using the vrf Command \(see page 472\)](#)
- [Quagga Operation in a VRF \(see page 474\)](#)
  - [Example Configuration \(see page 475\)](#)
- [Example Quagga Commands \(see page 475\)](#)
  - [Showing VRF Data with vtysh \(see page 476\)](#)

- Showing VRF Data Using ip Commands (see page 480)
- Using BGP Unnumbered Interfaces with VRF (see page 483)
- Using DHCP with VRF (see page 484)
  - Caveats for DHCP with VRF (see page 485)
  - Example Configuration (see page 485)
- Caveats (see page 488)

## Configuring VRF

Each routing table is called a *VRF table*, and has its own table ID. You configure VRF using `ifupdown2` or the `vrf` command.

You create the VRF in `/etc/network/interfaces`, then place the layer 3 interface in the VRF. You can have a maximum of 64 VRFs on a switch.

When you configure VRF with `ifupdown2`, you follow a similar process to other network interfaces. Keep in the mind the following for a VRF table:

- It can have an IP address, a loopback interface for the VRF
- Associated rules are added automatically
- You can also add a default route to avoid skipping across tables when the kernel forwards the packet
- Names for VRF tables can be up to 15 characters. However, you **cannot** use the name `mgmt`, as this name can **only** be used for **management VRF** (see page 489).

Here is a sample VRF configuration in `/etc/network/interfaces`:

```
auto <vrf-name>
iface <vrf-name>
  vrf-table auto

auto swp#
iface swp#
  address <IP address>
  down ip addr flush dev swp#
  ...
vrf <vrf-name>
```

## Specifying a Table ID

Instead of having Cumulus Linux assign a table ID for the VRF table, you can specify your own table ID in the configuration. The table ID to name mapping is saved in `/etc/iproute2/rt_tables.d/` for name-based references. So instead of using the `auto` option above, specify the table ID like this:

```
auto <vrf-name>
iface <vrf-name>
    vrf-table <table-id>
```

Note that if you do specify a table ID, it **must** be in the range of 1001 to 1255.

## Using the vrf Command

The `vrf` command returns information about VRF tables. You can also use it to execute non-VRF-specific commands and perform other tasks related to VRF tables.

To get a list of VRF tables, run:

```
cumulus@switch:~$ vrf list
```

VRF	Table
red	1010
blue	1020
mgmt	1001

To see a list of links associated with a particular VRF table, run `vrf link list <vrf-name>`. For example:

```
cumulus@switch:~$ vrf link list red
```

VRF: red		
swp1.10@swp1	UP	6c:64:1a:00:5a:0c <BROADCAST,MULTICAST,UP,
LOWER_UP>		
swp2.10@swp2	UP	6c:64:1a:00:5a:0d <BROADCAST,MULTICAST,UP,
LOWER_UP>		

To see a list of routes associated with a particular VRF table, run `vrf route list <vrf-name>`. For example:

```
cumulus@switch:~$ vrf route list red
```

VRF: red		
unreachable	default	metric 8192

```

10.1.1.0/24 via 10.10.1.2 dev swp2.10
10.1.2.0/24 via 10.99.1.2 dev swp1.10
broadcast 10.10.1.0 dev swp2.10 proto kernel scope link src 10.10.1.1
10.10.1.0/28 dev swp2.10 proto kernel scope link src 10.10.1.1
local 10.10.1.1 dev swp2.10 proto kernel scope host src 10.10.1.1
broadcast 10.10.1.15 dev swp2.10 proto kernel scope link src 10.10.1.1
broadcast 10.99.1.0 dev swp1.10 proto kernel scope link src 10.99.1.1
10.99.1.0/30 dev swp1.10 proto kernel scope link src 10.99.1.1
local 10.99.1.1 dev swp1.10 proto kernel scope host src 10.99.1.1
broadcast 10.99.1.3 dev swp1.10 proto kernel scope link src 10.99.1.1

local fe80:: dev lo proto none metric 0 pref medium
local fe80:: dev lo proto none metric 0 pref medium
local fe80::6e64:1aff:fe00:5a0c dev lo proto none metric 0 pref medium
local fe80::6e64:1aff:fe00:5a0d dev lo proto none metric 0 pref medium
fe80::/64 dev swp1.10 proto kernel metric 256 pref medium
fe80::/64 dev swp2.10 proto kernel metric 256 pref medium
ff00::/8 dev swp1.10 metric 256 pref medium
ff00::/8 dev swp2.10 metric 256 pref medium
unreachable default dev lo metric 8192 error -101 pref medium

```

You can execute non-VRF-specific Linux commands and perform other tasks against a given VRF table. They affect only AF\_INET and AF\_INET6 sockets opened by the command that gets executed; it has no impact on netlink sockets, associated with the `ip` command. To execute such a command against a VRF table, run `vrf task exec <vrf-name> <command>`. For example:

```

cumulus@switch:~$ sudo vrf task exec red cl-resource-query
IPv4/IPv6 host entries:      4,   0% of maximum value  8192
IPv4 neighbors:              4
IPv6 neighbors:              0
IPv4 route entries:          23,   0% of maximum value 32668
IPv6 route entries:          20,   0% of maximum value 16384
IPv4 Routes:                 23
IPv6 Routes:                 20
Total Routes:                43,   0% of maximum value 32768
ECMP nexthops:               0,   0% of maximum value 16327
MAC entries:                  0,   0% of maximum value 32768
Ingress ACL entries:         61,   4% of maximum value 1280
Ingress ACL counters:        82,   6% of maximum value 1280
Ingress ACL meters:          21,   0% of maximum value 4096
Ingress ACL slices:          2,   50% of maximum value     4
Egress ACL entries:          25,   9% of maximum value 256

```

Egress ACL counters:	50, 4% of maximum value	1024
Egress ACL meters:	25, 9% of maximum value	256
Egress ACL slices:	1, 100% of maximum value	1

To return a list of processes and PIDs associated with a specific VRF table, run `vrf task list <vrf-name>`. For example:

```
cumulus@switch:~$ vrf task list red

VRF: red
-----
dhclient      2508
sshd          2659
bash           2681
su             2702
bash           2720
vrf            2829
```

To determine which VRF table is associated with a particular PID, run `vrf task identify <pid>`. For example:

```
cumulus@switch:~$ vrf task identify 2829

red
```

## Quagga Operation in a VRF

In Cumulus Linux 3.0, BGP and static routing (IPv4 and IPv6) are supported within a VRF context. Various Quagga routing constructs, such as routing tables, nexthops, router-id, and related processing are also VRF-aware. However, OSPFv2 and OSPFv3 are not VRF-aware and can only operate in the default VRF.

Quagga (see page 391) learns of VRFs provisioned on the system as well as interface attachment to a VRF through notifications from the kernel.

You can assign switch ports to each VRF table with an interface-level configuration, and BGP instances can be assigned to the table with a BGP router-level command. Note that OSPFv2 and OSPFv3 are **not** VRF-aware.

Because BGP is VRF-aware, it supports per-VRF neighbors, both iBGP and eBGP as well as numbered and unnumbered interfaces. Non-interface-based VRF neighbors are bound to the VRF, which is how you can have overlapping address spaces in different VRFs. Each VRF can have its own parameters, such as address families and redistribution. Incoming connections rely on the Linux kernel for VRF-global sockets. BGP neighbors can be tracked using BFD (see page 446), both for single and multiple hops. You can configure multiple BGP (see page 419) instances, associating each with a VRF.

As mentioned above, VRFs are provisioned through `/etc/network/interfaces`. VRFs can be pre-provisioned in Quagga too, but they become active only when configured through `/etc/network/interfaces`.

- A VRF can be pre-provisioned in Quagga by running the command `vrf vrf-name`.
- A BGP instance corresponding to a VRF can be pre-provisioned by configuring `router bgp asn vrf vrf-name`. Under this context, all existing BGP parameters can be configured - neighbors, peer-groups, address-family configuration, redistribution etc.
- Static routes (IPv4 and IPv6) can be provisioned in a VRF by specifying the VRF along with the static route configuration. For example, `ip route prefix nexthop vrf vrf-name`. The VRF has to exist for this configuration to be accepted - either already defined through `/etc/network/interfaces` or pre-provisioned in Quagga.

## **Example Configuration**

Here's an example VRF configuration in BGP:

```

router bgp 64900 vrf vrf1012
  bgp router-id 6.0.2.7
  no bgp default ipv4-unicast
  neighbor ISL peer-group
  neighbor ISLv6 peer-group
  neighbor swp1.2 interface v6only peer-group ISLv6
  neighbor swp1.2 remote-as 65000
  neighbor swp3.2 interface v6only peer-group ISLv6
  neighbor swp3.2 remote-as 65000
  neighbor 169.254.2.18 remote-as 65000
  neighbor 169.254.2.18 peer-group ISL
!
address-family ipv4 unicast
  network 20.7.2.0/24
  neighbor ISL activate
  neighbor ISL route-map ALLOW_BR2 out
exit-address-family
!
address-family ipv6 unicast
  network 2003:7:2::/125
  neighbor ISLv6 activate
  neighbor ISLv6 route-map ALLOW_BR2_v6 out
exit-address-family
!
```

## **Example Quagga Commands**

You can view VRF data either directly in Cumulus Linux with `ip` commands or with the `vtysh` shell.

## Showing VRF Data with vtysh

You run the following Quagga commands are run in the **vtysh** terminal.

Show all VRFs learned by Quagga from the kernel. The table ID shows the corresponding routing table in the kernel either automatically assigned or manually defined in `/etc/network/interfaces`:

```
switch# show vrf
vrf vrf1012 id 14 table 1012
vrf vrf1013 id 21 table 1013
vrf vrf1014 id 28 table 1014
```

Show VRFs configured in BGP, including the default. A non-zero ID is a VRF that has also been actually provisioned — that is, defined in `/etc/network/interfaces`:

```
switch# show bgp vrfs
Type   Id      RouterId          #PeersCfg  #PeersEstb  Name
DFLT   0       6.0.0.7           0          0  Default
VRF    14      6.0.2.7           6          6  vrf1012
VRF    21      6.0.3.7           6          6  vrf1013
VRF    28      6.0.4.7           6          6  vrf1014

Total number of VRFs (including default): 4
```

Display interfaces known to Quagga and attached to this VRF:

```
switch# show interface vrf vrf1012
Interface br2 is up, line protocol is down
  PTM status: disabled
  vrf: vrf1012
  index 13 metric 0 mtu 1500
  flags: <UP,BROADCAST,MULTICAST>
  inet 20.7.2.1/24

  inet6 fe80::202:ff:fe00:a/64
    ND advertised reachable time is 0 milliseconds
    ND advertised retransmit interval is 0 milliseconds
    ND router advertisements are sent every 600 seconds
    ND router advertisements lifetime tracks ra-interval
    ND router advertisement default router preference is medium
    Hosts use stateless autoconfig for addresses.
```

To see more interfaces, click [here](#).

```
Interface swp1.2 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 8 metric 0 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:07
  inet 169.254.2.9/30

    inet6 fe80::202:ff:fe00:7/64
      ND advertised reachable time is 0 milliseconds
      ND advertised retransmit interval is 0 milliseconds
      ND router advertisements are sent every 1 seconds
      ND router advertisements lifetime tracks ra-interval
      ND router advertisement default router preference is medium
      Hosts use stateless autoconfig for addresses.

    Neighbor address(s):
      inet6 fe80::202:ff:fe00:1b/128

Interface swp2.2 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 9 metric 0 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:08
  inet 169.254.2.13/30

    inet6 fe80::202:ff:fe00:8/64
      ND advertised reachable time is 0 milliseconds
      ND advertised retransmit interval is 0 milliseconds
      ND router advertisements are sent every 1 seconds
      ND router advertisements lifetime tracks ra-interval
      ND router advertisement default router preference is medium
      Hosts use stateless autoconfig for addresses.

    Neighbor address(s):
      inet6 fe80::202:ff:fe00:1f/128

Interface swp3.2 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 10 metric 0 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:09
  inet 169.254.2.17/30
```

```
inet6 fe80::202:ff:fe00:9/64
ND advertised reachable time is 0 milliseconds
ND advertised retransmit interval is 0 milliseconds
ND router advertisements are sent every 1 seconds
ND router advertisements lifetime tracks ra-interval
ND router advertisement default router preference is medium
Hosts use stateless autoconfig for addresses.

Neighbor address(s):
inet6 fe80::202:ff:fe00:23/128

Interface swp4.2 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 11 metric 0 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:0a

Interface swp5.2 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 12 metric 0 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:02:00:00:00:0b

Interface vrf1012 is up, line protocol is up
  PTM status: disabled
  vrf: vrf1012
  index 14 metric 0 mtu 1500
  flags: <UP,RUNNING,NOARP>
  HWaddr: 46:96:c7:64:4d:fa
```

To show the routes in the VRF:

```
switch# show ip route vrf vrf1012
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, T - Table,
       > - selected route, * - FIB route

C>* 169.254.2.8/30 is directly connected, swp1.2
C>* 169.254.2.12/30 is directly connected, swp2.2
C>* 169.254.2.16/30 is directly connected, swp3.2
```

To show the BGP summary for the VRF:

```

switch# show ip bgp vrf vrf1012 summary
BGP router identifier 6.0.2.7, local AS number 64900 vrf-id 14
BGP table version 0
RIB entries 1, using 120 bytes of memory
Peers 6, using 97 KiB of memory
Peer groups 2, using 112 bytes of memory

Neighbor          V     AS MsgRcvd MsgSent      TblVer  InQ OutQ Up/Down  State
/PfxRcd
s3(169.254.2.18)
                  4 65000  102039  102040          0       0     0 3d13h03m      0
s1(169.254.2.10)
                  4 65000  102039  102040          0       0     0 3d13h03m      0
s2(169.254.2.14)
                  4 65000  102039  102040          0       0     0 3d13h03m      0

Total number of neighbors 3

```

To show BGP (IPv4) routes in the VRF:

```

switch# show ip bgp vrf vrf1012
BGP table version is 0, local router ID is 6.0.2.7
Status codes: s suppressed, d damped, h history, * valid, > best, =
multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

Network          Next Hop            Metric LocPrf Weight Path
20.7.2.0/24      0.0.0.0           0        32768 i

Total number of prefixes 1

```

BGP IPv6 routes in the VRF:

```

switch# show bgp vrf vrf1012
BGP table version is 0, local router ID is 6.0.2.7
Status codes: s suppressed, d damped, h history, * valid, > best, =
multipath,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

```

```

Network          Next Hop          Metric LocPrf Weight Path
2003:7:2::/125 ::                      0        32768 i

Total number of prefixes 1
switch#
switch#
switch# exit
(jessie-switch-amd64)root@switch:/home/cumulus$
```

## Showing VRF Data Using ip Commands

To list all VRFs provisioned, showing the VRF ID (14, 21, 28 below) as well as the table ID:

```

cumulus@switch:~$ ip -d link show type vrf
14: vrf1012: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 46:96:c7:64:4d:fa brd ff:ff:ff:ff:ff:ff promiscuity 0
    vrf table 1012 addrgenmode eui64
21: vrf1013: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 7a:8a:29:0f:5e:52 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vrf table 1013 addrgenmode eui64
28: vrf1014: <NOARP,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether e6:8c:4d:fc:eb:b1 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vrf table 1014 addrgenmode eui64
```

To list the interfaces attached to a specific VRF:

```

cumulus@switch:~$ ip -d link show master vrf1012
8: swp1.2@swp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master vrf1012 state UP mode DEFAULT group default
    link/ether 00:02:00:00:00:07 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 2 <REORDER_HDR>
    vrf_slave addrgenmode eui64
9: swp2.2@swp2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master vrf1012 state UP mode DEFAULT group default
    link/ether 00:02:00:00:00:08 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 2 <REORDER_HDR>
    vrf_slave addrgenmode eui64
10: swp3.2@swp3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
```

```

master vrf1012 state UP mode DEFAULT group default
    link/ether 00:02:00:00:00:09 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 2 <REORDER_HDR>
    vrf_slave addrgenmode eui64
11: swp4.2@swp4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master vrf1012 state UP mode DEFAULT group default
    link/ether 00:02:00:00:00:0a brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 2 <REORDER_HDR>
    vrf_slave addrgenmode eui64
12: swp5.2@swp5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master vrf1012 state UP mode DEFAULT group default
    link/ether 00:02:00:00:00:0b brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 2 <REORDER_HDR>
    vrf_slave addrgenmode eui64
13: br2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue master
vrf1012 state DOWN mode DEFAULT group default
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0
    bridge forward_delay 100 hello_time 200 max_age 2000 ageing_time 30000
    stp_state 0 priority 32768
        vlan_filtering 0 vlan_protocol 802.1Q bridge_id 8000.0:0:0:0:0:0
    designated_root 8000.0:0:0:0:0:0
        root_port 0 root_path_cost 0 topology_change 0 topology_change_detected
0 hello_timer      0.00
        tcn_timer      0.00 topology_change_timer      0.00 gc_timer  202.23
    vlan_default_pvid 1 group_fwd_mask 0
        group_address 01:80:c2:00:00:00 mcast_snooping 1 mcast_router 1
    mcast_query_use_ifaddr 0 mcast_querier 0
        mcast_hash_elasticity 4096 mcast_hash_max 4096 mcast_last_member_count
2 mcast_startup_query_count 2
        mcast_last_member_interval 100 mcast_membership_interval 26000
    mcast_querier_interval 25500
        mcast_query_interval 12500 mcast_query_response_interval 1000
    mcast_startup_query_interval 3125
        nf_call_iptables 0 nf_call_ip6tables 0 nf_call_arptables 0
        vrf_slave addrgenmode eui64
cumulus@switch:~$
```

To show IPv4 routes in a VRF:

```

cumulus@switch:~$ ip route show table vrf1012
unreachable default metric 240
broadcast 20.7.2.0 dev br2 proto kernel scope link src 20.7.2.1 dead
linkdown
```

```
20.7.2.0/24 dev br2 proto kernel scope link src 20.7.2.1 dead linkdown
local 20.7.2.1 dev br2 proto kernel scope host src 20.7.2.1
broadcast 20.7.2.255 dev br2 proto kernel scope link src 20.7.2.1 dead
linkdown
broadcast 169.254.2.8 dev swp1.2 proto kernel scope link src 169.254.2.9
169.254.2.8/30 dev swp1.2 proto kernel scope link src 169.254.2.9
local 169.254.2.9 dev swp1.2 proto kernel scope host src 169.254.2.9
broadcast 169.254.2.11 dev swp1.2 proto kernel scope link src
169.254.2.9
broadcast 169.254.2.12 dev swp2.2 proto kernel scope link src
169.254.2.13
169.254.2.12/30 dev swp2.2 proto kernel scope link src 169.254.2.13
local 169.254.2.13 dev swp2.2 proto kernel scope host src 169.254.2.13
broadcast 169.254.2.15 dev swp2.2 proto kernel scope link src
169.254.2.13
broadcast 169.254.2.16 dev swp3.2 proto kernel scope link src
169.254.2.17
169.254.2.16/30 dev swp3.2 proto kernel scope link src 169.254.2.17
local 169.254.2.17 dev swp3.2 proto kernel scope host src 169.254.2.17
broadcast 169.254.2.19 dev swp3.2 proto kernel scope link src
169.254.2.17
cumulus@switch:~$
```

To show IPv6 routes in a VRF:

```
cumulus@switch:~$ ip -6 route show table vrf1012
local fe80:: dev lo proto none metric 0 pref medium
local fe80:: dev lo proto none metric 0 pref medium
local fe80:: dev lo proto none metric 0 pref medium
local fe80:: dev lo proto none metric 0 pref medium
local fe80::202:ff:fe00:7 dev lo proto none metric 0 pref medium
local fe80::202:ff:fe00:8 dev lo proto none metric 0 pref medium
local fe80::202:ff:fe00:9 dev lo proto none metric 0 pref medium
local fe80::202:ff:fe00:a dev lo proto none metric 0 pref medium
fe80::/64 dev br2 proto kernel metric 256 dead linkdown pref medium
fe80::/64 dev swp1.2 proto kernel metric 256 pref medium
fe80::/64 dev swp2.2 proto kernel metric 256 pref medium
fe80::/64 dev swp3.2 proto kernel metric 256 pref medium
ff00::/8 dev br2 metric 256 dead linkdown pref medium
ff00::/8 dev swp1.2 metric 256 pref medium
ff00::/8 dev swp2.2 metric 256 pref medium
```

```
ff00::/8 dev swp3.2 metric 256 pref medium
unreachable default dev lo metric 240 error -101 pref medium
cumulus@switch:~$
```

## **Using BGP Unnumbered Interfaces with VRF**

BGP unnumbered interface configurations (see page 419) are supported with VRF. In BGP unnumbered, there are no addresses on any interface. However, debugging tools like **traceroute** need at least a single IP address per node as the node's source IP address. Typically, this address was assigned to the loopback device. With VRF, you need a loopback device for each VRF table since VRF is based on interfaces, not IP addresses. While Linux does not support multiple loopback devices, it does support the concept of a dummy interface, which is used to achieve the same goal.

An IP address can be associated with the VRF device, which will then act as the dummy (loopback-like) interface for that VRF.

1. Open the `/etc/network/interfaces` file in a text editor.
2. Configure the BGP unnumbered configuration and save the file. The BGP unnumbered configuration is the same for a non-VRF, applied under the VRF context (`router bgp asn vrf vrf-name`).

```
auto vrf1
iface vrf1
    vrf-table auto
    address 6.1.0.6/32
    address 2001:6:1::6/128

auto swp1.101
iface swp1.101
    link-speed 10000
    link-duplex full
    link-autoneg off
    vrf vrf1

auto br101
iface br101
    address 20.1.6.1/24
    address 2001:20:1:6::1/80
    down ip addr flush dev br101
    bridge-ports swp3.101 swp4.101
    vrf vrf1
```

Quagga BGP configuration:

```
!
router bgp 65001 vrf vrf1
```

```

no bgp default ipv4-unicast
bgp bestpath as-path multipath-relax
bgp bestpath compare-routerid
neighbor LEAF peer-group
neighbor LEAF remote-as external
neighbor LEAF capability extended-nexthop
neighbor swp1.101 interface peer-group LEAF
neighbor swp2.101 interface peer-group LEAF
!
address-family ipv4 unicast
  redistribute connected
  neighbor LEAF activate
  maximum-paths 4
exit-address-family
!
address-family ipv6 unicast
  redistribute connected
  neighbor LEAF activate
  maximum-paths 4
exit-address-family
!
```

## Using DHCP with VRF

Since you can use VRF to bind IPv4 and IPv6 sockets to non-default VRF tables, you have the ability to start DHCP servers and relays in any non-default VRF table using the `dhcpd` and `dhcrelay` services, respectively. These services must be managed by `systemd` in order to run in a VRF context; in addition, the services must be listed in `/etc/vrf/systemd.conf`. By default, this file already lists these two services, as well as others like `ntp` and `snmpd`. You can add more services as needed, such as `dhcpd6` and `dhcrelay6` for IPv6.

If you edit `/etc/vrf/systemd.conf`, run `sudo systemctl daemon-reload` to generate the `systemd` instance files for the newly added service(s). Then you can start the service in the VRF using `systemctl start <service>@<vrf-name>.service`, where `<service>` is the name of the service — such as `dhcpd` or `dhcrelay` — and `<vrf-name>` is the name of the VRF.

For example, to start the `dhcrelay` service after you configured a VRF named `blue`, run:

```
cumulus@switch:~$ sudo systemctl start dhcrelay@blue.service
```

To enable the service at boot time you should also run `systemctl enable <service>@<vrf-name>`. To continue with the previous example:

```
cumulus@switch:~$ sudo systemctl enable dhcrelay@blue.service
```

In addition, you need to create a separate default file in `/etc/default` for every instance of a DHCP server and/or relay in a non-default VRF; this is where you set the server and relay options. To run multiple instances of any of these services, you need a separate file for each instance. The files must be named as follows:

- `isc-dhcp-server-<vrf-name>`
- `isc-dhcp-server6-<vrf-name>`
- `isc-dhcp-relay-<vrf-name>`
- `isc-dhcp-relay6-<vrf-name>`

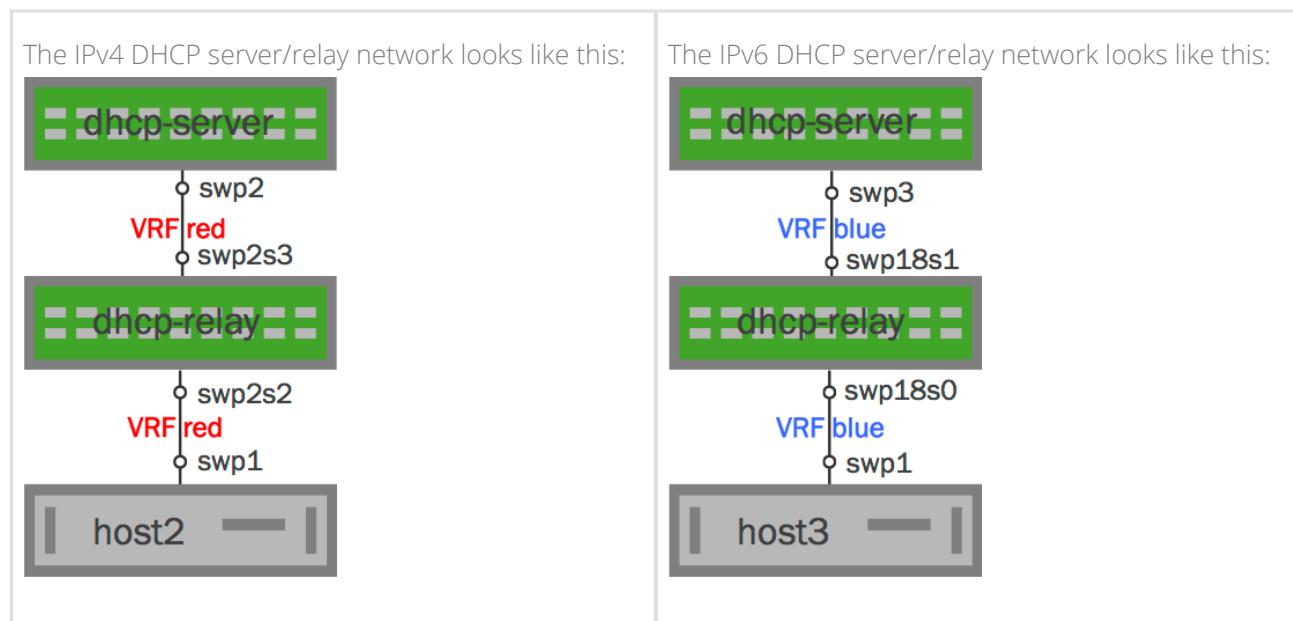
See the example configuration below for more details.

### **Caveats for DHCP with VRF**

- Cumulus Linux does **not** support DHCP server and relay across VRFs, so the server and host cannot be in different VRF tables. In addition, the server and relay cannot be in different VRF tables.
- Typically a service running in the default VRF owns a port across all VRFs. If the VRF local instance is preferred, the global one may need to be disabled and stopped first.
- VRF is a layer 3 routing feature. It only makes sense to run programs that use AF\_INET and AF\_INET6 sockets in a VRF. VRF context does not affect any other aspects of the operation of a program.
- This method only works with `systemd`-based services.

### **Example Configuration**

In the following example, there is one IPv4 network with a VRF named *red* and one IPv6 network with a VRF named *blue*.



Configure each DHCP server and relay as follows:

Sample DHCP Server Configuration	Sample DHCP6 Server Configuration
<ol style="list-style-type: none"> <li>1. Create the file <code>isc-dhcp-server-red</code> in <code>/etc/default/</code>. Here is sample content:</li> </ol>	<ol style="list-style-type: none"> <li>1. Create the file <code>isc-dhcp-server6-blue</code> in <code>/etc/default/</code>. Here is sample content:</li> </ol>

```

# Defaults for isc-dhcp-
server initscript
# sourced by /etc/init.d
/etc/dhcp-server
# installed at /etc/default
/etc/dhcp-server by the
maintainer scripts
#
# This is a POSIX shell
fragment
#
# Path to dhcpcd's config
file (default: /etc/dhcp
/dhcpcd.conf).
DHCPD_CONF="-cf /etc/dhcp
/dhcpcd-red.conf"
# Path to dhcpcd's PID file
(default: /var/run/dhcpcd.
pid).
DHCPD_PID="-pf /var/run
/dhcpcd-red.pid"
# Additional options to
start dhcpcd with.
# Don't use options -cf or
-pf here; use DHCPD_CONF/
DHCPD_PID instead
#OPTIONS=""
# On what interfaces
should the DHCP server
(dhcpcd) serve DHCP
requests?
# Separate multiple
interfaces with spaces, e.
g. "eth0 eth1".
INTERFACES="swp2"

```

2. Enable the DHCP server:  
`cumulus@switch:~$ sudo systemctl enable dhcpd@red.service`
3. Start the DHCP server:  
`cumulus@switch:~$ sudo systemctl start dhcpd@red.service`  
 or  
`cumulus@switch:~$ sudo systemctl restart dhcpd@red.service`
4. Check status:  
`cumulus@switch:~$ sudo systemctl status dhcpd@red.service`

```

# Defaults for isc-dhcp-
server initscript
# sourced by /etc/init.d
/etc/dhcp-server
# installed at /etc/default
/etc/dhcp-server by the
maintainer scripts
#
# This is a POSIX shell
fragment
#
# Path to dhcpcd's config
file (default: /etc/dhcp
/dhcpcd.conf).
DHCPD_CONF="-cf /etc/dhcp
/dhcpcd6-blue.conf"
# Path to dhcpcd's PID file
(default: /var/run/dhcpcd.
pid).
DHCPD_PID="-pf /var/run
/dhcpcd6-blue.pid"
# Additional options to
start dhcpcd with.
# Don't use options -cf or
-pf here; use DHCPD_CONF/
DHCPD_PID instead
#OPTIONS=""
# On what interfaces
should the DHCP server
(dhcpcd) serve DHCP
requests?
# Separate multiple
interfaces with spaces, e.
g. "eth0 eth1".
INTERFACES="swp3"

```

2. Enable the DHCP server:  
`cumulus@switch:~$ sudo systemctl enable dhcpd6@blue.service`
3. Start the DHCP server:  
`cumulus@switch:~$ sudo systemctl start dhcpd6@blue.service`  
 or  
`cumulus@switch:~$ sudo systemctl restart dhcpd6@blue.service`
4. Check status:  
`cumulus@switch:~$ sudo systemctl status dhcpd6@blue.service`



You can create this configuration using the `vrf` command ([see above \(see page\)](#) for more details):

```
cumulus@switch:~$ sudo vrf
task exec red /usr/sbin
/dhcpd -f -q -cf /
/etc/dhcp/dhcpd-red.
conf -pf /var/run/dhcpd-
red.pid swp2
```



You can create this configuration using the `vrf` command ([see above \(see page\)](#) for more details):

```
cumulus@switch:~$ sudo vrf
task exec blue dhcpcd -6 -q
-cf /
/etc/dhcp/dhcpd6-blue.
conf -pf /var/run/dhcpd6-
blue.pid swp3
```

## Sample DHCP Relay Configuration

1. Create the file `isc-dhcp-relay-red` in `/etc/default/`. Here is sample content:

```
# Defaults for isc-dhcp-
relay initscript
# sourced by /etc/init.d
/isc-dhcp-relay
# installed at /etc/default
/isc-dhcp-relay by the
maintainer scripts
#
# This is a POSIX shell
fragment
#
# What servers should the
DHCP relay forward
requests to?
SERVERS="102.0.0.2"
# On what interfaces
should the DHCP relay
(dhrelay) serve DHCP
requests?
# Always include the interf
ace towards the DHCP
server.
# This variable requires a
-i for each interface
configured above.
# This will be used in the
actual dhcrelay command
```

## Sample DHCP6 Relay Configuration

1. Create the file `isc-dhcp-relay6-blue` in `/etc/default/`. Here is sample content:

```
# Defaults for isc-dhcp-
relay initscript
# sourced by /etc/init.d
/isc-dhcp-relay
# installed at /etc/default
/isc-dhcp-relay by the
maintainer scripts
#
# This is a POSIX shell
fragment
#
# What servers should the
DHCP relay forward
requests to?
#SERVERS="103.0.0.2"
# On what interfaces
should the DHCP relay
(dhrelay) serve DHCP
requests?
# Always include the interf
ace towards the DHCP
server.
# This variable requires a
-i for each interface
configured above.
# This will be used in the
actual dhcrelay command
```

```
# For example, "-i eth0 -i
eth1"
INTF_CMD="-i swp2s2 -i
swp2s3"
# Additional options that
are passed to the DHCP
relay daemon?
OPTIONS=""
```

2. Enable the DHCP relay:  
`cumulus@switch:~$ sudo systemctl
enable dhcrelay@red.service`
3. Start the DHCP relay:  
`cumulus@switch:~$ sudo systemctl
start dhcrelay@red.service`  
or  
`cumulus@switch:~$ sudo systemctl
restart dhcrelay@red.service`
4. Check status:  
`cumulus@switch:~$ sudo systemctl
status dhcrelay@red.service`



You can create this configuration using the `vrf` command ([see above \(see page\)](#) for more details):

```
cumulus@switch:~$ sudo vrf
task exec red /usr/sbin
/dhcrelay -d -q -i /
swp2s2 -i swp2s3
102.0.0.2
```

```
# For example, "-i eth0 -i
eth1"
INTF_CMD="-i swp18s0 -u
swp18s1"
# Additional options that
are passed to the DHCP
relay daemon?
OPTIONS="-pf /var/run
/dhcrelay6@blue.pid"
```

2. Enable the DHCP relay:  
`cumulus@switch:~$ sudo systemctl
enable dhcrelay6@blue.service`
3. Start the DHCP relay:  
`cumulus@switch:~$ sudo systemctl
start dhcrelay6@blue.service`  
or  
`cumulus@switch:~$ sudo systemctl
restart dhcrelay6@blue.service`
4. Check status:  
`cumulus@switch:~$ sudo systemctl
status dhcrelay6@blue.service`



You can create this configuration using the `vrf` command ([see above \(see page\)](#) for more details):

```
cumulus@switch:~$ sudo vrf
task exec blue /usr/sbin
/dhcrelay -d -q -6 -l /
swp18s0 -u swp18s1 -pf
/var/run/dhcrelay6@blue.pid
```

## Caveats

- The Penguin Computing Arctica 4804IP switch does not support VRFs.
- While there is a fixed limit of 64 VRF devices, Cumulus Networks has validated up to 20 VRF devices on a switch at one time.
- Table selection based on the incoming interface only; currently, packet attributes or output-interface-based selection are not available.
- BGP (IPv4/IPv6) is the only routing protocol supported currently. There is no support for OSPF (IPv4 /IPv6) at this time.

- Setting the router ID outside of BGP via the `router-id` option causes all BGP instances to get the same router ID. If you want each BGP instance to have its own router ID, specify the `router-id` under the BGP instance using `bgp router-id`. If both are specified, the one under the BGP instance overrides the one provided outside BGP.
- It is not possible to leak routes across VRFs within Quagga.

## Management VRF

*Management VRF* — a subset of VRF (see page 469) (virtual routing tables and forwarding) — provides a separation between the out-of-band management network and the in-band data plane network. For all VRFs, the *main* routing table is the default table for all of the data plane switch ports. With management VRF, a second table, *mgmt*, is used for routing through eth0.

Cumulus Linux only supports eth0 as the management interface. VLAN subinterfaces, bonds, bridges and the front panel switch ports are not supported as management interfaces.

When management VRF is enabled, logins to the switch are set into the management VRF context. IPv4 and IPv6 networking applications run by an administrator communicate out the management network by default. This default context does not impact services run through `systemd` and the `systemctl` command, and does not impact commands examining the state of the switch; for example, using the `ip` command to list links, neighbors or routes.

## Contents

- Enabling Management VRF (see page 489)
  - Enabling NTP (see page 490)
  - Enabling snmpd (see page 491)
  - Using ping or traceroute (see page 492)
- OSPF and BGP (see page 493)
  - Redistributing Routes in Management VRF (see page 493)
- SNMP Traps Use eth0 Only (see page 493)
- sFlow and Management VRF (see page 493)
- Using SSH within a Management VRF Context (see page 494)
- Viewing the Routing Tables (see page 494)
  - Viewing a Single Route (see page 494)
- Using the mgmt Interface Class (see page 495)
- Management VRF and DNS (see page 496)
- Incompatibility with cl-ns-mgmt (see page 496)

## Enabling Management VRF

To enable management VRF on eth0, complete the following steps:

1. Open `/etc/network/interfaces` in a text editor.
2. Configure the following, and save the file. Remember you must name the VRF *mgmt* to distinguish the mgmt VRF from a data plane VRF.

```
auto mgmt
iface mgmt
    address 127.0.0.1/8
    vrf-table auto

auto eth0
iface eth0 inet dhcp
    vrf mgmt
```

3. Reboot the switch to activate the mgmt VRF:

```
cumulus@switch:~$ sudo reboot
```

## Enabling NTP

To enable NTP to run in the mgmt VRF:

1. Configure the *mgmt* VRF in `/etc/networking/interfaces`. In this example, `eth0` is assigned an IP address and default route via DHCP.

```
auto mgmt
iface mgmt
    address 127.0.0.1/8
    vrf-table auto

auto eth0
iface eth0 inet dhcp
    vrf mgmt
```

2. Reboot the switch to activate the mgmt VRF.

```
cumulus@switch:~$ sudo reboot
```

3. By default, NTP is running in the default VRF. Stop NTP if it is currently running.

```
cumulus@switch:~$ sudo systemctl stop ntp.service
```

4. By default, NTP is configured to automatically start in the default VRF when the system boots. Disable NTP from automatically starting in the default VRF.

```
cumulus@switch:~$ sudo systemctl disable ntp.service
```

5. Start NTP in the mgmt VRF.

```
cumulus@switch:~$ sudo systemctl start ntp@mgmt
```

6. Verify that NTP peers are active.

```
cumulus@switch:~$ ntpq -pn
      remote          refid      st t when poll reach   delay
offset  jitter
=====
=====
*38.229.71.1    204.9.54.119    2 u   42    64   377   31.275
-0.625  3.105
-104.131.53.252 209.51.161.238    2 u   47    64   377   16.381
-5.251  0.681
+45.79.10.228   200.98.196.212    2 u   44    64   377   42.998
0.115  0.585
+74.207.240.206 127.67.113.92     2 u   43    64   377   73.240
-1.623  0.320
```

7. Enable ntp@mgmt so it starts when the switch boots:

```
cumulus@switch:~$ sudo systemctl enable ntp@mgmt
```

## ***Enabling snmpd***

To enable `snmpd` to run in the mgmt VRF:

1. Configure the mgmt VRF in `/etc/networking/interfaces`. In this example, eth0 is assigned an IP address and default route via DHCP.

```
auto mgmt
iface mgmt
  address 127.0.0.1/8
  vrf-table auto
```

```
auto eth0
iface eth0 inet dhcp
    vrf mgmt
```

2. Reboot the switch to activate the mgmt VRF.

```
cumulus@switch:~$ sudo reboot
```

3. Stop `snmpd` if it is running.

```
cumulus@switch:~$ sudo systemctl stop snmpd.service
```

4. Make sure `snmpd` does not try to start in the default VRF if the system is rebooted.

```
cumulus@switch:~$ sudo systemctl disable snmpd.service
```

5. Start `snmpd` in the mgmt VRF.

```
cumulus@switch:~$ sudo systemctl start snmpd@mgmt
```

6. Enable `snmpd@mgmt` so it starts when the switch boots:

```
cumulus@switch:~$ sudo systemctl enable snmpd@mgmt
```

## Using ping or traceroute

By default, issuing a `ping` or `traceroute` assumes the packet should be sent to the dataplane network (the main routing table). If you wish to use `ping` or `traceroute` on the management network, use the `-I` flag for ping and `-i` for traceroute.

```
cumulus@switch:~$ ping -I mgmt
```

or

```
cumulus@switch:~$ sudo traceroute -i mgmt
```

## OSPF and BGP

In general, no changes are required for either BGP or OSPF. Quagga was updated in Cumulus Linux 3.0 to be VRF-aware and automatically sends packets based on the switch port routing table. This includes BGP peering via loopback interfaces. BGP does routing lookups in the default table. However, one modification you may consider has to do with how your routes get redistributed.

## Redistributing Routes in Management VRF

Management VRF uses the mgmt table, including local routes. It does not affect how the routes are redistributed when using routing protocols such as OSPF and BGP.

To redistribute the routes in your network, use the **redistribute connected** command under BGP or OSPF. This enables the directly connected network out of eth0 to be advertised to its neighbor.



This also creates a route on the neighbor device to the management network through the data plane, which may not be desired.

Cumulus Networks recommends you always use route maps to control the advertised networks redistributed by the **redistribute connected** command. For example, you can specify a route map to redistribute routes in this way (for both BGP and OSPF):

```
<routing protocol>
redistribute connected route-map redistribute-connected

route-map redistribute-connected deny 100
  match interface eth0
!
route-map redistribute-connected permit 1000
```

## SNMP Traps Use eth0 Only

SNMP cannot use a switch port to send data. For any SNMP traps, this traffic gets sent out to eth0. Cumulus Networks plans to support switch ports in the future.



For SNMP, this restriction only applies to traps. SNMP polling is not affected.

## sFlow and Management VRF

With management VRF enabled, sFlow can send and receive packets through switch ports as well as the management port, as long as `hsflowd` has a route available out of the main table. To enable this, contact the [Cumulus Networks support team](#).

## Using SSH within a Management VRF Context

If you SSH to the switch through a switch port, it works as expected. If you need to SSH from the device out a switch port, use `vrf exec default ssh <ip_address_of_swport>`. For example:

```
cumulus@switch:~$ sudo ip addr show swp17
19: swp17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 500
    link/ether ec:f4:bb:fc:19:23 brd ff:ff:ff:ff:ff:ff
        inet 10.23.23.2/24 scope global swp17
            inet6 fe80::eef4:bbff:fefc:1923/64 scope link
                valid_lft forever preferred_lft forever

cumulus@switch:~$ sudo vrf exec default ssh 10.23.23.2 10.3.3.3
```

## Viewing the Routing Tables

When you look at the routing table with `ip route show`, you are looking at the switch port (*main*) table. You can also see the dataplane routing table with `ip route show table main`.

To look at information about eth0 (the management routing table), use `ip route show table mgmt`.

```
cumulus@switch:~$ ip route show table mgmt
default via 192.168.0.1 dev eth0

cumulus@switch:~$ ip route show
default via 10.23.23.3 dev swp17 proto zebra metric 20
10.3.3.3 via 10.23.23.3 dev swp17
10.23.23.0/24 dev swp17 proto kernel scope link src 10.23.23.2
192.168.0.0/24 dev eth0 proto kernel scope link src 192.168.0.11
```

## Viewing a Single Route

Note that if you use `ip route get` to return information about a single route, the command resolves over the *mgmt* table by default. To get information about the route in the switching silicon, use:

```
cumulus@switch:~$ ip route get <addr>
```

Or:

```
cumulus@switch:~$ sudo cl-rctl ip route show <addr>
```

To get the route for any VRF, use `ip route get <addr> oif <vrf name>`. So to get the route for the mgmt VRF, run:

```
cumulus@switch:~$ ip route get <addr> oif mgmt
```

## **Using the `mgmt` Interface Class**

In `ifupdown2` interface classes (see page 136) are used to create a user-defined grouping for interfaces. The special class `mgmt` is available to separate the switch's management interfaces from the data interfaces. This allows you to manage the data interfaces by default using `ifupdown2` commands. Performing operations on the `mgmt` interfaces requires specifying the `--allow-mgmt` option, which prevents inadvertent outages on the management interfaces. Cumulus Linux by default brings up all interfaces in both the `auto` (default) class and the `mgmt` interface class when the switch boots.

You configure the management interface in `/etc/network/interfaces`. In the example below, the management interface, `eth0`, and the mgmt VRF stanzas are added to the `mgmt` interface class:

```
auto lo
iface lo inet loopback

allow-mgmt eth0
iface eth0 inet dhcp
    vrf mgmt

allow-mgmt mgmt
iface mgmt
    address 127.0.0.1/8
    vrf-table auto
```

When you run `ifupdown2` commands against the interfaces in the `mgmt` class, include `--allow=mgmt` with the commands. For example, to see which interfaces are in the `mgmt` interface class, run:

```
cumulus@switch:~$ ifquery l --allow=mgmt
eth0
mgmt
```

To reload the configurations for interfaces in the `mgmt` class, run:

```
cumulus@switch:~$ sudo ifreload --allow=mgmt
```

However, you can still bring the management interface up and down using `ifup eth0` and `ifdown eth0`.

## ***Management VRF and DNS***

Cumulus Linux supports both DHCP and static DNS entries over management VRF through IP FIB rules. These rules are added to direct lookups to the DNS addresses out of the management VRF. However, nameservers configured through DHCP are automatically updated, while statically configured nameservers (configured in `/etc/resolv.conf`) only get updated when you run `ifreload -a`.

Because DNS lookups are forced out of the management interface using FIB rules, this could affect data plane ports if there are overlapping addresses.

## ***Incompatibility with cl-ns-mgmt***



Management VRF has replaced the management namespace functionality in Cumulus Linux. The management namespace feature (via the `cl-ns-mgmt` utility) has been deprecated, and the `cl-ns-mgmt` command has been removed.

# Monitoring and Troubleshooting

This chapter introduces monitoring and troubleshooting Cumulus Linux.

## Contents

(Click to expand)

- [Contents \(see page 497\)](#)
- [Commands \(see page 497\)](#)
- [Using the Serial Console \(see page 497\)](#)
  - [Configuring the Serial Console on ARM Switches \(see page 497\)](#)
  - [Configuring the Serial Console on x86 Switches \(see page 498\)](#)
- [Diagnostics Using cl-support \(see page 499\)](#)
- [Sending Log Files to a syslog Server \(see page 501\)](#)
- [Next Steps \(see page 503\)](#)

## Commands

- [cl-support](#)
- [fw\\_setenv \(ARM switches only\)](#)

## Using the Serial Console

The serial console can be a useful tool for debugging issues, especially when you find yourself rebooting the switch often or if you don't have a reliable network connection.

The default serial console baud rate is 115200, which is the baud rate ONIE uses.

### ***Configuring the Serial Console on ARM Switches***

On ARM switches, the U-Boot environment variable `baudrate` identifies the baud rate of the serial console. To change the `baudrate` variable, use the `fw_setenv` command:

```
cumulus@switch:~$ sudo fw_setenv baudrate 9600
Updating environment variable: `baudrate'
Proceed with update [N/y]? y
```

You must reboot the switch for the `baudrate` change to take effect.

The valid values for `baudrate` are:

- 300

- 600
- 1200
- 2400
- 4800
- 9600
- 19200
- 38400
- 115200

## Configuring the Serial Console on x86 Switches

On x86 switches, you configure serial console baud rate by editing `grub`.

- !** Incorrect configuration settings in `grub` can cause the switch to be inaccessible via the console. Grub changes should be carefully reviewed before implementation.

The valid values for the baud rate are:

- 300
- 600
- 1200
- 2400
- 4800
- 9600
- 19200
- 38400
- 115200

To change the serial console baud rate:

1. Edit `/etc/default/grub`. The two relevant lines in `/etc/default/grub` are as follows; replace the 115200 value with a valid value specified above in the `--speed` variable in the first line and in the `console` variable in the second line:

```
GRUB_SERIAL_COMMAND="serial --port=0x2f8 --speed=115200 --word=8 --
parity=no --stop=1"
GRUB_CMDLINE_LINUX="console=ttyS1,115200n8
cl_platform=accton_as5712_54x"
```

2. After you save your changes to the grub configuration, type the following at the command prompt:

```
cumulus@switch:~$ update-grub
```

3. If you plan on accessing your switch's BIOS over the serial console, you need to update the baud rate in the switch BIOS. For more information, see [this knowledge base article](#).
4. Reboot the switch.

## Diagnostics Using cl-support

You can use **cl-support** to generate a single export file that contains various details and the configuration from a switch. This is useful for remote debugging and troubleshooting.

You should run **cl-support** before you submit a support request to Cumulus Networks as this file helps in the investigation of issues:

```
cumulus@switch:~$ sudo cl-support -h
Usage: cl-support [-h] [-s] [-t] [-v] [reason]...
Args:
[reason]: Optional reason to give for invoking cl-support.
          Saved into tarball's cmdline.args file.
Options:
-h: Print this usage statement
-s: Security sensitive collection
-t: User filename tag
-v: Verbose
-e MODULES: Enable modules. Comma separated module list (run with -e help
for module names)
-d MODULES: Disable modules. Comma separated module list (run with -d help
for module names)
```

Example output:

```
cumulus@switch:~$ ls /var/support
cl_support_20130806_032720.tar.xz
```

The directory structure is compressed using LZMA2 compression and can be extracted using the **tar** command:

```
cumulus@switch:~$ cd /var/support
cumulus@switch:~$ sudo tar xf cl_support_20160527_140040.tar
cumulus@switch:~$ ls -l cl_support_20160529_140040/
-rwxr-xr-x 1 root root 7724 May 27 14:00 cl-support
-rw-r--r-- 1 root root 52 May 27 14:00 cmdline.args
drwxr-xr-x 2 root root 4096 May 27 14:00 core
```

```
drwxr-xr-x 64 root root 4096 May 27 13:51 etc
drwxr-xr-x  4 root root 4096 May 27 14:00 proc
drwxr-xr-x  2 root root 4096 May 27 14:01 support
drwxr-xr-x  3 root root 4096 May 27 14:00 sys
drwxr-xr-x  3 root root 4096 Mar  8 15:22 var
```

The directory contains the following elements:

Directory	Description
core	Contains the core files on the switch, including those generated from <code>switchd</code> .
etc	Is a replica of the switch's <code>/etc</code> directory. <code>/etc</code> contains all the general Linux configuration files, as well as configurations for the system's network interfaces, <code>quagga</code> and other packages.
log	Is a replica of the switch's <code>/var/log</code> directory. Most Cumulus Linux log files are located in this directory. Notable log files include <code>switchd.log</code> , <code>daemon.log</code> , <code>quagga</code> log files, and <code>syslog</code> . For more information, read this <a href="#">knowledge base article</a> .
proc	Is a replica of the switch's <code>/proc</code> directory. In Linux, <code>/proc</code> contains runtime system information (like system memory, devices mounted, and hardware configuration). These files are not actual files but the current state of the system.
support	Is a set of files containing further system information, which is obtained by <code>cl-support</code> running commands such as <code>ps -aux</code> , <code>netstat -i</code> , and so forth — even the routing tables.

`cl-support`, when untarred, contains a `cmdline.args` file. This file indicates what reason triggered it. When contacting Cumulus Networks technical support, please attach the `cl-support` file if possible. For more information about `cl-support`, read [Understanding and Decoding the cl-support Output File](#) (see page 517).



If you have issues extracting the script with the `tar` command, like an error saying the file does not look like tar archive, try using the `unxz` command first:

```
cumulus@switch:~$ sudo unxz cl_support_20130729_140040.tar.xz
```

You can save a lot of disk space and perhaps some time if you do not run `unxz` on the tar file.

## Sending Log Files to a syslog Server

All logging on Cumulus Linux is done with `rsyslog`. `rsyslog` provides both local logging to the `syslog` file as well as the ability to export logs to an external `syslog` server. High precision timestamps are enabled for all `rsyslog` log files; here's an example:

```
2015-08-14T18:21:43.337804+00:00 cumulus switchd[3629]:  
switchd.c:1409 switchd version 1.0-cl2.5+5
```

**Local logging:** Most logs within Cumulus Linux are sent to files in the `/var/log` directory. Most relevant information is placed within the `/var/log/syslog` file. For more information on specific log files, see [Troubleshooting Log Files \(see page 520\)](#).

**Export logging:** To send `syslog` files to an external `syslog` server, add a rule specifying to copy all messages (`*.*`) to the IP address and switch port of your `syslog` server in the `rsyslog` configuration files as described below.

In the following example, 192.168.1.2 is the remote `syslog` server and 514 is the port number. For UDP-based syslog, use a single @ before the IP address: `@192.168.1.2:514`. For TCP-based syslog, use two @@ before the IP address: `@@192.168.1.2:514`.

1. Create a file called something like `/etc/rsyslog.d/90-remotesyslog.conf`. Make sure it starts with a number lower than 99 so that it executes before `99-syslog.conf`. Add content like the following:

```
## Copy all messages to the remote syslog server at 192.168.1.2 port  
514  
*.* @192.168.1.2:514
```

2. Restart `rsyslog`.

```
cumulus@switch:~$ sudo systemctl restart rsyslog.service
```



All Cumulus Linux rules are stored in separate files in `/etc/rsyslog.d/`, which are called at the end of the `GLOBAL DIRECTIVES` section of `/etc/rsyslog.conf`. As a result, the `RULES` section at the end of `rsyslog.conf` is ignored because the messages have to be processed by the rules in `/etc/rsyslog.d` and then dropped by the last line in `/etc/rsyslog.d/99-syslog.conf`.



In the case of the `switchd` rules file, the file must be numbered lower than 25. For example, `13-switchd-remote.conf`.

If you need to send other log files (e.g. `switchd` logs) to a `syslog` server, configure a new file in `/etc/rsyslog.d`, as described above, and add lines similar to the following lines:

```
## Logging switchd messages to remote syslog server
$ModLoad imfile
$InputFileName /var/log/switchd.log
$InputFileStateFile logfile-log
$InputFileTag switchd:
$InputFileSeverity info
$InputFileFacility local7
$InputFilePollInterval 5
$InputRunFileMonitor

if $programname == 'switchd' then @192.168.1.2:514
```

Then restart `syslog`:

```
cumulus@switch:~$ sudo systemctl restart rsyslog.service
```

In the above configuration, each setting is defined as follows:

Setting	Description
\$ModLoad <i>imfile</i>	Enables the <code>rsyslog</code> module to watch file contents.
\$InputFileName	The file to be sent to the <code>syslog</code> server. In this example, you are going to send changes made to <code>/var/log/switchd.log</code> to the <code>syslog</code> server.
\$InputFileStateFile	This is used by <code>rsyslog</code> to track state of the file being monitored. This must be unique for each file being monitored.
\$InputFileTag	Defines the <code>syslog</code> tag that will precede the <code>syslog</code> messages. In this example, all logs are prefaced with <code>switchd</code> .
\$InputFileSeverity	Defines the logging severity level sent to the <code>syslog</code> server.
\$InputFileFacility	Defines the logging format. <code>local7</code> is common.
\$InputFilePollInterval	

Setting	Description
	Defines how frequently in seconds <code>rsyslog</code> looks for new information in the file. Lower values provide faster updates but create slightly more load on the CPU.
<code>\$InputRunFileMonitor</code>	Enables the file monitor module with the configured settings.

In most cases, the settings to customize include:

Setting	Description
<code>\$InputFileName</code>	The file to stream to the <code>syslog</code> server.
<code>\$InputFileStateFile</code>	A unique name for each file being watched.
<code>\$InputFileTag</code>	A prefix to the log message on the server.

Finally, the `if $programname` line is what sends the log files to the `syslog` server. It follows the same syntax as the `/var/log/syslog` file, where @ indicates UDP, 192.168.1.2 is the IP address of the `syslog` server, and 514 is the UDP port. The value `switchd` must match the value in `$InputFileTag`.

## Next Steps

The links below discuss more specific monitoring topics.

## Single User Mode - Boot Recovery

Use single user mode to assist in troubleshooting system boot issues or for password recovery. Entering single user mode is [platform-specific](#), so follow the appropriate steps for your x86 or ARM switch.

## Contents

(Click to expand)

- [Contents \(see page 503\)](#)
- [Entering Single User Mode on an x86 Switch \(see page 503\)](#)
- [Entering Single User Mode on an ARM Switch \(see page 505\)](#)

## Entering Single User Mode on an x86 Switch

1. Boot the switch, as soon as you see the GRUB menu.

GNU GRUB version 2.02~beta2-22+deb8u1

```
+-----+  
-----+  
| *Cumulus Linux GNU  
/Linux  
| Advanced options for Cumulus Linux GNU  
/Linux  
|  
ONIE  
|  
|  
|  
|  
+-----+  
-----+
```

2. Use the ^ and v arrow keys to select **Advanced options for Cumulus Linux GNU/Linux**. A menu similar to the following should appear:

```
GNU GRUB  version 2.02~beta2-22+deb8u1  
  
+-----+  
-----+  
| Cumulus Linux GNU/Linux, with Linux 4.1.0-cl-1-  
amd64  
|  
| Cumulus Linux GNU/Linux, with Linux 4.1.0-cl-1-amd64  
(sysvinit)  
|  
| *Cumulus Linux GNU/Linux, with Linux 4.1.0-cl-1-amd64 (recovery  
mode)  
|  
|  
|  
+-----+  
-----+
```

3. Select **Cumulus Linux GNU/Linux, with Linux 4.1.0-cl-1-amd64 (recovery mode)**.
4. Press ctrl-x to reboot.
5. After the system reboots, set a new password.

```
# passwd  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

6. Reboot the system:

```
# sync  
# reboot -f  
Restarting the system.
```

## ***Entering Single User Mode on an ARM Switch***

1. From the console, boot the switch, interrupting the U-Boot countdown to enter the U-Boot prompt. Enter the following:

```
=> setenv lbootargs init=/bin/bash  
=> boot
```

2. After the system boots, the shell command prompt appears. In this mode, you can change the root password or test a boot service that is hanging the boot process.

```
# passwd  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

3. Reboot the system.

```
cumulus@switch:~$ sudo sync  
cumulus@switch:~$ sudo reboot -f  
Restarting the system.
```

## Resource Diagnostics Using cl-resource-query

You can use **cl-resource-query** to retrieve information about host entries, MAC entries, L2 and L3 routes, and ECMPs (equal-cost multi-path routes, see [Load Balancing \(see page 391\)](#)) that are in use. This is especially useful because Cumulus Linux syncs routes between the kernel and the switching silicon. If the required resource pools in hardware fill up, new kernel routes can cause existing routes to move from being fully allocated to being partially allocated.

In order to avoid this, routes in the hardware should be monitored and kept below the ASIC limits. For example, on systems with a Broadcom Trident II chipset, the limits are as follows:

```
routes: 8092 <<< if all routes are IPv6, or 16384 if all routes are IPv4
long mask routes 2048 <<< these are routes with a mask longer than the
route mask limit
route mask limit 64
host_routes: 8192
ecmp_nhs: 16346
ecmp_nhs_per_route: 52
```

This translates to about 314 routes with ECMP next hops, if every route has the maximum ECMP NHs.

You can monitor this in Cumulus Linux with the **cl-resource-query** command. Results vary between switches running on different chipsets.

**cl-resource-query** results for a Mellanox Spectrum switch:

```
cumulus@switch:~$ sudo cl-resource-query
Host entries:          2,    0% of maximum value  5120
IPv4 neighbors:        2
IPv6 neighbors:        0
IPv4 entries:          33,   0% of maximum value 39936
IPv6 entries:          13,   0% of maximum value 15360
IPv4 Routes:           33
IPv6 Routes:           13
Total Routes:          46,   0% of maximum value 32768
ECMP nexthops:         0,    0% of maximum value 209664
MAC entries:           25,   0% of maximum value 409600
```

**cl-resource-query** results for a Broadcom Tomahawk switch:

```
cumulus@switch:~$ sudo cl-resource-query
Host entries:          1,    0% of maximum value  20480 <<< 2 IPv4
neighbors can use one entry
```

```

IPv4 neighbors:           1
IPv6 neighbors:           0
IPv4 entries:             5,   0% of maximum value  32668 <<< switch
overrides the SDK max limits
IPv6 entries:             4,   0% of maximum value  16384 <<<
IPv4 Routes:              5
IPv6 Routes:              4
Total Routes:             9,   0% of maximum value  32768
ECMP nexthops:            0,   0% of maximum value  16350
MAC entries:              2,   0% of maximum value  40960

```

cl-resource-query results for a Broadcom Trident II switch:

```

cumulus@switch:~$ sudo cl-resource-query
Host entries:           1,   0% of maximum value  8192 <<< this is
the default software-imposed limit, 50% of the hardware limit
IPv4 neighbors:           1       <<< these are counts of the number
of valid entries in the table
IPv6 neighbors:           0
IPv4 entries:             13,   0% of maximum value  32668
IPv6 entries:             18,   0% of maximum value  16384
IPv4 Routes:              13
IPv6 Routes:              18
Total Routes:             31,   0% of maximum value  32768
ECMP nexthops:            0,   0% of maximum value  16346
MAC entries:              12,   0% of maximum value  32768

```

## Monitoring System Hardware

You monitor system hardware in these ways, using:

- decode-syseeprom
- sensors
- smond
- [Net-SNMP \(see page 567\)](#)
- watchdog

## Contents

(Click to expand)

- [Contents \(see page 507\)](#)
- [Commands \(see page 508\)](#)

- Monitoring Hardware Using decode-syseeprom (see page 508)
  - Command Options (see page 509)
  - Related Commands (see page 509)
- Monitoring Hardware Using sensors (see page 509)
  - Command Options (see page 510)
- Monitoring Switch Hardware Using SNMP (see page 511)
- Monitoring System Units Using smond (see page 511)
  - Command Options (see page 511)
- Keeping the Switch Alive Using the Hardware Watchdog (see page 512)
- Configuration Files (see page 512)
- Useful Links (see page 512)

## **Commands**

- decode-syseeprom
- dmidecode
- lshw
- sensors
- smond

## **Monitoring Hardware Using decode-syseeprom**

The **decode-syseeprom** command enables you to retrieve information about the switch's EEPROM. If the EEPROM is writable, you can set values on the EEPROM.

For example:

```
cumulus@switch:~$ decode-syseeprom
TlvInfo Header:
  Id String:      TlvInfo
  Version:        1
  Total Length:  114
TLV Name          Code Len Value
-----  -----  ---  -----
Product Name      0x21   4  4804
Part Number       0x22   14 R0596-F0009-00
Device Version    0x26   1  2
Serial Number     0x23   19 D1012023918PE000012
Manufacture Date  0x25   19 10/09/2013 20:39:02
Base MAC Address  0x24   6  00:E0:EC:25:7B:D0
MAC Addresses     0x2A   2  53
Vendor Name       0x2D   17 Penguin Computing
Label Revision    0x27   4  4804
```

```

Manufacture Country 0x2C 2 CN
CRC-32             0xFE   4 0x96543BC5
(checksum valid)

```

## Command Options

Usage: /usr/cumulus/bin/decode-syseeprom [-a][-r][-s [args]][-t]

Option	Description
-h, -help	Displays the help message and exits.
-a	Prints the base MAC address for switch interfaces.
-r	Prints the number of MACs allocated for switch interfaces.
-s	Sets the EEPROM content if the EEPROM is writable. <b>args</b> can be supplied in command line in a comma separated list of the form '<field>=<value>, ...'. ' ' and '=' are illegal characters in field names and values. Fields that are not specified will default to their current values. If <b>args</b> are supplied in the command line, they will be written without confirmation. If <b>args</b> is empty, the values will be prompted interactively.
-t TARGET	Selects the target EEPROM ( <b>board</b> , <b>psu2</b> , <b>psu1</b> ) for the read or write operation; default is <b>board</b> .
-e, -serial	Prints the device serial number.

## Related Commands

You can also use the **dmidecode** command to retrieve hardware configuration information that's been populated in the BIOS.

You can use **apt-get** to install the **lshw** program on the switch, which also retrieves hardware configuration information.

## Monitoring Hardware Using sensors

The **sensors** command provides a method for monitoring the health of your switch hardware, such as power, temperature and fan speeds. This command executes **lm-sensors**.

For example:

```

cumulus@switch:~$ sensors
tmp75-i2c-6-48
Adapter: i2c-1-mux (chan_id 0)

```

```

temp1:          +39.0 C  (high = +75.0 C, hyst = +25.0 C)

tmp75-i2c-6-49
Adapter: i2c-1-mux (chan_id 0)
temp1:          +35.5 C  (high = +75.0 C, hyst = +25.0 C)

ltc4215-i2c-7-40
Adapter: i2c-1-mux (chan_id 1)
in1:            +11.87 V
in2:            +11.98 V
power1:         12.98 W
curr1:          +1.09 A

max6651-i2c-8-48
Adapter: i2c-1-mux (chan_id 2)
fan1:           13320 RPM  (div = 1)
fan2:           13560 RPM

```



Output from the `sensors` command varies depending upon the switch hardware you use, as each platform ships with a different type and number of sensors.

## Command Options

Usage: `sensors [OPTION]... [CHIP]...`

Option	Description
<code>-c, --config-file</code>	Specify a config file; use <code>-</code> after <code>-c</code> to read the config file from <code>stdin</code> ; by default, <code>sensors</code> references the configuration file in <code>/etc/sensors.d/</code> .
<code>-s, --set</code>	Executes set statements in the config file (root only); <code>sensors -s</code> is run once at boot time and applies all the settings to the boot drivers.
<code>-f, --fahrenheit</code>	Show temperatures in degrees Fahrenheit.
<code>-A, --no-adapter</code>	Do not show the adapter for each chip.
<code>--bus-list</code>	Generate bus statements for <code>sensors.conf</code> .

If `[CHIP]` is not specified in the command, all chip info will be printed. Example chip names include:

- `Im78-i2c-0-2d *-i2c-0-2d`

- lm78-i2c-0-\* \*-i2c-0-\*
- lm78-i2c-\*-\* 2d \*-i2c-\*-\* 2d
- lm78-i2c-\*-\* \*-i2c-\*-\*
- lm78-isa-0290 \*-isa-0290
- lm78-isa-\* \*-isa-\*
- lm78-\*

## **Monitoring Switch Hardware Using SNMP**

The Net-SNMP documentation has been moved to a new chapter, [available here \(see page 567\)](#).

## **Monitoring System Units Using smond**

The **smond** daemon monitors system units like power supply and fan, updates their corresponding LEDs, and logs the change in the state. Changes in system unit state are detected via the **cp1d** registers. **smond** utilizes these registers to read all sources, which impacts the health of the system unit, determines the unit's health, and updates the system LEDs.

Use **smonctl** to display sensor information for the various system units:

```
cumulus@switch:~$ smonctl
Board : OK
Fan : OK
PSU1 : OK
PSU2 : BAD
Temp1 (Networking ASIC Die Temp Sensor ) : OK
Temp10 (Right side of the board ) : OK
Temp2 (Near the CPU (Right) ) : OK
Temp3 (Top right corner ) : OK
Temp4 (Right side of Networking ASIC ) : OK
Temp5 (Middle of the board ) : OK
Temp6 (P2020 CPU die sensor ) : OK
Temp7 (Left side of the board ) : OK
Temp8 (Left side of the board ) : OK
Temp9 (Right side of the board ) : OK
```

## **Command Options**

Usage: **smonctl** [OPTION]... [CHIP]...

Option	Description
-s SENSOR, --sensor SENSOR	Displays data for the specified sensor.
-v, --verbose	Displays detailed hardware sensors data.

For more information, read `man smond` and `man smonctl`.

## Keeping the Switch Alive Using the Hardware Watchdog

Cumulus Linux includes a simplified version of the `wd_keepalive(8)` daemon from the standard Debian package `watchdog`. `wd_keepalive` writes to a file called `/dev/watchdog` periodically to keep the switch from resetting, at least once per minute. Each write delays the reboot time by another minute. After one minute of inactivity where `wd_keepalive` doesn't write to `/dev/watchdog`, the switch resets itself. The watchdog is enabled by default on all supported switches, and starts when you boot the switch, before `switchd` starts.

To enable the hardware watchdog, edit the `/etc/watchdog.d/<your_platform>` file and set `run_watchdog` to 1:

```
run_watchdog=1
```

To disable the watchdog, edit the `/etc/watchdog.d/<your_platform>` file and set `run_watchdog` to 0

```
run_watchdog=0
```

Then stop the daemon:

```
cumulus@switch:~$ sudo systemctl stop wd_keepalive.service
```

You can modify the settings for the watchdog — like the timeout setting and scheduler priority — in its configuration file, `/etc/watchdog.conf`.

## Configuration Files

- `/etc/cumulus/switchd.conf`
- `/etc/cumulus/sysledcontrol.conf`
- `/etc/sensors.d/<switch>.conf` - sensor configuration file (do **not** edit it!)
- `/etc/watchdog.conf`

## Useful Links

- <http://packages.debian.org/search?keywords=lshw>
- <http://lm-sensors.org>
- Net-SNMP tutorials

# Monitoring Virtual Device Counters

Cumulus Linux gathers statistics for VXLANS and VLANs using virtual device counters. These counters are supported on Tomahawk, Trident II+ and Trident II-based platforms only; see the [Cumulus Networks HCL](#) for a list of supported platforms.

You can retrieve the data from these counters using tools like `ip -s link show`, `ifconfig`, `/proc/net/dev`, or `netstat -i`.

## Contents

(Click to expand)

- [Contents \(see page 513\)](#)
- [Sample VXLAN Statistics \(see page 513\)](#)
- [Sample VLAN Statistics \(see page 514\)
  - \[For VLANs Using the non-VLAN-aware Bridge Driver \\(see page 514\\)\]\(#\)
  - \[For VLANs Using the VLAN-aware Bridge Driver \\(see page 515\\)\]\(#\)](#)
- [Configuring the Counters in switchd \(see page 516\)
  - \[Configuring the Poll Interval \\(see page 516\\)\]\(#\)
  - \[Configuring Internal VLAN Statistics \\(see page 516\\)\]\(#\)
  - \[Clearing Statistics \\(see page 516\\)\]\(#\)](#)
- [Caveats and Errata \(see page 517\)](#)

## Sample VXLAN Statistics

VXLAN statistics are available as follows:

- Aggregate statistics are available per VNI; this includes access and network statistics.
- Network statistics are available for each VNI and displayed against the VXLAN device. This is independent of the VTEP used, so this is a summary of the VNI statistics across all tunnels.
- Access statistics are available per VLAN subinterface.

First, get interface information regarding the VXLAN bridge:

```
cumulus@switch:~$ brctl show br-vxln16757104
bridge name      bridge id      STP enabled    interfaces
-vxln16757104   8000.443839006988   no           swp2s0.6
                                         swp2s1.6
                                         swp2s2.6
                                         swp2s3.6
                                         vxln16757104
```

To get VNI statistics, run:

```
cumulus@switch:~$ ip -s link show br-vxln16757104
62: br-vxln16757104: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    link/ether 44:38:39:00:69:88 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
      10848       158        0        0        0        0
    TX: bytes   packets   errors   dropped carrier collsns
      27816       541        0        0        0        0
```

To get access statistics, run:

```
cumulus@switch:~$ ip -s link show swp2s0.6
63: swp2s0.6@swp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-vxln16757104 state UP mode DEFAULT
    link/ether 44:38:39:00:69:88 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
      2680        39        0        0        0        0
    TX: bytes   packets   errors   dropped carrier collsns
      7558       140        0        0        0        0
```

To get network statistics, run:

```
cumulus@switch:~$ ip -s link show vxln16757104
61: vxln16757104: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-vxln16757104 state UNKNOWN mode DEFAULT
    link/ether e2:37:47:db:f1:94 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
      0         0         0         0         0         0
    TX: bytes   packets   errors   dropped carrier collsns
      0         0         0         9         0         0
```

## **Sample VLAN Statistics**

### **For VLANs Using the non-VLAN-aware Bridge Driver**

In this case, each bridge is a single L2 broadcast domain and is associated with an internal VLAN. This internal VLAN's counters are displayed as bridge netdev stats.

```
cumulus@switch:~$ brctl show br0
bridge name      bridge id          STP enabled    interfaces
br0              8000.443839006989    yes           bond0.100
                                         swp2s2.100

cumulus@switch:~$ ip -s link show br0
42: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT
    link/ether 44:38:39:00:69:89 brd ff:ff:ff:ff:ff:ff
    RX: bytes   packets   errors   dropped overrun mcast
    23201498    227514     0        0        0        0
    TX: bytes   packets   errors   dropped carrier collsns
    18198262    178443     0        0        0        0
```

## **For VLANs Using the VLAN-aware Bridge Driver**

For a bridge using the [VLAN-aware driver](#) (see page 235), the bridge is a just a container and each VLAN (VID /PVID) in the bridge is an independent L2 broadcast domain. As there is no netdev available to display these VLAN statistics, the `switchd` nodes are used instead:

```
cumulus@switch:~$ ifquery bridge
auto bridge
iface bridge inet static
    bridge-vlan-aware yes
    bridge-ports swp2s0 swp2s1
    bridge-stp on
    bridge-vids 2000-2002 4094
cumulus@switch:~$ ls /cumulus/switchd/run/stats/vlan/
2 2000 2001 2002 all
cumulus@switch:~$ cat /cumulus/switchd/run/stats/vlan/2000/aggregate
Vlan id                  : 2000
L3 Routed In Octets       : -
L3 Routed In Packets      : -
L3 Routed Out Octets      : -
L3 Routed Out Packets     : -
Total In Octets           : 375
Total In Packets          : 3
Total Out Octets          : 387
Total Out Packets         : 3
```

## Configuring the Counters in `switchd`

These counters are enabled by default. To configure them, use `cl-cfg` and configure them as you would any other `switchd` parameter (see page 126). The `switchd` parameters are as follows:

- `stats.vlan.aggregate`, which controls the statistics available for each VLAN. Its value defaults to *BRIEF*.
- `stats.vxlan.aggregate`, which controls the statistics available for each VNI (access and network). Its value defaults to *DETAIL*.
- `stats.vxlan.member`, which controls the statistics available for each local/access port in a VXLAN bridge. Its value defaults to *BRIEF*.

The values for each parameter can be one of the following:

- `NONE`: This disables the counter.
- `BRIEF`: This provides tx/rx packet/byte counters for the associated parameter.
- `DETAIL`: This provides additional feature-specific counters. In the case of `stats.vxlan.aggregate`, `DETAIL` provides access vs. network statistics. For the other types, `DETAIL` has the same effect as `BRIEF`.



If you change one of these settings on the fly, the new configuration applies only to those VNIs or VLANs set up after the configuration changed; previously allocated counters remain as is.

## Configuring the Poll Interval

The virtual device counters are polled periodically. This can be CPU intensive, so the interval is configurable in `switchd`, with a default of 2 seconds.

```
# Virtual devices hw-stat poll interval (in seconds)
#stats.vdev_hw_poll_interval = 2
```

## Configuring Internal VLAN Statistics

For debugging purposes, you may need to access packet statistics associated with internal VLAN IDs. These statistics are hidden by default, but can be configured in `switchd`:

```
#stats.vlan.show_internal_vlans = FALSE
```

## Clearing Statistics

Since `ethtool` is not supported for virtual devices, you cannot clear the statistics cache maintained by the kernel. You can clear the hardware statistics via `switchd`:

```
cumulus@switch:~$ sudo echo 1 > /cumulus/switchd/clear/stats/vlan
cumulus@switch:~$ sudo echo 1 > /cumulus/switchd/clear/stats/vxlan
cumulus@switch:~$
```

## Caveats and Errata

- Currently the CPU port is internally added as a member of all VLANs. Because of this, packets sent to the CPU are counted against the corresponding VLAN's tx packets/bytes. There is no workaround.
- When checking the virtual counters for the bridge, the TX count is the number of packets destined to the CPU before any hardware policers take effect. For example, if 500 broadcast packets are sent into the bridge, the CPU is also sent 500 packets. These 500 packets are policed by the default ACLs in Cumulus Linux, so the CPU might receive fewer than the 500 packets if the incoming packet rate is too high. The TX counter for the bridge should be equal to 500\*(number of ports in the bridge - incoming port + CPU port) or just 500 \* number of ports in the bridge.
- You cannot use `ethtool -S` for virtual devices. This is because the counters available via `netdev` are sufficient to display the vlan/vxlan counters currently supported in the hardware (only rx/tx packets/bytes are supported currently).

## Understanding and Decoding the cl-support Output File

***The cl-support command generates a tar archive of useful information for troubleshooting that can be auto-generated or manually created. To manually create it, run the cl-support command. The cl-support file is automatically generated when:***

- There is a `core` file dump of any application (not specific to Cumulus Linux, but something all Linux distributions support)
- Memory usage surpasses 90% of the total system memory (memory usage > 90% for 1 cycle)
- The `loadavg` over 15 minutes has on average greater than 2 (`loadavg (15min) > 2`)

The Cumulus Networks support team may request you submit the output from `cl-support` to help with the investigation of issues you might experience with Cumulus Linux.

```
cumulus@switch:~$ sudo cl-support -h
Usage: cl-support [-h] [reason]...
Args:
[reason]: Optional reason to give for invoking cl-support.
          Saved into tarball's reason.txt file.
Options:
-h: Print this usage statement
```

Example output:

```
cumulus@switch:~$ ls /var/support
cl_support_switch_20141204_203833
```

(Click to expand)

- The **cl-support** command generates a tar archive of useful information for troubleshooting that can be auto-generated or manually created. To manually create it, run the **cl-support** command. The **cl-support** file is automatically generated when: (see page 517)
- Understanding the File Naming Scheme (see page 518)
- Decoding the Output (see page 518)

## ***Understanding the File Naming Scheme***

The **cl-support** command generates a file under **/var/support** with the following naming scheme. The following example describes the file called **cl\_support\_switch\_20141204\_203833.tar.xz**.

<b>cl_support</b>	<b>switch</b>	<b>20141204</b>	<b>203833</b>
This is always prepended to the <b>.tar.gz</b> output.	This is the hostname of the switch where <b>cl-support</b> was executed.	The date in year, month, day; so 20141204 is December, 4th, 2014.	The time in hours, minutes, seconds; so 203833 is 20, 38, 33 (20:38:33) or the equivalent to 8:38:33 PM.

## ***Decoding the Output***

Decoding a **cl\_support** file is a simple process performed using the **tar** command. The following example illustrates extracting the **cl\_support** file:

```
tar -xf cl_support_switch_20141204_203834.tar.xz
```

The **-xf** options are defined here:

<b>Option</b>	<b>Description</b>
<b>-x</b>	Extracts to disk from the archive.
<b>-f</b>	Reads the archive from the specified file.

```
cumulus@switch:~$ ls -l cl_support_switch_20141204_203834/
-rwxr-xr-x 1 root root 7724 Jul 29 14:00 cl-support
-rw-r--r-- 1 root root 52 Jul 29 14:00 cmdline.args
drwxr-xr-x 2 root root 4096 Jul 29 14:00 core
drwxr-xr-x 64 root root 4096 Jul 29 13:51 etc
```

```
drwxr-xr-x  4 root root 4096 Jul 29 14:00 proc
drwxr-xr-x  2 root root 4096 Jul 29 14:01 support
drwxr-xr-x  3 root root 4096 Jul 29 14:00 sys
drwxr-xr-x  3 root root 4096 Aug  8 15:22 var
```

The **cl\_support** file, when untarred, contains a **reason.txt** file. This file indicates what reason triggered the event. When contacting Cumulus Networks technical support, please attach the **cl-support** file if possible.

The directory contains the following elements:

Directory	Description
cl-support	This is a copy of the <b>cl-support</b> script that generated the <b>cl_support</b> file. It is copied so Cumulus Networks knows exactly which files were included and which weren't. This helps to fix future <b>cl-support</b> requests in the future.
core	Contains the core files generated from the Cumulus Linux HAL (hardware abstraction layer) process, <b>switchd</b> .
etc	<b>etc</b> is the core system configuration directory. <b>cl-support</b> replicates the switch's <b>/etc</b> directory. <b>/etc</b> contains all the general Linux configuration files, as well as configurations for the system's network interfaces, <b>quagga</b> , and other packages.
var/log	<b>/var</b> is the "variable" subdirectory, where programs record runtime information. System logging, user tracking, caches and other files that system programs create and monitor go into <b>/var</b> . <b>cl-support</b> includes only the <b>log</b> subdirectory of the <b>var</b> system-level directory and replicates the switch's <b>/var/log</b> directory. Most Cumulus Linux log files are located in this directory. Notable log files include <b>switchd.log</b> , <b>daemon.log</b> , <b>quagga</b> log files, and <b>syslog</b> . For more information, read this <a href="#">knowledge base article</a> .
proc	<b>proc</b> (short for processes) provides system statistics through a directory-and-file interface. In Linux, <b>/proc</b> contains runtime system information (like system memory, devices mounted, and hardware configuration). <b>cl-support</b> simply replicates the switch's <b>/proc</b> directory to determine the current state of the system.
support	<b>support</b> is <b>not</b> a replica of the Linux file system like the other folders listed above. Instead, it is a set of files containing the output of commands from the command line. Examples include the output of <b>ps -aux</b> , <b>netstat -i</b> , and so forth — even the routing tables are included.

Here is more information on the file structure:

- [Troubleshooting the etc Directory \(see page 523\)](#) — In terms of sheer numbers of files, **/etc** contains the largest number of files to send to Cumulus Networks by far. However, log files could be significantly larger in file size.
- [Troubleshooting Log Files \(see page 520\)](#) — This guide highlights the most important log files to look at. Keep in mind, **cl-support** includes all of the log files.

- Troubleshooting the support Directory (see page 534) — This is an explanation of the `support` directory included in the `c1-support` output.

## Troubleshooting Log Files

The only real unique entity for logging on Cumulus Linux compared to any other Linux distribution is `switchd.log`, which logs the HAL (hardware abstraction layer) from hardware like the Broadcom or Mellanox ASIC.

This guide on [NixCraft](#) is amazing for understanding how `/var/log` works. The green highlighted rows below are the most important logs and usually looked at first when debugging.

Log	Description	Why is this important?
<code>/var/log/alternatives.log</code>	Information from the update-alternatives are logged into this log file.	
<code>/var/log/apt</code>	Information the <code>apt</code> utility can send logs here; for example, from <code>apt-get install</code> and <code>apt-get remove</code> .	
<code>/var/log/audit/*</code>	Contains log information stored by the Linux audit daemon, <code>audited</code> .	
<code>/var/log/auth.log</code>	Authentication logs. Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.	
<code>/var/log/autoprovision</code>	Logs output generated by running the <a href="#">zero touch provisioning</a> (see page 59) script.	
<code>/var/log/boot.log</code>	Contains information that is logged when the system boots.	
<code>/var/log/btmp</code>	This file contains information about failed login attempts. Use the <code>last</code> command to view the <code>btmp</code> file. For example:	
	<code>last -f /var/log/btmp   more</code>	
<code>/var/log/clagd.log</code>	Logs status of the <code>clagd</code> service (see page 244).	
	Log file for cron jobs.	

Log	Description	Why is this important?
/var/log/cron.log	Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.	
/var/log/daemon.log	<p>Contains information logged by the various background daemons that run on the system.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/debug	<p>Debugging information.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/dmesg	Contains kernel ring buffer information. When the system boots up, it prints number of messages on the screen that display information about the hardware devices that the kernel detects during boot process. These messages are available in the kernel ring buffer and whenever a new message arrives, the old message gets overwritten. You can also view the content of this file using the <b>dmesg</b> command.	<b>dmesg</b> is one of the few places to determine hardware errors.
/var/log/dpkg.log	Contains information that is logged when a package is installed or removed using the <b>dpkg</b> command.	
/var/log/faillog	Contains failed user login attempts. Use the <b>faillog</b> command to display the contents of this file.	
/var/log/fsck/*	The <b>fsck</b> utility is used to check and optionally repair one or more Linux filesystems.	
/var/log/kern.log	<p>Logs produced by the kernel and handled by <b>syslog</b>.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/lastlog	Formats and prints the contents of the last login log file.	
/var/log/lpr.log	<p>Printer logs.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/mail.log	<p>Mail server logs. Also includes <b>mail.err</b>, <b>mail.info</b> and <b>mail.warn</b>.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	

Log	Description	Why is this important?
/var/log/messages	<p>General messages and system-related information.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/news/*	<p>The <b>news</b> command keeps you informed of news concerning the system.</p> <p>Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.</p>	
/var/log/ntpstats	Logs for network configuration protocol.	
/var/log/openvswitch/*	<b>ovsdb-server</b> logs.	
/var/log/quagga/*	Where Quagga logs to once enabled.	This is how Cumulus Networks troubleshoots routing. For example an md5 or mtu mismatch with OSPF.
/var/log/switchd.log	The HAL log for Cumulus Linux.	This is specific to Cumulus Linux. Any <b>switchd</b> crashes are logged here.
/var/log/syslog	The main system log, which logs everything except auth-related messages.	The primary log; it's easiest to <b>grep</b> this file to see what occurred during a problem.
/var/log/user.log	Note that Cumulus Linux does not write to this log file; but because it's a standard file, Cumulus Linux creates it as a zero length file.	
/var/log/watchdog	<b>Hardware watchdog</b> (see page 512) log files.	

Log	Description	Why is this important?
/var/log/wtmp	Login records file.	

## Troubleshooting the `etc` Directory

The `c1-support` (see page 517) script replicates the `/etc` directory.

Files that `c1-support` deliberately excludes are:

File	Description
<code>/etc/nologin</code>	<code>nologin</code> prevents unprivileged users from logging into the system.
<code>/etc/alternatives</code>	<code>update-alternatives</code> creates, removes, maintains and displays information about the symbolic links comprising the Debian alternatives system.

This is the alphabetical of the output from running `ls -l` on the `/etc` directory structure created by `c1-support`. The green highlighted rows are the ones Cumulus Networks finds most important when troubleshooting problems.

File	Description	Why is this important?
<code>adduser.conf</code>	The file <code>/etc/adduser.conf</code> contains defaults for the programs <code>adduser</code> , <code>addgroup</code> , <code>deluser</code> , and <code>delgroup</code> .	
<code>adjtime</code>	Corrects the time to synchronize the <code>system clock</code> .	
<code>apt</code>	<code>apt</code> (Advanced Package Tool) is the command-line <code>tool for handling packages</code> . This folder contains all the configurations.	<code>apt</code> interactions or unsupported apps can affect machine performance.
<code>audisp</code>	The directory that contains <code>audisp-remote.conf</code> , which is the file that controls the <code>configuration of the audit remote logging subsystem</code> .	
<code>audit</code>	The directory that contains the <code>/etc/audit/auditd.conf</code> , which contains configuration information specific to the <code>audit daemon</code> .	
<code>bash.bashrc</code>	<code>Bash is an sh-compatible command language interpreter</code> that executes commands read from standard input or from a file.	
<code>bash_completion</code>		

File	Description	Why is this important?
	This points to <code>/usr/share/bash-completion/bash_completion</code> .	
<code>bash_completion.d</code>	This folder contains app-specific code for Bash completion on Cumulus Linux, such as <code>mstpcctl</code> .	
<code>bcm.d</code>	Broadcom-specific ASIC file structure (hardware interaction). If there are questions, <a href="#">contact the Cumulus Networks Support team</a> . This is unique to Cumulus Linux.	
<code>bindresvport.blacklist</code>	This file contains a list of port numbers between 600 and 1024, which should not be used by <code>bindresvport</code> .	
<code>ca-certificates</code>	The folder for <code>ca-certificates</code> . It is empty by default on Cumulus Linux; see below for more information.	
<code>ca-certificates.conf</code>	Each lines list the pathname of activated CA certificates under <code>/usr/share/ca-certificates</code> .	
<code>calendar</code>	The system-wide <a href="#">default calendar file</a> .	
<code>chef</code>	This is an example of something that is not included by default. In this instance, <code>cl-support</code> included the <code>chef</code> folder for some reason.	This is not installed by default, but this tool could have been installed or configured incorrectly, which is why it's included in the <code>cl-support</code> output.
<code>cron.d</code>	<code>cron</code> is a daemon that <a href="#">executes scheduled commands</a> .	
<code>cron.daily</code>	See above.	
<code>cron.hourly</code>	See above.	
<code>cron.monthly</code>	See above.	
<code>cron.weekly</code>	See above.	
<code>crontab</code>	See above.	
<code>cumulus</code>	This directory contains the following:	

File	Description	Why is this important?
	<ul style="list-style-type: none"> <li>ACL information, stored in the <code>acl</code> directory.</li> <li><code>switchd</code> configuration file, <code>switchd.conf</code>.</li> <li><code>qos</code>, which is under the <code>datapath</code> directory.</li> <li>The routing protocol process priority, <code>nice.conf</code>.</li> <li>The breakout cable configuration, under <code>ports.conf</code>.</li> </ul>	This folder is specific to Cumulus Linux and does not exist on other Linux platforms. For example, while you can configure <code>iptables</code> , to hardware accelerate rules into the hardware you need to use <code>c1-acltool</code> and have the rules under the <code>/etc/cumulus/acl/policy.d/&lt;filename.rules</code>
debconf.conf	Debconf is a <a href="#">configuration system for Debian packages</a> .	
debian_version	The complete <a href="#">Debian version string</a> .	
debsums-ignore	<code>debsums</code> <a href="#">verifies installed package files</a> against their MD5 checksums. This file identifies the packages to ignore.	
default	This folder contains files with configurable flags for many different applications (most installed by default or added manually). For example, <code>/etc/default/networking</code> has a flag for <code>EXCLUDE_INTERFACES=</code> , which is set to nothing by default, but a user could change it to something like <code>swp3</code> .	
deluser.conf	The file <code>/etc/deluser.conf</code> contains defaults for the programs <code>deluser</code> and <code>delgroup</code> .	
dhcp	This directory contains <a href="#">DHCP-specific information</a> .	
dpkg	The <a href="#">package manager</a> for Debian.	
e2fsck.conf	The <a href="#">configuration file for e2fsck</a> . It controls the default behavior of <code>e2fsck</code> while it checks ext2, ext3 or ext4 filesystems.	
environment	Utilized by <code>pam_env</code> for setting and unsetting environment variables.	
ethertypes	This file can be used to <a href="#">show readable characters</a> instead of hexadecimal numbers for the protocols. For example, <code>0x0800</code> will be represented by IPv4.	

File	Description	Why is this important?
fstab	Static information about the <a href="#">filesystems</a> .	
fstab.d	The directory that can contain additional <code>fstab</code> information; it is empty by default.	
fw_env.config	Configuration file utilized by <a href="#">U-Boot</a> .	
gai.conf	Configuration file for sorting the return information from <a href="#">getaddrinfo</a> .	
groff	The directory containing information for <code>groffer</code> , an application used for displaying <a href="#">Unix man pages</a> .	
group	The <code>/etc/group</code> file is a text file that <a href="#">defines the groups</a> on the system.	
group-	Backup for the <code>/etc/group</code> file.	
gshadow	<code>/etc/gshadow</code> contains the <a href="#">shadowed information for group accounts</a> .	
gshadow-	Backup for the <code>/etc/gshadow</code> file.	
host.conf	<a href="#">Resolver configuration file</a> , which contains options like <code>multi</code> that determines whether <code>/etc/hosts</code> will respond with multiple entries for DNS names.	
hostname	The <a href="#">system host name</a> , such as leaf1, spine1, sw1.	
hosts	The <a href="#">static table lookup</a> for hostnames.	
hosts.allow	The part of the <code>host_access</code> program for controlling a simple access control language. <code>hosts</code> . <code>allow=Access</code> is granted when a daemon/client pair matches an entry.	
hosts.deny	See <code>hosts.allow</code> above, except that access is denied when a daemon/client pair matches an entry.	
init	Default location of the <a href="#">system job configuration files</a> .	
init.d		

File	Description	Why is this important?
	In order for a service to start when the switch boots, you should add the <a href="#">necessary script</a> to the director here. The differences between <code>init</code> and <code>init.d</code> are explained well <a href="#">here</a> .	
inittab	The format of the <code>inittab</code> file used by the <code>sysv-</code> compatible <code>init</code> process.	
inputrc	The initialization file utilized by <code>readline</code> .	
insserv	This application <a href="#">enables installed system init scripts</a> ; this directory is empty by default.	
insserv.conf	Configuration file for <code>insserv</code> .	
insserv.conf.d	Additional directory for <code>insserv</code> configurations.	
iproute2	Directory containing values for the Linux command line tool <code>ip</code> .	
issue	<code>/etc/issue</code> is a text file that contains a <a href="#">message or system identification</a> to be printed before the login prompt.	
issue.net	Identification file for <code>telnet</code> sessions.	
ld.so.cache	Contains a <a href="#">compiled list of candidate libraries</a> previously found in the augmented library path.	
ld.so.conf	Used by the <code>ldconfig</code> tool, which <a href="#">configures dynamic linker run-time bindings</a> .	
ld.so.conf.d	The directory that contains additional <code>ld.so.conf</code> configuration (see above).	
ldap	The directory containing the <code>ldap.conf</code> configuration file used to set the system-wide default to be applied when running LDAP clients.	
libaudit.conf	Configuration file utilized by <code>get_auditfail_action</code> .	
libnl-3		

File	Description	Why is this important?
	Directory for the configuration relating to the <b>libnl library</b> , which is the core library for implementing the fundamentals required to use the netlink protocol such as socket handling, message construction and parsing, and sending and receiving of data.	
lldpd.d	Directory containing configuration files whose commands are executed by <b>lldpcli</b> at startup.	
localtime	Copy of the original data file for <b>/etc/timezone</b> .	
logcheck	Directory containing <b>logcheck.conf</b> and logfiles utilized by the <b>log check</b> program, which scans system logs for interesting lines.	
login.defs	<b>Shadow password suite configuration.</b>	
logrotate.conf	Rotates, compresses and mails <b>system logs</b> .	
logrotate.d	Directory containing additional log rotate configurations.	
lsb-release	Shows the current version of <b>Linux</b> on the system. Run <b>cat /etc/lsb-release</b> for output.	This shows you the version of the operating system you are running; also compare this to the output of <b>onie-select</b> .
magic	Used by the <b>file command</b> to determine file type. <b>magic</b> tests check for files with data in particular fixed formats.	
magic.mime	The <b>magic MIME type</b> causes the <b>file</b> command to output MIME type strings rather than the more traditional human readable ones.	
mailcap	The <b>mailcap</b> file is read by the metamail program to determine how to <b>display non-text at the local site</b> .	
mailcap.order	The <b>order of entries</b> in the <b>/etc/mailcap</b> file can be altered by editing the <b>/etc/mailcap.order</b> file.	
manpath.config		

File	Description	Why is this important?
	The <a href="#">manpath configuration file</a> is used by the manual page utilities to assess users' manpaths at run time, to indicate which manual page hierarchies (manpaths) are to be treated as system hierarchies and to assign them directories to be used for storing cat files.	
mime.types	MIME type description file for <a href="#">cups</a> .	
mke2fs.conf	Configuration file for <a href="#">mke2fs</a> , which is a program that <a href="#">creates an ext, ext3 or ext4 filesystem</a> .	
mlx	Mellanox-specific ASIC file structure (hardware interaction). If there are questions, <a href="#">contact the Cumulus Networks Support team</a> . This is unique to Cumulus Linux.	
modprobe.d	Configuration directory for <a href="#">modprobe</a> , which is a utility that can <a href="#">add and remove modules from the Linux kernel</a> .	
modules	The kernel modules to <a href="#">load at boot time</a> .	
motd	The contents of <a href="#">/etc/motd</a> ("message of the day") are displayed by <a href="#">pam_motd</a> after a successful login but just before it executes the login shell.	
mtab	The programs <a href="#">mount</a> and <a href="#">umount</a> maintain a list of <a href="#">currently mounted filesystems</a> in the <a href="#">/etc/mtab</a> file. If no arguments are given to <a href="#">mount</a> , this list is printed.	
nanorc	The GNU <a href="#">nano</a> <a href="#">rcfile</a> .	
network	Contains the <a href="#">network interface configuration</a> for <a href="#">ifup</a> and <a href="#">ifdown</a> .	The main configuration file is under <a href="#">/etc/network/interfaces</a> . This is where you configure L2 and L3 information for all of your front panel ports (swp interfaces). Settings like MTU, link speed, IP address information, VLANs are all done here.
networks	Network name information.	
nsswitch.conf		

File	Description	Why is this important?
	System databases and name service switch configuration file.	
ntp.conf	NTP (network time protocol) server configuration file.	
openvswitch	The directory containing the <b>conf.db</b> file, which is used by <b>ovsdb-server</b> .	
openvswitch-vtep	Configuration files used for the VTEP daemon and <b>ovsdb-server</b> .	
opt	Host-specific configuration files for <b>add-on applications</b> installed in <b>/opt</b> .	
os-release	Operating system identification.	
pam.conf	The PAM (pluggable authentication module) configuration file. When a PAM-aware privilege granting application is started, it activates its attachment to the PAM-API. This activation performs a number of tasks, the most important being the reading of the configuration file(s).	
pam.d	Alternate directory to configure PAM (see above).	
passwd	<b>User account information</b> .	
passwd-	Backup file for <b>/etc/passwd</b> .	
perl	Perl is an available scripting language. <b>/etc/perl</b> contains configuration files specific to Perl.	
profile	<b>/etc/profile</b> is utilized by <b>sysprofile</b> , a modular centralized shell configuration.	
profile.d	The directory version of the above, which contains configuration files.	
protocols	The <b>protocols definition file</b> , a plain ASCII file that describes the various DARPA net protocols that are available from the TCP/IP subsystem.	
ptm.d	The directory containing scripts that are run if <b>PTM</b> (see page 198) passes or fails.	Cumulus Linux-specific folder for PTM (prescriptive topology manager).

File	Description	Why is this important?
python	<code>python</code> is an available scripting language.	
python2.6	The 2.6 version of <code>python</code> .	
python2.7	The 2.7 version of <code>python</code> .	
quagga	Contains the configuration files for the <a href="#">Quagga routing suite (see page 393)</a> , the preferred Cumulus Linux routing engine.	
rc.local	The <code>/etc/rc.local</code> script is used by the system administrator to <a href="#">execute after all the normal system services are started</a> , at the end of the process of switching to a multiuser runlevel. You can use it to start a custom service, for example, a server that's installed in <code>/usr/local</code> . Most installations don't need <code>/etc/rc.local</code> ; it's provided for the minority of cases <a href="#">where it's needed</a> .	
rc0.d	Like <code>rc.local</code> , these scripts are booted by default, but the number of the folder represents the <a href="#">Linux runlevel</a> . This folder 0 represents runlevel 0 (halt the system).	
rc1.d	This is run level 1, which is single-user/minimal mode.	
rc2.d	Runlevels 2 through 5 are multiuser modes. Debian systems (such as Cumulus Linux) come with <code>id=2</code> , which indicates that the <a href="#">default runlevel will be 2 when the multi-user state is entered</a> , and the scripts in <code>/etc/rc2.d/</code> will be run.	
rc3.d	See above.	
rc4.d	See above.	
rc5.d	See above.	
rc6.d	Runlevel 6 is reboot the system.	
rcS.d	S stands for <i>single</i> and is equivalent to rc1.	
resolv.conf	<a href="#">Resolver configuration file</a> , which is where DNS is set (domain, nameserver and search).	You need DNS to reach the Cumulus Linux repository.

File	Description	Why is this important?
rmt	This is not a mistake. The <a href="#">shell script</a> <code>/etc/rmt</code> is provided for compatibility with other Unix-like systems, some of which have utilities that expect to find (and execute) <code>rmt</code> in the <code>/etc</code> directory on remote systems.	
rpc	The <code>rpc</code> file contains <a href="#">human-readable names</a> that can be used in place of RPC program numbers.	
rsyslog.conf	The <code>rsyslog.conf</code> file is the main configuration file for <code>rsyslogd</code> , which logs system messages on *nix systems.	
rsyslog.d	The directory containing additional configuration for <code>rsyslog.conf</code> (see above).	
securetty	This file lists terminals into which the <a href="#">root user can log in</a> .	
security	The <code>/etc/security</code> directory contains <a href="#">security-related configurations files</a> . Whereas PAM concerns itself with the methods used to authenticate any given user, the files under <code>/etc/security</code> are concerned with just what a user can or cannot do. For example, the <code>/etc/security/access.conf</code> file contains a list of which users are allowed to log in and from what host (for example, using telnet). The <code>/etc/security/limits.conf</code> file contains various system limits, such as maximum number of processes.	
selinux	<a href="#">NSA Security-Enhanced Linux</a> .	
sensors.d	The directory from which the <code>sensors</code> program loads its configuration; this is unique for each hardware platform. See also <a href="#">Monitoring System Hardware</a> (see page 507).	
sensors3.conf	The <code>sensors.conf</code> file describes how <code>libsensors</code> , and thus all programs using it, should translate the raw readings from the kernel modules to real-world values.	
services	<code>services</code> is a plain ASCII file providing a mapping between human-readable textual names for internet services and their underlying assigned port numbers and protocol types.	

File	Description	Why is this important?
shadow	shadow is a file that contains the password information for the system's accounts and optional aging information.	
shadow-	The backup for the /etc/shadow file.	
shells	The pathnames of valid login shells.	
skel	The skeleton directory (usually /etc/skel) is used to copy default files and also sets a umask for the creation used by <code>pam_mkhomedir</code> .	
snmp	Interface functions to the SNMP (simple network management protocol) toolkit.	
ssh	The ssh configuration.	
ssl	The OpenSSL ss1 library implements the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. This directory holds certificates and configuration.	
staff-group-for-usr-local	Use <code>cat</code> or <code>more</code> on this file to learn more information, see <a href="http://bugs.debian.org/299007">http://bugs.debian.org/299007</a> .	
sudoers	The sudoers policy plugin determines a user's sudo privileges.	
sudoers.d	The directory file containing additional sudoers configuration (see above).	
sysctl.conf	Configures kernel parameters at boot.	
sysctl.d	The directory file containing additional configuration (see above).	
systemd	<code>systemd</code> system and service manager.	
terminfo	Terminal capability database.	
timezone	If this file exists, it is read and its contents are used as the time zone name.	
ucf.conf	The update configuration file preserves user changes in configuration files.	

udev	Dynamic device management.	
ufw	Provides both a command line interface and a framework for managing a <a href="#">netfilter firewall</a> .	
vim	Configuration file for command line tool <a href="#">vim</a> .	
wgetrc	Configuration file for command line tool <a href="#">wget</a> .	

## Troubleshooting the support Directory

The `support` directory is unique in the fact that it is not a copy of the switch's filesystem. Actually, it is the output from various commands. For example:

File	Equivalent Command	Description
<code>support /ip. addr</code>	<code>cumulus@switch:~\$ ip addr show</code>	This shows you all the interfaces (including swp front panel ports), IP address information, admin state and physical state.

## Troubleshooting Network Interfaces

The following sections describe various ways you can troubleshoot `ifupdown2`.

### Contents

(Click to expand)

- [Contents \(see page 534\)](#)
- [Enabling Logging for Networking \(see page 534\)](#)
- [Using ifquery to Validate and Debug Interface Configurations \(see page 535\)](#)
- [Debugging Mako Template Errors \(see page 536\)](#)
- [ifdown Cannot Find an Interface that Exists \(see page 537\)](#)
- [Removing All References to a Child Interface \(see page 537\)](#)
- [MTU Set on a Logical Interface Fails with Error: "Numerical result out of range" \(see page 538\)](#)
- [Interpreting iproute2 batch Command Failures \(see page 538\)](#)
- [Understanding the "RTNETLINK answers: Invalid argument" Error when Adding a Port to a Bridge \(see page 539\)](#)

### Enabling Logging for Networking

The `/etc/default/networking` file contains two settings for logging:

- To get `ifupdown2` logs when the switch boots (stored in `syslog`)
- To enable logging when you run `systemctl [start|stop|reload] networking.service`

This file also contains an option for excluding interfaces when you boot the switch or run `systemctl start|stop|reload networking.service`. You can exclude any interface specified in `/etc/network/interfaces`. These interfaces do not come up when you boot the switch or start/stop/reload the networking service.

```
$cat /etc/default/networking
#
#
# Parameters for the /etc/init.d/networking script
#
#
# Change the below to yes if you want verbose logging to be enabled
VERBOSE="no"

# Change the below to yes if you want debug logging to be enabled
DEBUG="no"

# Change the below to yes if you want logging to go to syslog
SYSLOG="no"

# Exclude interfaces
EXCLUDE_INTERFACES=
```

## ***Using ifquery to Validate and Debug Interface Configurations***

You use `ifquery` to print parsed `interfaces` file entries.

To use `ifquery` to pretty print `iface` entries from the `interfaces` file, run:

```
cumulus@switch:~$ sudo ifquery bond0
auto bond0
iface bond0
    address 14.0.0.9/30
    address 2001:ded:beef:2::1/64
    bond-slaves swp25 swp26
```

Use `ifquery --check` to check the current running state of an interface within the `interfaces` file. It will return exit code 0 or 1 if the configuration does not match. The line `bond-xmit-hash-policy layer3+7` below fails because it should read `bond-xmit-hash-policy layer3+4`.

```
cumulus@switch:~$ sudo ifquery --check bond0
iface bond0
    bond-xmit-hash-policy layer3+7 [fail]
    bond-slaves swp25 swp26 [pass]
    address 14.0.0.9/30 [pass]
    address 2001:ded:beef:2::1/64 [pass]
```



`ifquery --check` is an experimental feature.

Use `ifquery --running` to print the running state of interfaces in the `interfaces` file format:

```
cumulus@switch:~$ sudo ifquery --running bond0
auto bond0
iface bond0
    bond-slaves swp25 swp26
    address 14.0.0.9/30
    address 2001:ded:beef:2::1/64
```

`ifquery --syntax-help` provides help on all possible attributes supported in the `interfaces` file. For complete syntax on the `interfaces` file, see `man interfaces` and `man ifupdown-addons-interfaces`.

You can use `ifquery --print-savedstate` to check the `ifupdown2` state database. `ifdown` works only on interfaces present in this state database.

```
cumulus@leaf1$ sudo ifquery --print-savedstate eth0
auto eth0
iface eth0 inet dhcp
```

## Debugging Mako Template Errors

An easy way to debug and get details about template errors is to use the `mako-render` command on your `interfaces` template file or on `/etc/network/interfaces` itself.

```
cumulus@switch:~$ sudo mako-render /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback
```

```

# The primary network interface
auto eth0
iface eth0 inet dhcp
#auto eth1
#iface eth1 inet dhcp

# Include any platform-specific interface configuration
source /etc/network/interfaces.d/*.*if

# ssim2 added

auto swp45
iface swp45

auto swp46
iface swp46

cumulus@switch:~$ sudo mako-render /etc/network/interfaces.d
/<interfaces_stub_file>

```

## ***ifdown Cannot Find an Interface that Exists***

If you are trying to bring down an interface that you know exists, use `ifdown` with the `--use-current-config` option to force `ifdown` to check the current `/etc/network/interfaces` file to find the interface. This can solve issues where the `ifup` command issues for that interface was interrupted before it updated the state database. For example:

```

cumulus@switch:~$ sudo ifdown br0
error: cannot find interfaces: br0 (interface was probably never up ?)

cumulus@switch:~$ sudo brctl show
bridge name      bridge id           STP enabled      interfaces
br0              8000.44383900279f    yes            downlink
                                         peerlink

cumulus@switch:~$ sudo ifdown br0 --use-current-config

```

## ***Removing All References to a Child Interface***

If you have a configuration with a child interface, whether it's a VLAN, bond or another physical interface, and you remove that interface from a running configuration, you must remove every reference to it in the configuration. Otherwise, the interface continues to be used by the parent interface.

For example, consider the following configuration:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto bond1
iface bond1
    bond-slaves swp2 swp1

auto bond3
iface bond3
    bond-slaves swp8 swp6 swp7

auto br0
iface br0
    bridge-ports swp3 swp5 bond1 swp4 bond3
    bridge-pathcosts swp3=4 swp5=4 swp4=4
    address 11.0.0.10/24
    address 2001::10/64
```

Notice that bond1 is a member of br0. If you comment out or simply delete bond1 from `/etc/network/interfaces`, you must remove the reference to it from the br0 configuration. Otherwise, if you reload the configuration with `ifreload -a`, bond1 is still part of br0.

## ***MTU Set on a Logical Interface Fails with Error: "Numerical result out of range"***

This error occurs when the [MTU](#) (see page 152) you are trying to set on an interface is higher than the MTU of the lower interface or dependent interface. Linux expects the upper interface to have an MTU less than or equal to the MTU on the lower interface.

In the example below, the swp1.100 VLAN interface is an upper interface to physical interface swp1. If you want to change the MTU to 9000 on the VLAN interface, you must include the new MTU on the lower interface swp1 as well.

```
auto swp1.100
iface swp1.100
    mtu 9000

auto swp1
iface swp1
    mtu 9000
```

## Interpreting iproute2 batch Command Failures

ifupdown2 batches `iproute2` commands for performance reasons. A batch command contains `ip -force -batch -` in the error message. The command number that failed is at the end of this line:  
`Command failed -:1.`

Below is a sample error for the command 1: `link set dev host2 master bridge`. There was an error adding the bond `host2` to the bridge named `bridge` because `host2` did not have a valid address.

```
error: failed to execute cmd 'ip -force -batch - [link set dev host2 master
bridge
addr flush dev host2
link set dev host1 master bridge
addr flush dev host1
]' (RTNETLINK answers: Invalid argument
Command failed -:1)
warning: bridge configuration failed (missing ports)
```

## Understanding the "RTNETLINK answers: Invalid argument" Error when Adding a Port to a Bridge

This error can occur when the bridge port does not have a valid hardware address.

This can typically occur when the interface being added to the bridge is an incomplete bond; a bond without slaves is incomplete and does not have a valid hardware address.

## Monitoring Interfaces and Transceivers Using ethtool

The `ethtool` command enables you to query or control the network driver and hardware settings. It takes the device name (like `swp1`) as an argument. When the device name is the only argument to `ethtool`, it prints the current settings of the network device. See `man ethtool(8)` for details. Not all options are currently supported on switch port interfaces.

## Contents

(Click to expand)

- [Contents \(see page 539\)](#)
- [Commands \(see page 539\)](#)
- [Monitoring Interface Status Using ethtool \(see page 539\)
  - \[Viewing and Clearing Interface Counters \\(see page 541\\)\]\(#\)](#)
- [Monitoring Switch Port SFP/QSFP Hardware Information Using ethtool \(see page 542\)](#)

## Commands

- [cl-netstat](#)
- [ethtool](#)

## Monitoring Interface Status Using ethtool

To check the status of an interface using ethtool:

```
cumulus@switch:~$ ethtool swp1
Settings for swp1:
    Supported ports: [ FIBRE ]
    Supported link modes:  1000baseT/Full
                           10000baseT/Full
    Supported pause frame use: No
    Supports auto-negotiation: No
    Advertised link modes:  1000baseT/Full
    Advertised pause frame use: No
    Advertised auto-negotiation: No
    Speed: 10000Mb/s
    Duplex: Full
    Port: FIBRE
    PHYAD: 0
    Transceiver: external
    Auto-negotiation: off
    Current message level: 0x00000000 (0)

    Link detected: yes
```

To query interface statistics:

```
cumulus@switch:~$ sudo ethtool -S swp1
NIC statistics:
    HwIfInOctets: 1435339
    HwIfInUcastPkts: 11795
    HwIfInBcastPkts: 3
    HwIfInMcastPkts: 4578
    HwIfOutOctets: 14866246
    HwIfOutUcastPkts: 11791
    HwIfOutMcastPkts: 136493
    HwIfOutBcastPkts: 0
    HwIfInDiscards: 0
    HwIfInL3Drops: 0
    HwIfInBufferDrops: 0
    HwIfInAclDrops: 28
    HwIfInDot3LengthErrors: 0
    HwIfInErrors: 0
    SoftInErrors: 0
```

```

SoftInDrops: 0
SoftInFrameErrors: 0
HwIfOutDiscards: 0
HwIfOutErrors: 0
HwIfOutQDrops: 0
HwIfOutNonQDrops: 0
SoftOutErrors: 0
SoftOutDrops: 0
SoftOutTxFifoFull: 0
HwIfOutQLen: 0

```

## ***Viewing and Clearing Interface Counters***

Interface counters contain information about an interface. You can view this information when you run **cl-netstat**, **ifconfig**, or **cat /proc/net/dev**. You can also use **cl-netstat** to save or clear this information:

```

cumulus@switch:~# sudo cl-netstat
Kernel Interface table
Iface      MTU Met          RX_OK RX_ERR RX_DRP RX_OVR          TX_OK TX_ERR
TX_DRP TX_OVR     Flg
-----
eth0      1500 0            611   0     0     0           487   0
0          0   BMRU
lo       16436 0            0     0     0     0           0     0
0          0   LRU
swp1      1500 0            0     0     0     0           0     0
0          0   BMU

cumulus@switch:~# sudo :~# cl-netstat -c
Cleared counters

```

Option	Description
-c	Copies and clears statistics. It does not clear counters in the kernel or hardware.
-d	Deletes saved statistics, either the <b>uid</b> or the specified tag.
-D	Deletes all saved statistics.
-l	Lists saved tags.

Option	Description
-r	Displays raw statistics (unmodified output of <code>cl-netstat</code> ).
-t <tag name>	Saves statistics with <tag name>.
-v	Prints <code>cl-netstat</code> version and exits.

## Monitoring Switch Port SFP/QSFP Hardware Information Using ethtool

To see hardware capabilities and measurement information on the SFP or QSFP module installed in a particular port, use the `ethtool -m` command. If the SFP/QSFP supports Digital Optical Monitoring (that is, the `Optical diagnostics support` field in the output below is set to Yes), the optical power levels and thresholds are also printed below the standard hardware details.

In the sample output below, you can see that this module is a 1000BASE-SX short-range optical module, manufactured by JDSU, part number PLRXPL-VI-S24-22. The second half of the output displays the current readings of the Tx power levels (`Laser output power`) and Rx power (`Receiver signal average optical power`), temperature, voltage and alarm threshold settings.

```
cumulus@switch$ sudo ethtool -m swp3
      Identifier                      : 0x03 (SFP)
      Extended identifier              : 0x04 (GBIC/SFP defined
by 2-wire interface ID)
      Connector                       : 0x07 (LC)
      Transceiver codes               : 0x00 0x00 0x00 0x01
0x20 0x40 0x0c 0x05
      Transceiver type                : Ethernet: 1000BASE-SX
      Transceiver type                : FC: intermediate
distance (I)
      Transceiver type                : FC: Shortwave laser w/o
OFC (SN)
      Transceiver type                : FC: Multimode, 62.5um
(M6)
      Transceiver type                : FC: Multimode, 50um (M5)
      Transceiver type                : FC: 200 MBytes/sec
      Transceiver type                : FC: 100 MBytes/sec
      Encoding                         : 0x01 (8B/10B)
      BR, Nominal                     : 2100MBd
      Rate identifier                  : 0x00 (unspecified)
      Length (SMF,km)                 : 0km
      Length (SMF)                    : 0m
      Length (50um)                   : 300m
      Length (62.5um)                 : 150m
      Length (Copper)                 : 0m
```

Length (OM3)	:	0m
Laser wavelength	:	850nm
Vendor name	:	JDSU
Vendor OUI	:	00:01:9c
Vendor PN	:	PLRXPL-VI-S24-22
Vendor rev	:	1
Optical diagnostics support	:	Yes
Laser bias current	:	21.348 mA
Laser output power	:	0.3186 mW / -4.97 dBm
Receiver signal average optical power	:	0.3195 mW / -4.96 dBm
Module temperature	:	41.70 degrees C /
107.05 degrees F		
Module voltage	:	3.2947 V
Alarm/warning flags implemented	:	Yes
Laser bias current high alarm	:	Off
Laser bias current low alarm	:	Off
Laser bias current high warning	:	Off
Laser bias current low warning	:	Off
Laser output power high alarm	:	Off
Laser output power low alarm	:	Off
Laser output power high warning	:	Off
Laser output power low warning	:	Off
Module temperature high alarm	:	Off
Module temperature low alarm	:	Off
Module temperature high warning	:	Off
Module temperature low warning	:	Off
Module voltage high alarm	:	Off
Module voltage low alarm	:	Off
Module voltage high warning	:	Off
Module voltage low warning	:	Off
Laser rx power high alarm	:	Off
Laser rx power low alarm	:	Off
Laser rx power high warning	:	Off
Laser rx power low warning	:	Off
Laser bias current high alarm threshold	:	10.000 mA
Laser bias current low alarm threshold	:	1.000 mA
Laser bias current high warning threshold	:	9.000 mA
Laser bias current low warning threshold	:	2.000 mA
Laser output power high alarm threshold	:	0.8000 mW / -0.97 dBm
Laser output power low alarm threshold	:	0.1000 mW / -10.00 dBm
Laser output power high warning threshold	:	0.6000 mW / -2.22 dBm
Laser output power low warning threshold	:	0.2000 mW / -6.99 dBm
Module temperature high alarm threshold	:	90.00 degrees C /
194.00 degrees F		
Module temperature low alarm threshold	:	-40.00 degrees C /

```
-40.00 degrees F
    Module temperature high warning threshold : 85.00 degrees C /
185.00 degrees F
    Module temperature low warning threshold : -40.00 degrees C /
-40.00 degrees F
    Module voltage high alarm threshold      : 4.0000 V
    Module voltage low alarm threshold       : 0.0000 V
    Module voltage high warning threshold   : 3.6450 V
    Module voltage low warning threshold    : 2.9550 V
    Laser rx power high alarm threshold    : 1.6000 mW / 2.04 dBm
    Laser rx power low alarm threshold     : 0.0100 mW / -20.00 dBm
    Laser rx power high warning threshold : 1.0000 mW / 0.00 dBm
    Laser rx power low warning threshold  : 0.0200 mW / -16.99 dBm
```

## Network Troubleshooting

Cumulus Linux contains a number of command line and analytical tools to help you troubleshoot issues with your network.

### Contents

(Click to expand)

- [Contents \(see page 544\)](#)
- [Commands \(see page 545\)](#)
- [Checking Reachability Using ping \(see page 545\)](#)
- [Printing Route Trace Using traceroute \(see page 545\)](#)
- [Manipulating the System ARP Cache \(see page 546\)](#)
- [Generating Traffic Using mz \(see page 546\)](#)
- [Creating Counter ACL Rules \(see page 547\)](#)
- [Configuring SPAN and ERSPAN \(see page 548\)
  - \[Configuring SPAN for Switch Ports \\(see page 549\\)\]\(#\)
  - \[Configuring SPAN for Bonds \\(see page 552\\)\]\(#\)
  - \[Configuring ERSPAN \\(see page 553\\)\]\(#\)
  - \[Selective Spanning \\(see page 554\\)\]\(#\)
  - \[Removing SPAN Rules \\(see page 556\\)\]\(#\)](#)
- [Monitoring Control Plane Traffic with tcpdump \(see page 556\)](#)
- [Configuration Files \(see page 557\)](#)
- [Useful Links \(see page 558\)](#)
- [Caveats and Errata \(see page 558\)](#)

## Commands

- arp
- cl-acltool
- ip
- mz
- ping
- tcpdump
- traceroute

## Checking Reachability Using ping

`ping` is used to check reachability of a host. `ping` also calculates the time it takes for packets to travel the round trip. See `man ping` for details.

To test the connection to an IPv4 host:

```
cumulus@switch:~$ ping 192.0.2.45
PING 192.0.2.45 (192.0.2.45) 56(84) bytes of data.
64 bytes from 192.0.2.45: icmp_req=1 ttl=53 time=40.4 ms
64 bytes from 192.0.2.45: icmp_req=2 ttl=53 time=39.6 ms
...
...
```

To test the connection to an IPv6 host:

```
cumulus@switch:~$ ping6 -I swp1 2001::db8:ff:fe00:2
PING 2001::db8:ff:fe00:2(2001::db8:ff:fe00:2) from 2001::db8:ff:fe00:1
swp1: 56 data bytes
64 bytes from 2001::db8:ff:fe00:2: icmp_seq=1 ttl=64 time=1.43 ms
64 bytes from 2001::db8:ff:fe00:2: icmp_seq=2 ttl=64 time=0.927 ms
```

## Printing Route Trace Using traceroute

`traceroute` tracks the route that packets take from an IP network on their way to a given host. See `man traceroute` for details.

To track the route to an IPv4 host:

```
cumulus@switch:~$ traceroute www.google.com
traceroute to www.google.com (74.125.239.49), 30 hops max, 60 byte packets
1  cumulusnetworks.com (192.168.1.1)  0.614 ms  0.863 ms  0.932 ms
...
...
```

```

5 core2-1-1-0.pao.net.google.com (198.32.176.31) 22.347 ms 22.584 ms
24.328 ms
6 216.239.49.250 (216.239.49.250) 24.371 ms 25.757 ms 25.987 ms
7 72.14.232.35 (72.14.232.35) 27.505 ms 22.925 ms 22.323 ms
8 nuq04s19-in-f17.1e100.net (74.125.239.49) 23.544 ms 21.851 ms 22.604
ms

```

## ***Manipulating the System ARP Cache***

`arp` manipulates or displays the kernel's IPv4 network neighbor cache. See `man arp` for details.

To display the ARP cache:

```

cumulus@switch:~$ arp -a
? (11.0.2.2) at 00:02:00:00:00:10 [ether] on swp3
? (11.0.3.2) at 00:02:00:00:00:01 [ether] on swp4
? (11.0.0.2) at 44:38:39:00:01:c1 [ether] on swp1

```

To delete an ARP cache entry:

```

cumulus@switch:~$ arp -d 11.0.2.2
cumulus@switch:~$ arp -a
? (11.0.2.2) at <incomplete> on swp3
? (11.0.3.2) at 00:02:00:00:00:01 [ether] on swp4
? (11.0.0.2) at 44:38:39:00:01:c1 [ether] on swp1

```

To add a static ARP cache entry:

```

cumulus@switch:~$ arp -s 11.0.2.2 00:02:00:00:00:10
cumulus@switch:~$ arp -a
? (11.0.2.2) at 00:02:00:00:00:10 [ether] PERM on swp3
? (11.0.3.2) at 00:02:00:00:00:01 [ether] on swp4
? (11.0.0.2) at 44:38:39:00:01:c1 [ether] on swp1

```

## ***Generating Traffic Using mz***

`mz` is a fast traffic generator. It can generate a large variety of packet types at high speed. See `man mz` for details.

For example, to send two sets of packets to TCP port 23 and 24, with source IP 11.0.0.1 and destination 11.0.0.2, do the following:

```
cumulus@switch:~$ sudo mz swp1 -A 11.0.0.1 -B 11.0.0.2 -c 2 -v -t tcp
"dp=23-24"

Mausezahn 0.40 - (C) 2007-2010 by Herbert Haas - http://www.perihel.at/sec
/mz/
Use at your own risk and responsibility!
-- Verbose mode --

This system supports a high resolution clock.
The clock resolution is 4000250 nanoseconds.
Mausezahn will send 4 frames...
IP: ver=4, len=40, tos=0, id=0, frag=0, ttl=255, proto=6, sum=0, SA=11.
0.0.1, DA=11.0.0.2,
payload=[see next layer]
TCP: sp=0, dp=23, S=42, A=42, flags=0, win=10000, len=20, sum=0,
payload=

IP: ver=4, len=40, tos=0, id=0, frag=0, ttl=255, proto=6, sum=0, SA=11.
0.0.1, DA=11.0.0.2,
payload=[see next layer]
TCP: sp=0, dp=24, S=42, A=42, flags=0, win=10000, len=20, sum=0,
payload=

IP: ver=4, len=40, tos=0, id=0, frag=0, ttl=255, proto=6, sum=0, SA=11.
0.0.1, DA=11.0.0.2,
payload=[see next layer]
TCP: sp=0, dp=23, S=42, A=42, flags=0, win=10000, len=20, sum=0,
payload=

IP: ver=4, len=40, tos=0, id=0, frag=0, ttl=255, proto=6, sum=0, SA=11.
0.0.1, DA=11.0.0.2,
payload=[see next layer]
TCP: sp=0, dp=24, S=42, A=42, flags=0, win=10000, len=20, sum=0,
payload=
```

## ***Creating Counter ACL Rules***

In Linux, all ACL rules are always counted. To create an ACL rule for counting purposes only, set the rule action to ACCEPT. See the [Netfilter \(see page 95\)](#) chapter for details on how to use `cl-acltool` to set up iptables-/ip6tables-/ebtables-based ACLs.



Always place your rules files under `/etc/cumulus/acl/policy.d/`.

To count all packets going to a Web server:

```
cumulus@switch:~$ cat sample_count.rules

[iptables]
-A FORWARD -p tcp --dport 80 -j ACCEPT

cumulus@switch:~$ sudo cl-acltool -i -p sample_count.rules
Using user provided rule file sample_count.rules
Reading rule file sample_count.rules ...
Processing rules in file sample_count.rules ...
Installing acl policy... done.

cumulus@switch:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 16 packets, 2224 bytes)
pkts bytes target     prot opt in      out      source
destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in      out      source
destination
      2   156 ACCEPT     tcp   --  any    any     anywhere
anywhere           tcp  dpt:http

Chain OUTPUT (policy ACCEPT 44 packets, 8624 bytes)
pkts bytes target     prot opt in      out      source
destination
```



The **-p** option clears out all other rules, and the **-i** option is used to reinstall all the rules.

## Configuring SPAN and ERSPAN

SPAN (Switched Port Analyzer) provides for the mirroring of all packets coming in from or going out of an interface (the *SPAN source*), and being copied and transmitted out of a local port (the *SPAN destination*) for monitoring. The SPAN destination port is also referred to as a mirror-to-port (MTP). The original packet is still switched, while a mirrored copy of the packet is sent out of the MTP.

ERSPAN (Encapsulated Remote SPAN) enables the mirrored packets to be sent to a monitoring node located anywhere across the routed network. The switch finds the outgoing port of the mirrored packets by doing a lookup of the destination IP address in its routing table. The original L2 packet is encapsulated with GRE for IP delivery. The encapsulated packets have the following format:

```
-----| MAC_HEADER | IP_HEADER | GRE_HEADER | L2_Mirrored_Packet |-----
```



Mirrored traffic is not guaranteed. If the MTP is congested, mirrored packets may be discarded.

SPAN and ERSPAN are configured via `cl-acltool`, the [same utility for security ACL configuration \(see page 95\)](#). The match criteria for SPAN and ERSPAN can only be an interface; more granular match terms are not supported. The SPAN source interface can be a port, a subinterface or a bond interface. Both ingress and egress traffic on interfaces can be matched.

Cumulus Linux supports a maximum of 2 SPAN destinations. Multiple rules (SPAN sources) can point to the same SPAN destination, although a given SPAN source cannot specify 2 SPAN destinations. The SPAN destination (MTP) interface can be a physical port, a subinterface, or a bond interface. The SPAN/ERSPAN action is independent of security ACL actions. If packets match both a security ACL rule and a SPAN rule, both actions will be carried out.



Always place your rules files under `/etc/cumulus/acl/policy.d/`.

## **Configuring SPAN for Switch Ports**

This section describes how to set up, install, verify and uninstall SPAN rules. In the examples that follow, you will span (mirror) switch port `swp4` input traffic and `swp4` output traffic to destination switch port `swp19`.

First, create a rules file in `/etc/cumulus/acl/policy.d/`:

```
cumulus@switch:~$ sudo bash -c 'cat <<EOF > /etc/cumulus/acl/policy.d/span.rules
[iptables]
-A FORWARD --in-interface swp4 -j SPAN --dport swp19
-A FORWARD --out-interface swp4 -j SPAN --dport swp19
EOF'
```



Using `cl-acltool` with the `--out-interface` rule applies to transit traffic only; it does not apply to traffic sourced from the switch.

Next, verify all the rules that are currently installed:

```
cumulus@switch:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
```

pkts	bytes	target	prot	opt	in	out	source
<b>destination</b>							
0	0	DROP	all	--	swp+	any	240.0.0.0/5
<b>anywhere</b>							
0	0	DROP	all	--	swp+	any	loopback/8
<b>anywhere</b>							
0	0	DROP	all	--	swp+	any	base-address.mcast.net/8
<b>anywhere</b>							
0	0	DROP	all	--	swp+	any	255.255.255.255
<b>anywhere</b>							
0	0	SETCLASS	ospf	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	ospf	--	any	any	anywhere
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	dpt:bgp	SETCLASS	class:7
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	dpt:bgp	POLICE	mode:pkt rate:2000 burst:2000
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	spt:bgp	SETCLASS	class:7
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	spt:bgp	POLICE	mode:pkt rate:2000 burst:2000
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	dpt:5342	SETCLASS	class:7
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	dpt:5342	POLICE	mode:pkt rate:2000 burst:2000
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	spt:5342	SETCLASS	class:7
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	spt:5342	POLICE	mode:pkt rate:2000 burst:2000
<b>anywhere</b>							
0	0	SETCLASS	icmp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	icmp	--	any	any	anywhere
<b>anywhere</b>							
15	5205	SETCLASS	udp	--	swp+	any	anywhere
<b>anywhere</b>							
11	3865	POLICE	udp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	udp	--	any	any	anywhere
<b>anywhere</b>							
0	0	SETCLASS	tcp	--	swp+	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	any	any	anywhere
<b>anywhere</b>							
0	0	POLICE	tcp	--	dpt:bootps	POLICE	mode:pkt rate:100 burst:100
<b>anywhere</b>							

```

anywhere          tcp dpt:bootpc POLICE mode:pkt rate:100 burst:100
    17 1088 SETCLASS igmp -- swp+ any      anywhere
anywhere          SETCLASS class:6
    17 1156 POLICE   igmp -- any     any      anywhere
anywhere          POLICE  mode:pkt rate:300 burst:100
    394 41060 POLICE  all   -- swp+ any      anywhere
anywhere          ADDRTYPE match dst-type LOCAL POLICE mode:pkt rate:
1000 burst:1000 class:0
    0      0 POLICE   all   -- swp+ any      anywhere
anywhere          ADDRTYPE match dst-type IPROUTER POLICE mode:pkt rate:
400 burst:100 class:0
    988 279K SETCLASS all   -- swp+ any      anywhere
anywhere          SETCLASS class:0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out      source
destination
    0      0 DROP       all   -- swp+ any      240.0.0.0/5
anywhere
    0      0 DROP       all   -- swp+ any      loopback/8
anywhere
    0      0 DROP       all   -- swp+ any      base-address.mcast.net/8
anywhere
    0      0 DROP       all   -- swp+ any      255.255.255.255
anywhere
26864 4672K SPAN      all   -- swp4  any      anywhere
anywhere          dport:swp19 <---- input packets on swp4

40722  47M SPAN      all   -- any     swp4  anywhere
anywhere          dport:swp19 <---- output packets on swp4

Chain OUTPUT (policy ACCEPT 67398 packets, 5757K bytes)
pkts bytes target      prot opt in      out      source
destination

```

Install the rules:

```

cumulus@switch:~$ sudo cl-acltool -i
[sudo] password for cumulus:
Reading rule file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Reading rule file /etc/cumulus/acl/policy.d/99control_plane_catch_all.rules

```

```
...
Processing rules in file /etc/cumulus/acl/policy.d
/99control_plane_catch_all.rules ...
Reading rule file /etc/cumulus/acl/policy.d/span.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/span.rules ...
Installing acl policy
done.
```



Running the following command is incorrect and will remove **all** existing control-plane rules or other installed rules and only install the rules defined in **span.rules**:

```
cumulus@switch:~$ sudo cl-acltool -i -P /etc/cumulus/acl/policy.d
/span.rules
```

Verify that the SPAN rules were installed:

```
cumulus@switch:~$ sudo cl-acltool -L all | grep SPAN
38025 7034K SPAN      all -- swp4    any     anywhere
anywhere          dport:swp19
50832   55M SPAN      all -- any     swp4    anywhere
anywhere          dport:swp19
```

## Configuring SPAN for Bonds

This section describes how to configure SPAN for all packets going out of **bond0** locally to **bond1**.

First, create a rules file in **/etc/cumulus/acl/policy.d/**:

```
cumulus@switch:~$ sudo bash -c 'cat <<EOF > /etc/cumulus/acl/policy.d
/span_bond.rules
[iptables]
-A FORWARD --out-interface bond0 -j SPAN --dport bond1
EOF'
```



Using **cl-acltool** with the **--out-interface** rule applies to transit traffic only; it does not apply to traffic sourced from the switch.

Install the rules:

```
cumulus@switch:~$ sudo cl-acltool -i
[sudo] password for cumulus:
Reading rule file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Reading rule file /etc/cumulus/acl/policy.d/99control_plane_catch_all.rules
...
Processing rules in file /etc/cumulus/acl/policy.d
/99control_plane_catch_all.rules ...
Reading rule file /etc/cumulus/acl/policy.d/span_bond.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/span_bond.rules ...
Installing acl policy
done.
```

Verify that the SPAN rules were installed:

```
cumulus@switch:~$ sudo iptables -L -v | grep SPAN
 19  1938 SPAN      all  --  any    bond0    anywhere
 anywhere          dport:bond1
```

## Configuring ERSPAN

This section describes how to configure ERSPAN for all packets coming in from `swp1` to `12.0.0.2`:

First, create a rules file in `/etc/cumulus/acl/policy.d/`:

```
cumulus@switch:~$ sudo bash -c 'cat <<EOF > /etc/cumulus/acl/policy.d
/erspan.rules
[iptables]
-A FORWARD --in-interface swp1 -j ERSPAN --src-ip 12.0.0.1 --dst-ip
12.0.0.2 --ttl 64
EOF'
```

Install the rules:

```
cumulus@switch:~$ sudo cl-acltool -i
Reading rule file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/00control_plane.rules ...
Reading rule file /etc/cumulus/acl/policy.d/99control_plane_catch_all.rules
...
Processing rules in file /etc/cumulus/acl/policy.d
```

```
/99control_plane_catch_all.rules ...
Reading rule file /etc/cumulus/acl/policy.d/erspan.rules ...
Processing rules in file /etc/cumulus/acl/policy.d/erspan.rules ...
Installing acl policy
done.
```

Verify that the ERSPAN rules were installed:

```
cumulus@switch:~$ sudo iptables -L -v | grep SPAN
 69  6804 ERSPAN      all  --  swp1    any     anywhere
anywhere          ERSPAN src-ip:12.0.0.1 dst-ip:12.0.0.2
```

The **src-ip** option can be any IP address, whether it exists in the routing table or not. The **dst-ip** option must be an IP address reachable via the routing table. The destination IP address must be reachable from a front-panel port, and not the management port. Use **ping** or **ip route get <ip>** to verify that the destination IP address is reachable. Setting the **--ttl1** option is recommended.



When using **Wireshark** to review the ERSPAN output, Wireshark may report the message "Unknown version, please report or test to use fake ERSPAN preference", and the trace is unreadable. To resolve this, go into the General preferences for Wireshark, then go to **Protocols** > **ERSPAN** and check the **Force to decode fake ERSPAN frame** option.

## Selective Spanning

SPAN/ERSPAN traffic rules can be configured to limit the traffic that is spanned, to reduce the volume of copied data.



Cumulus Linux 3.0 supports selective spanning for iptables only. ip6tables and ebtables are not supported.

The following matching fields are supported:

- IPv4 SIP/DIP
- IP protocol
- L4 (TCP/UDP) src/dst port
- TCP flags
- An ingress port/wildcard (swp+) can be specified in addition



Only two unique mirror targets are supported in a given rule set.

## SPAN Examples

- To mirror forwarded packets from all ports matching SIP 20.0.1.0 and DIP 20.0.1.2 to port swp1s1:

```
-A FORWARD --in-interface swp+ -s 20.0.0.2 -d 20.0.1.2 -j SPAN --
dport swp1s2
```

- To mirror icmp packets from all ports to swp1s2:

```
-A FORWARD --in-interface swp+ -s 20.0.0.2 -p icmp -j SPAN --
dport swp1s2
```

- To mirror forwarded UDP packets received from port swp1s0, towards DIP 20.0.1.2 and destination port 53:

```
-A FORWARD --in-interface swp1s0 -d 20.0.1.2 -p udp --dport 53 -
j SPAN --dport swp1s2
```

- To mirror all forwarded TCP packets with only SYN set:

```
-A FORWARD --in-interface swp+ -p tcp --tcp-flags ALL SYN -j
SPAN --dport swp1s2
```

- To mirror all forwarded TCP packets with only FIN set:

```
-A FORWARD --in-interface swp+ -p tcp --tcp-flags ALL FIN -j
SPAN --dport swp1s2
```

## ERSPAN Examples

- To mirror forwarded packets from all ports matching SIP 20.0.1.0 and DIP 20.0.1.2:

```
-A FORWARD --in-interface swp+ -s 20.0.0.2 -d 20.0.1.2 -j ERSPAN
--src-ip 90.0.0.1 --dst-ip 20.0.2.2
```

- To mirror ICMP packets from all ports:

```
-A FORWARD --in-interface swp+ -s 20.0.0.2 -p icmp -j ERSPAN --
src-ip 90.0.0.1 --dst-ip 20.0.2.2
```

- To mirror forwarded UDP packets received from port swp1s0, towards DIP 20.0.1.2 and destination port 53:

```
-A FORWARD --in-interface swp1s0 -d 20.0.1.2 -p udp --dport 53 -
j ERSPAN --src-ip 90.0.0.1 --dst-ip 20.0.2.2
```

- To mirror all forwarded TCP packets with only SYN set:

```
-A FORWARD --in-interface swp+ -p tcp --tcp-flags ALL SYN -j
ERSPAN --src-ip 90.0.0.1 --dst-ip 20.0.2.2
```

- To mirror all forwarded TCP packets with only FIN set:

```
-A FORWARD --in-interface swp+ -p tcp --tcp-flags ALL FIN -j
ERSPAN --src-ip 90.0.0.1 --dst-ip 20.0.2.2
```

## Removing SPAN Rules

To remove your SPAN rules, run:

```
#Remove rules file:
cumulus@switch:~$ sudo rm /etc/cumulus/acl/policy.d/span.rules
#Reload the default rules
cumulus@switch:~$ sudo cl-acltool -i
cumulus@switch:~$
```

To verify that the SPAN rules were removed:

```
cumulus@switch:~$ sudo cl-acltool -L all | grep SPAN
cumulus@switch:~$
```

## Monitoring Control Plane Traffic with `tcpdump`

You can use `tcpdump` to monitor control plane traffic — traffic sent to and coming from the switch CPUs. `tcpdump` does **not** monitor data plane traffic; use `cl-acltool` instead (see above).

For more information on `tcpdump`, read the [tcpdump documentation](#) and the [tcpdump man page](#).

The following example incorporates a few `tcpdump` options:

- `-i bond0`, which captures packets from bond0 to the CPU and from the CPU to bond0
- `host 169.254.0.2`, which filters for this IP address
- `-c 10`, which captures 10 packets then stops

```
cumulus@switch:~$ sudo tcpdump -i bond0 host 169.254.0.2 -c 10
tcpdump: WARNING: bond0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on bond0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:24:42.532473 IP 169.254.0.2 > 169.254.0.1: ICMP echo request, id 27785,
seq 6, length 64
16:24:42.532534 IP 169.254.0.1 > 169.254.0.2: ICMP echo reply, id 27785,
seq 6, length 64
16:24:42.804155 IP 169.254.0.2.40210 > 169.254.0.1.5342: Flags [.], seq
266275591:266277039, ack 3813627681, win 58, options [nop,nop,TS val
590400681 ecr 530346691], length 1448
16:24:42.804228 IP 169.254.0.1.5342 > 169.254.0.2.40210: Flags [.], ack
1448, win 166, options [nop,nop,TS val 530348721 ecr 590400681], length 0
16:24:42.804267 IP 169.254.0.2.40210 > 169.254.0.1.5342: Flags [P.], seq
1448:1836, ack 1, win 58, options [nop,nop,TS val 590400681 ecr 530346691],
length 388
16:24:42.804293 IP 169.254.0.1.5342 > 169.254.0.2.40210: Flags [.], ack
1836, win 165, options [nop,nop,TS val 530348721 ecr 590400681], length 0
16:24:43.532389 IP 169.254.0.2 > 169.254.0.1: ICMP echo request, id 27785,
seq 7, length 64
16:24:43.532447 IP 169.254.0.1 > 169.254.0.2: ICMP echo reply, id 27785,
seq 7, length 64
16:24:43.838652 IP 169.254.0.1.59951 > 169.254.0.2.5342: Flags [.], seq
2555144343:2555145791, ack 2067274882, win 58, options [nop,nop,TS val
530349755 ecr 590399688], length 1448
16:24:43.838692 IP 169.254.0.1.59951 > 169.254.0.2.5342: Flags [P.], seq
1448:1838, ack 1, win 58, options [nop,nop,TS val 530349755 ecr 590399688],
length 390
10 packets captured
12 packets received by filter
0 packets dropped by kernel
```

## Configuration Files

- /etc/cumulus/acl/policy.conf

## Useful Links

- [www.perihel.at/sec/mz/mzguide.html](http://www.perihel.at/sec/mz/mzguide.html)
- [en.wikipedia.org/wiki/Ping](http://en.wikipedia.org/wiki/Ping)
- [www.tcpdump.org](http://www.tcpdump.org)
- [en.wikipedia.org/wiki/Traceroute](http://en.wikipedia.org/wiki/Traceroute)

## Caveats and Errata

- SPAN rules cannot match outgoing subinterfaces.
- ERSPAN rules must include `ttl1` for versions 1.5.1 and earlier.

## Using `netshow` to Troubleshoot Your Network Configuration

`netshow` is a tool in Cumulus Linux that quickly returns a lot of information about your network configuration. It's a tool designed by network operators for network troubleshooters since existing command line tools have too many options. `netshow` addresses this by leveraging the network troubleshooting experience from a wide group of troubleshooters and boiling it down to just a few important options. `netshow` quickly aggregates basic network information on Linux devices with numerous interfaces. `netshow` intelligently informs the administrator what network type an interface belongs to, and shows the most relevant information to a network administrator.

`netshow` can be used on any distribution of Linux, not just Cumulus Linux.

## Installing `netshow`

Starting with Cumulus Linux 3.0.0, `netshow` is installed by default in Cumulus Linux.

## Installing `netshow` on a Linux Server or in OpenStack

To install `netshow` on a Linux server, run:

```
pip install netshow-linux-lib
```



Debian and Red Hat packages will be available in the near future.

## Using `netshow`

Running `netshow` with no arguments displays all available command line arguments usable by `netshow`. (Running `netshow --help` gives you the same information.) The output looks like this:

```
cumulus@leaf1$ netshow
```

**Usage:**

```
netshow system [--json | -j ]
netshow counters [errors] [all] [--json | -j | -l | --legend ]
netshow lldp [--json | -j | -l | --legend ]
netshow interface [<iface>] [all] [--mac | -m ] [--oneline | -1 | --
json | -j | -l | --legend ]
netshow access [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l
| --legend ]
netshow bridges [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l
| --legend ]
netshow bonds [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l |
--legend ]
netshow bondmems [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l
| --legend ]
netshow mgmt [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l |
--legend ]
netshow 12 [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l | --
legend ]
netshow 13 [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l | --
legend ]
netshow trunks [all] [--mac | -m ] [--oneline | -1 | --json | -j | -l
| --legend ]
netshow (--version | -V)
```

**Help:**

*	default is to show interfaces only in the UP state.
counters	summary of physical port counters.
interface	summary info of all interfaces
access	summary of physical ports with 12 or 13 config
bonds	summary of bonds
bondmems	summary of bond members
bridges	summary of ports with bridge members
mgmt	summary of mgmt ports
13	summary of ports with an IP.
12	summary of access, trunk and bridge interfaces
phy	summary of physical ports
trunks	summary of trunk interfaces
lldp	physical device neighbor information
interface <iface>	list summary of a single interface
system	system information

**Options:**

all	show all ports include those are down or admin down
--mac	show interface MAC in output

```
--version netshow software version
--oneline output each entry on one line
-1 alias for --oneline
--json print output in json
-l alias for --legend
--legend print legend key explaining abbreviations

cumulus@leaf1$
```

A Linux administrator can quickly see the few options available with the tool. One core tenet of **netshow** is for it to have a small number of command options. **netshow** is not designed to solve your network problem, but to help answer this simple question: "What is the basic network setup of my Linux device?" By helping to answer that question, a Linux administrator can spend more time troubleshooting the specific network problem instead of spending most of their time understanding the basic network state.

Originally developed for Cumulus Linux, **netshow** works on Debian-based servers and switches and Red Hat-based Linux systems.

**netshow** is designed by network operators, which has rarely occurred in the networking industry, where most command troubleshooting tools are designed by developers and are most useful in the network application development process.

## Showing Interfaces

To show all available interfaces that are physically UP, run **netshow interface**:

```
cumulus@leaf1$ netshow interface
-----
To view the legend, rerun "netshow" cmd with the "--legend" option
-----
      Name    Speed     MTU   Mode   Summary
---  -----  -----  -----  -----
UP   eth0     1G      1500  Mgmt   IP: 192.168.0.12/24(DHCP)
UP   lo       N/A     16436  Mgmt   IP: 127.0.0.1/8, ::1/128
cumulus@leaf1$
```

Whereas **netshow interface all** displays every interface regardless of state:

```
cumulus@leaf1$ netshow interface all
      Name    Speed     Mtu   Mode   Summary
---  -----  -----  -----  -----
UP   lo       N/A     16436  Loopback  IP: 127.0.0.1/8, ::1/128
UP   eth0     1G      1500  Mgmt   IP: 192.168.0.11/24 (DHCP)
ADMDN swp1s0  10G(4x10) 1500  Unknwn
ADMDN swp1s1  10G(4x10) 1500  Unknwn
```

ADMDN	swp1s2	10G(4x10)	1500	Unknwn
ADMDN	swp1s3	10G(4x10)	1500	Unknwn
ADMDN	swp2	40G(QSFP)	1500	Unknwn
ADMDN	swp3	40G(QSFP)	1500	Unknwn
ADMDN	swp4	40G(QSFP)	1500	Unknwn
ADMDN	swp5	40G(QSFP)	1500	Unknwn
ADMDN	swp6	40G(QSFP)	1500	Unknwn
ADMDN	swp7	40G(QSFP)	1500	Unknwn
ADMDN	swp8	40G(QSFP)	1500	Unknwn
ADMDN	swp9	40G(QSFP)	1500	Unknwn
ADMDN	swp10	40G(QSFP)	1500	Unknwn
ADMDN	swp11	40G(QSFP)	1500	Unknwn
ADMDN	swp12	40G(QSFP)	1500	Unknwn
ADMDN	swp13	40G(QSFP)	1500	Unknwn
ADMDN	swp14	40G(QSFP)	1500	Unknwn
ADMDN	swp15	40G(QSFP)	1500	Unknwn
ADMDN	swp16	40G(QSFP)	1500	Unknwn
ADMDN	swp17	40G(QSFP)	1500	Unknwn
ADMDN	swp18	40G(QSFP)	1500	Unknwn
ADMDN	swp19	40G(QSFP)	1500	Unknwn
ADMDN	swp20	40G(QSFP)	1500	Unknwn
ADMDN	swp21	40G(QSFP)	1500	Unknwn
ADMDN	swp22	40G(QSFP)	1500	Unknwn
ADMDN	swp23	40G(QSFP)	1500	Unknwn
ADMDN	swp24	40G(QSFP)	1500	Unknwn
ADMDN	swp25	40G(QSFP)	1500	Unknwn
ADMDN	swp26	40G(QSFP)	1500	Unknwn
ADMDN	swp27	40G(QSFP)	1500	Unknwn
ADMDN	swp28	40G(QSFP)	1500	Unknwn
ADMDN	swp29	40G(QSFP)	1500	Unknwn
ADMDN	swp30	40G(QSFP)	1500	Unknwn
ADMDN	swp31	40G(QSFP)	1500	Unknwn
ADMDN	swp32s0	10G(4x10)	1500	Unknwn
ADMDN	swp32s1	10G(4x10)	1500	Unknwn
ADMDN	swp32s2	10G(4x10)	1500	Unknwn
ADMDN	swp32s3	10G(4x10)	1500	Unknwn

You can get information about the switch itself by running `netshow system`:

```
cumulus@leaf1$ netshow system

QCT T3048-LY8
Cumulus Version 3.0.0~1462473422.02602ac
```

```
Build: Cumulus Linux 3.0.0~1462473422.02602ac
```

```
Chipset: Broadcom Trident2 BCM56854
```

```
Port Config: 48 x 10G-SFP+ & 4 x 40G-QSFP+
```

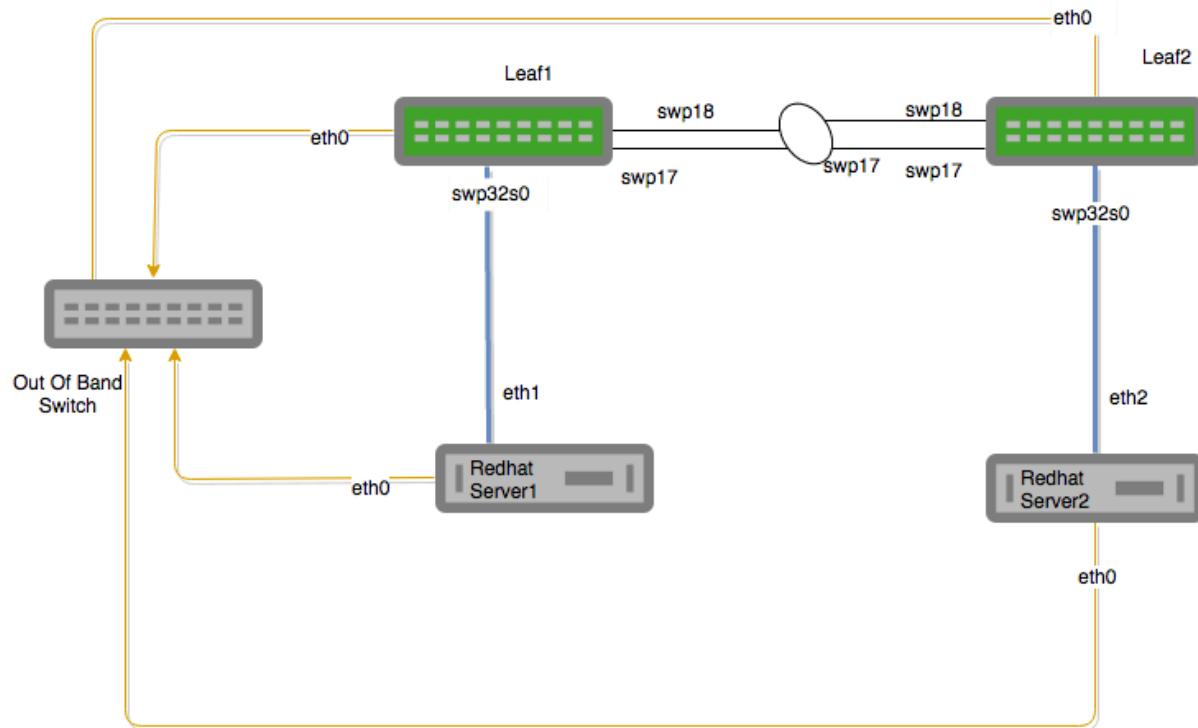
```
CPU: (x86_64) Intel Atom C2758 2.40GHz
```

```
Uptime: 2 days, 21:31:00
```

```
cumulus@leaf1$
```

## Troubleshooting Example: OpenStack

Looking at an OpenStack Environment, here is the physical diagram:



For server2, **netshow** can help us see the OpenStack network configuration. The **netshow** output below shows a summary of a Kilo-based OpenStack server running 3 tenants.

```
[root@server2 ~]# netshow int
-----
To view the legend, rerun "netshow" cmd with the "--legend" option
-----
      Name      Speed      MTU      Mode      Summary
-----
```

---					
UP	Interface Name	MTU	Link Layer Type	Bridge Membership	
UP	brq0b6f10c7-42	N/A	1500	Bridge/L2	802.1q Tag: 141 STP: Disabled Untagged Members:
	<b>tap079cf993-c7</b>				Tagged Members: eth1.141
UP	brq8cdc0589-9b	N/A	1500	Bridge/L2	802.1q Tag: 155 STP: Disabled Untagged Members:
	<b>tap5353b20a-68</b>				Tagged Members: eth1.155
UP	brq8ff99102-29	N/A	1500	Bridge/L2	802.1q Tag: 168 STP: Disabled Untagged Members:
	<b>tapfc2203e4-5b</b>				Tagged Members: eth1.168
UP	eth0	N/A	1500	Interface/L3	IP: 192.168.0.105/24
UP	eth1	N/A	1500	IntTypeUnknown	
UP	eth1	N/A	1500	Trunk/L2	Bridge Membership: Tagged: brq0b6f10c7-42
	(141), brq8cdc0589-9b(155), brq8ff99102-29(168)				
UP	lo	N/A	65536	Loopback	IP: 127.0.0.1/8, ::1/128
UP	tap079cf993-c7	10M	1500	Access/L2	Untagged: brq0b6f10c7-42
UP	tap5353b20a-68	10M	1500	Access/L2	Untagged: brq8cdc0589-9b
UP	tapfc2203e4-5b	10M	1500	Access/L2	Untagged: brq8ff99102-29

OpenStack interface numbering is not the easiest read, but here `netshow` can quickly show you:

- A list of all the interfaces in admin UP state and carrier UP state
- 3 bridges
- That STP is disabled for all the bridges
- An uplink trunk interface with 3 VLANs configured on it
- Many tap interfaces, most likely the virtual machines

This output took about 5 seconds to get and another 1 minute to analyze. To get this same level of understanding using traditional tools such as:

- `ip link show`
- `brctl show`
- `ip addr show`

... could take about 10 minutes. This is a significant improvement in productivity!

`netshow` uses a plugin architecture and can be easily expanded. An OpenStack interface discovery module is currently in development. If `netshow` is run on a hypervisor with OpenStack Keystone login environment variables like `OS_TENANT_NAME`, `netshow` should show the above output with a better interface discovery state, where `netshow` collects from OpenStack information from `libvirt`, `nova` and `neutron` to overlay the virtual machine and tenant subnet information over the interface kernel state information.

Interface discovery is one of the most powerful features of `netshow`. The ability to expand its interface discovery capabilities further simplifies understanding basic network troubleshooting, making the Linux administrator more productive and improving time to resolution while investigating network problems.

## **Other Useful netshow Features**

`netshow` uses the `python network-docopt` package. This is inspired by `docopt` and provides the ability to specify partial commands, without tab completion and running the complete option. For example:

```
netshow int runs netshow interface  
netshow sys runs netshow system
```

`netshow` will eventually support interface name autocomplete. In the near future, if you run `netshow int tap123` and there is only one interface starting with `tap123`, `netshow` will autocomplete the command option with the full interface.

## **Contributions Welcome!**

`netshow` is an open source project licensed under GPLv2. To contribute please contact Cumulus Networks through the [Cumulus Community Forum](#) or the [Netshow Linux Provider Github Repository Home](#). You can find developer documentation at [netshow.readthedocs.org](#). The documentation is still under development.

## **Monitoring System Statistics and Network Traffic with sFlow**

sFlow is a monitoring protocol that samples network packets, application operations, and system counters. sFlow collects both interface counters and sampled 5-tuple packet information, enabling you to monitor your network traffic as well as your switch state and performance metrics. An outside server, known as an *sFlow collector*, is required to collect and analyze this data.

`hsflowd` is the daemon that samples and sends sFlow data to configured collectors. `hsflowd` is not included in the base Cumulus Linux installation. After installation, `hsflowd` will automatically start when the switch boots up.



sFlow is not supported on switches with Spectrum ASICs.

## **Contents**

(Click to expand)

- [Contents \(see page 564\)](#)
- [Installing hsflowd \(see page 565\)](#)
- [Configuring sFlow \(see page 565\)
  - \[Configuring sFlow via DNS-SD \\(see page 565\\)\]\(#\)
  - \[Manually Configuring /etc/hsflowd.conf \\(see page 566\\)\]\(#\)](#)

- Configuring sFlow Visualization Tools (see page 567)
- Configuration Files (see page 567)
- Useful Links (see page 567)

## **Installing hsflowd**

To download and install the `hsflowd` package, use `apt-get`:

```
cumulus@switch:~$ sudo apt-get update
cumulus@switch:~$ sudo apt-get install -y hsflowd
```

## **Configuring sFlow**

You can configure `hsflowd` to send to the designated collectors via two methods:

- DNS service discovery (DNS-SD)
- Manually configuring `/etc/hsflowd.conf`

## **Configuring sFlow via DNS-SD**

With this method, you need to configure your DNS zone to advertise the collectors and polling information to all interested clients. Add the following content to the zone file on your DNS server:

```
_sflow._udp SRV 0 0 6343 collector1
_sflow._udp SRV 0 0 6344 collector2
_sflow._udp TXT (
"txtvers=1"
"sampling.1G=2048"
"sampling.10G=4096"
"sampling.40G=8192"
"polling=20"
)
```

The above snippet instructs `hsflowd` to send sFlow data to collector1 on port 6343 and to collector2 on port 6344. `hsflowd` will poll counters every 20 seconds and sample 1 out of every 2048 packets.



From Cumulus Linux 2.5.3 onwards, the sampling rate is limited to 16k pps. This can be configured by editing the `/etc/cumulus/datapath/traffic.conf` file, as shown below:

```
# Set sflow/sample ingress cpu packet rate and burst in packets/sec
# Values: {0..16384}
#sflow.rate = 16384
#sflow.burst = 16384
```

After the initial configuration is ready, bring up the sFlow daemon by running:

```
cumulus@switch:~$ sudo systemctl start hsflowd.service
```

No additional configuration is required in `/etc/hsflowd.conf`.

## **Manually Configuring `/etc/hsflowd.conf`**

With this method you will set up the collectors and variables on each switch.

Edit `/etc/hsflowd.conf` and change `DNSSD = on` to `DNSSD = off`:

```
DNSSD = off
```

Then set up your collectors and sampling rates in `/etc/hsflowd.conf`:

```
# Manual Configuration (requires DNSSD=off above)
#####
#
# Typical configuration is to send every 30 seconds
polling = 20

sampling.1G=2048
sampling.10G=4096
sampling.40G=8192

collector {
    ip = 192.0.2.100
    udpport = 6343
}

collector {
    ip = 192.0.2.200
    udpport = 6344
}
```

This configuration polls the counters every 20 seconds, samples 1 of every 2048 packets and sends this information to a collector at 192.0.2.100 on port 6343 and to another collector at 192.0.2.200 on port 6344.



Some collectors require each source to transmit on a different port, others may listen on only one port. Please refer to the documentation for your collector for more information.

## Configuring sFlow Visualization Tools

For information on configuring various sFlow visualization tools, read this [Help Center article](#).

## Configuration Files

- /etc/hsflowd.conf

## Useful Links

- [sFlow Collectors](#)
- [sFlow Wikipedia page](#)

## SNMP Monitoring

Cumulus Linux utilizes the open source Net-SNMP agent `snmpd`, v5.7.3, which provides support for most of the common industry-wide MIBs, including interface counters and TCP/UDP IP stack data.



Cumulus Linux does not prevent customers from extending SNMP features. However, Cumulus Networks encourages the use of higher performance monitoring environments, rather than SNMP.

## Contents

This chapter covers ...

- Introduction to SNMP (Simple Network Management Protocol) (see page 568)
- Configuring Ports for SNMP to Listen for Requests (see page 568)
- Starting the SNMP Daemon (see page 568)
- Configuring SNMP (see page 569)
  - Setting up the Custom Cumulus Networks MIBs (see page 569)
  - Enabling the .1.3.6.1.2.1 Range (see page 570)
  - Enabling Public Community (see page 571)
  - Configuring SNMPv3 (see page 571)
  - Cumulus Linux Configuration (see page 574)
  - Nutanix Configuration (see page 575)
- Switch Information Displayed on Nutanix Prism (see page 578)
- Troubleshooting (see page 579)
- Enabling LLDP/CDP on VMware ESXi (Hypervisor on Nutanix) (see page 579)

- Enabling LLDP/CDP on Nutanix Acropolis (Hypervisor on Nutanix Acropolis) (see page 581)
- snmpwalk the Switch from Another Linux Device (see page 581)
- Troubleshooting Connections without LLDP or CDP (see page 583)
- SNMP Traps (see page 585)
  - snmptrapd.conf (see page 585)
  - Generating Event Notification Traps (see page 586)
- Supported MIBs (see page 590)

## ***Introduction to SNMP (Simple Network Management Protocol)***

SNMP is an IETF standards-based network management architecture and protocol that traces its roots back to Carnegie-Mellon University in 1982. Since then, it's been modified by programmers at the University of California. In 1995, this code was also made publicly available as the UCD project. After that, `ucd-snmp` was extended by work done at the University of Liverpool as well as later in Denmark. In late 2000, the project name changed to net-snmp and became a fully-fledged collaborative open source project. The version used by Cumulus Networks is base on the latest `net-snmp` 5.7.3 branch with added custom MIBs and pass through and pass persist scripts.

## ***Configuring Ports for SNMP to Listen for Requests***

For security reasons, the default port binding for `snmpd` is the loopback local address; consequently by default, the SNMP service does not listen for SNMP requests from outside the switch. In order to listen to requests from outside the switch, you need to change this binding to a specific IP address (or all interfaces) after configuring security access (community strings, users, and so forth). This is a change from older versions of Cumulus Linux (before version 3.0), which listened to incoming requests on all interfaces by default. The `snmpd` configuration file is `/etc/snmp/snmpd.conf` and should be modified before enabling and starting `snmpd`. The default configuration has no access community strings defined so `snmpd` will not respond to any SNMP requests until this is added.

## ***Starting the SNMP Daemon***

The following procedure is the recommended process to start `snmpd` and monitor it using `systemctl`.

To start the SNMP daemon:

1. Start the `snmpd` daemon:

```
cumulus@switch:~$ sudo systemctl start snmpd.service
```

2. Configure the `snmpd` daemon to start automatically after reboot:

```
cumulus@switch:~$ sudo systemctl enable snmpd.service
```

3. To enable `snmpd` to restart automatically after failure:
  - a. Create a file called `/etc/systemd/system/snmpd.service.d/restart.conf`.

- b. Add the following lines:

```
[Service]
Restart=always
RestartSec=60
```

- c. Run `systemctl daemon-reload`.

Once the service is started, SNMP can be used to manage various components on the Cumulus Linux switch.

## **Configuring SNMP**

Cumulus Linux ships with a production usable default `snmpd.conf` file included. This section covers a few basic configuration options in `snmpd.conf`. For more information regarding further configuring this file, refer to the `snmpd.conf` man page.

**!** The default `snmpd.conf` file does not include all supported MIBs or OIDs that can be exposed.

**!** Customers must at least change the default community string for v1 or v2c environments or the `snmpd` daemon will not respond to any requests.

## **Setting up the Custom Cumulus Networks MIBs**

**!** No changes are required in the `/etc/snmp/snmpd.conf` file on the switch, in order to support the custom Cumulus Networks MIBs. The following lines are already included by default:

```
view systemonly included .1.3.6.1.4.1.40310.1
view systemonly included .1.3.6.1.4.1.40310.2
sysObjectID 1.3.6.1.4.1.40310
pass_persist .1.3.6.1.4.1.40310.1 /usr/share/snmp/resq_pp.py
pass_persist .1.3.6.1.4.1.40310.2 /usr/share/snmp/cl_drop_cntrs_pp.py
```

However, several files need to be copied to the server, in order for the custom Cumulus MIB to be recognized on the destination NMS server.

- `/usr/share/snmp/mibs/Cumulus-Snmp-MIB.txt`
- `/usr/share/snmp/mibs/Cumulus-Counters-MIB.txt`
- `/usr/share/snmp/mibs/Cumulus-Resource-Query-MIB.txt`

## Enabling the .1.3.6.1.2.1 Range

Some MIBs, including storage information, are not included by default in `snmpd.conf` in Cumulus Linux. This results in some default views on common network tools (like `librenms`) to return less than optimal data. More MIBs can be included, by enabling all the .1.3.6.1.2.1 range. This simplifies the configuration file, removing concern that any required MIBs will be missed by the monitoring system. Various new MIBs were added for 3.0 and include the following: ENTITY and ENTITY-SENSOR MIB and parts of the BRIDGE-MIB and Q-BRIDGE-MIBs. These are included in the default configuration (Note: the view of the BRIDGE-MIB and Q-BRIDGE-MIB are commented out).



This configuration grants access to a large number of MIBs, including all MIB2 MIBs, which could reveal more data than expected. In addition to being a security vulnerability, it could consume more CPU resources.

To enable the .1.3.6.1.2.1 range:

1. Open `/etc/snmp/snmpd.conf` in a text editor.
2. Make sure the following lines are included in the configuration:

```
#####
#####
#
# ACCESS CONTROL
#
#
# system
view systemonly included .1.3.6.1.2.1
# quagga ospf6
view systemonly included .1.3.6.1.3.102
# lldpd (Note: lldpd must be restarted with the -x option
```

```
#     configured in order to send info to snmpd via Agent X
```

```
view systemonly included .1.0.8802.1.1.2
# Cumulus specific
view systemonly included .1.3.6.1.4.1.40310.1
view systemonly included .1.3.6.1.4.1.40310.2
```

3. Restart `snmpd`:

```
# sudo systemctl start snmpd.service
```

## Enabling Public Community

The `snmpd` authentication for versions 1 and 2 is disabled by default in Cumulus Linux. This password (called a community string) can be enabled by setting **rocommunity** (for read-only access) or **rwcommunity** (for read-write access). To enable read-only querying by a client:

1. Open `/etc/snmp/snmpd.conf` in a text editor.
2. To allow read-only access using a password `public` from any client IP address (*default*) for the view you defined before with `systemonly`, add the following line to the end of the file, then save it:

```
rocommunity public default -V systemonly
```

3. Restart `snmpd`:

```
cumulus@switch:~$ sudo systemctl restart snmpd.service
```

## Configuring SNMPv3

Since community strings in versions 1 and 2c are sent in the clear, SNMPv3 is often used to enable authentication and encryption. SNMPv3 was first released around 2000. A minimal example is shown here for `/etc/snmp/snmpd.conf` that defines three users, each with a different combination of authentication and encryption. Please change these usernames and passwords before using this in a network:



Make sure you change the usernames and passwords in the sample code below, as the ones used here are for explanatory purposes only.

```
# simple no auth user
createUser user1

# user with MD5 authentication
createUser user2 MD5 user2password

# user with MD5 for auth and DES for encryption
createUser user3 MD5 user3password DES user3encryption

# user999 with MD5 for authentication and DES for encryption

createUser user666 SHA user666password AES user666encryption
createUser user999 MD5 user999password DES user999encryption

# restrict users to certain OIDs
# (Note: creating rouser or rwuser will give
```

```
# access regardless of the createUser command above. However,
# createUser without rouser or rwuser will not provide any access).
rouser user1 noauth 1.3.6.1.2.1.1
rouser user2 auth 1.3.6.1.2.1
rwuser user3 priv 1.3.6.1.2.1
rwuser user666
rwuser user999
```

Once you make this configuration and restart the `snmpd` daemon, the user access can be checked with a client — the Debian package called `snmp` contains `snmpget` and `snmpwalk`, as well as other programs that are useful for checking daemon functionality from the switch itself or from another workstation. The following commands check the access for each user defined above from the localhost (the switch itself):

```
# check user1 which has no authentication or encryption (NoauthNoPriv)
snmpget -v 3 -u user1 -l NoauthNoPriv localhost 1.3.6.1.2.1.1.1.0
snmpwalk -v 3 -u user1 -l NoauthNoPriv localhost 1.3.6.1.2.1.1

# check user2 which has authentication but no encryption (authNoPriv)
snmpget -v 3 -u user2 -l authNoPriv -a MD5 -A user2password localhost
1.3.6.1.2.1.1.1.0
snmpget -v 3 -u user2 -l authNoPriv -a MD5 -A user2password localhost
1.3.6.1.2.1.2.1.0
snmpwalk -v 3 -u user2 -l authNoPriv -a MD5 -A user2password localhost
1.3.6.1.2.1

# check user3 which has both authentication and encryption (authPriv)
snmpget -v 3 -u user3 -l authPriv -a MD5 -A user3password -x DES -X
user3encryption localhost .1.3.6.1.2.1.1.1.0
snmpwalk -v 3 -u user3 -l authPriv -a MD5 -A user3password -x DES -X
user3encryption localhost .1.3.6.1.2.1
```

```
snmpwalk -v 3 -u user666 -l authPriv -a SHA -x AES -A user666password -X
user666encryption localhost 1.3.6.1.2.1.1
snmpwalk -v 3 -u user999 -l authPriv -a MD5 -x DES -A user999password -X
user999encryption localhost 1.3.6.1.2.1.1
```

A slightly more secure method of configuring SNMPv3 users without creating cleartext passwords is the following:

1. Install the `net-snmp-config` script that is in `libsntp-dev` package:

```
cumulus@switch:~$ sudo apt-get update
cumulus@switch:~$ sudo apt-get install libsnmp-dev
```

2. Stop the daemon:

```
cumulus@switch:~$ sudo systemctl stop snmpd.service
```

3. Use the `net-snmp-config` command to create two users, one with MD5 and DES, and the next with SHA and AES.



The minimum password length is 8 characters and the arguments `-a` and `-x` to `net-snmp-config` have different meanings than they do for `snmpwalk`.

```
cumulus@switch:~$ sudo net-snmp-config --create-snmpv3-user -a
md5authpass -x desprivpass -A MD5 -X DES userMD5withDES
cumulus@switch:~$ sudo net-snmp-config --create-snmpv3-user -a
shaauthpass -x aesprivpass -A SHA -X AES userSHAwithAES
cumulus@switch:~$ sudo systemctl start snmpd.service
```

This adds a `createUser` command in `/var/lib/snmp/snmpd.conf`. Do **not** edit this file by hand, unless you are removing usernames. It also adds the `rwuser` in `/usr/share/snmp/snmpd.conf`. You may want to edit this file and restrict access to certain parts of the MIB by adding `noauth`, `auth` or `priv` to allow unauthenticated access, require authentication or to enforce use of encryption, respectively.

The `snmpd` daemon reads the information from the `/var/lib/snmp/snmpd.conf` file and then the line is removed (eliminating the storage of the master password for that user) and replaced with the key that is derived from it (using the EngineID). This key is a localized key, so that if it is stolen it cannot be used to access other agents. To remove the two users `userMD5withDES` and `userSHAwithAES`, you need simply stop the `snmpd` daemon and edit the files `/var/lib/snmp/snmpd.conf` and `/usr/share/snmp/snmpd.conf`. Simply remove the lines containing the username. Then restart the `snmpd` daemon as in step 3 above.

From a client, you would access the MIB with the correct credentials. (Again, note that the roles of `-x`, `-a` and `-X` and `-A` are reversed on the client side as compared with the `net-snmp-config` command used above.)

```
snmpwalk -v 3 -u userMD5withDES -l authPriv -a MD5 -x DES -A md5authpass -X
desprivpass localhost 1.3.6.1.2.1.1.1
snmpwalk -v 3 -u userSHAwithAES -l authPriv -a SHA -x AES -A shaauthpass -X
aesprivpass localhost 1.3.6.1.2.1.1.1
```

## Configuring Nutanix Prism

Nutanix Prism is a graphical user interface (GUI) for managing infrastructures and virtual environments.

## Cumulus Linux Configuration

1. SSH to the Cumulus Linux switch that needs to be configured, replacing [switch] below as appropriate:

```
cumulus@workbench:~$ ssh cumulus@[switch]
```

2. Confirm the switch is running Cumulus Linux 2.5.5 or newer:

```
cumulus@switch$ cat /etc/lsb-release
DISTRIB_ID="Cumulus Linux"
DISTRIB_RELEASE=2.5.5
DISTRIB_DESCRIPTION=2.5.5-4cd66d9-201512071809-build
```

3. Open the `/etc/snmp/snmpd.conf` file in an editor.

4. Uncomment the following 3 lines in the `/etc/snmp/snmpd.conf` file, and save the file:

- `bridge_pp.py`

```
pass_persist .1.3.6.1.2.1.17 /usr/share/snmp/bridge_pp.py
```

- `Community`

```
rocommunity public default -V systemonly
```

- Line directly below the Q-BRIDGE-MIB (1.3.6.1.2.1.17)

```
# BRIDGE-MIB and Q-BRIDGE-MIB tables
view systemonly included .1.3.6.1.2.1.17
```

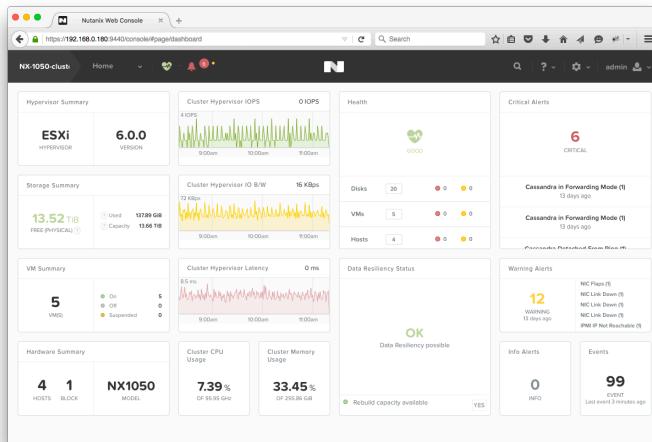
5. Restart `snmpd`:

```
cumulus@switch$ sudo systemctl restart snmpd.service
```

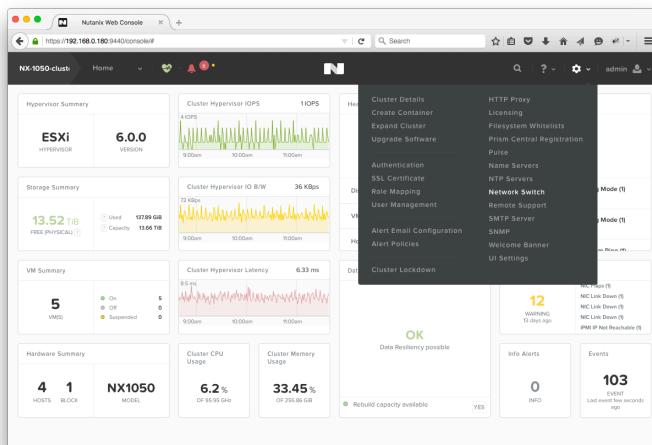
```
Restarting network management services: snmpd.
cumulus@switch$
```

## Nutanix Configuration

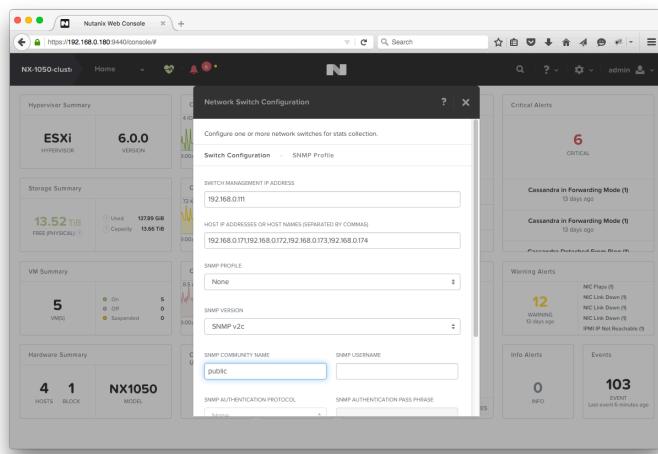
1. Log into the Nutanix Prism. Nutanix defaults to the Home menu, referred to as the Dashboard:



2. Click on the gear icon  in the top right corner of the dashboard, and select NetworkSwitch:



3. Click the **+Add Switch Configuration** button in the **Network Switch Configuration** pop up window.
4. Fill out the **Network Switch Configuration** for the Top of Rack (ToR) switch configured for snmpd in the previous section:



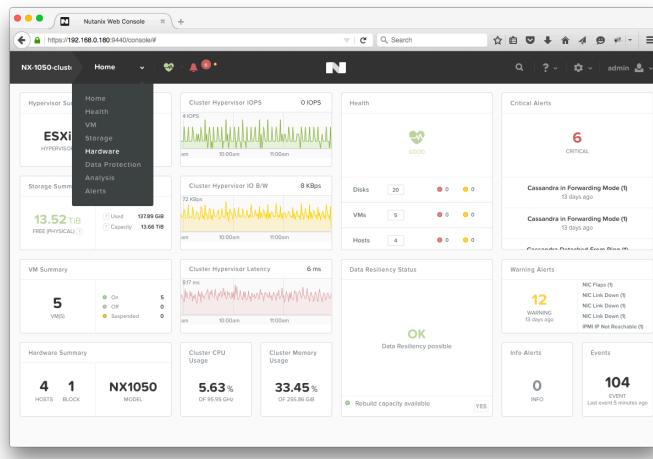
Configuration Parameter	Description	Value Used in Example
Switch Management IP Address	This can be any IP address on the box. In the screenshot above, the eth0 management IP is used.	192.168.0.111
Host IP Addresses or Host Names	IP addresses of Nutanix hosts connected to that particular ToR switch.	192.168.0.171,192.168.0.172,192.168.0.173,192.168.0.174
SNMP Profile	Saved profiles, for easy configuration when hooking up to multiple switches.	None
SNMP Version	SNMP v2c or SNMP v3. Cumulus Linux has only been tested with SNMP v2c for Nutanix integration.	SNMP v2c
SNMP Community Name		public

Configuration Parameter	Description	Value Used in Example
	SNMP v2c uses communities to share MIBs. The default community for snmpd is 'public'.	



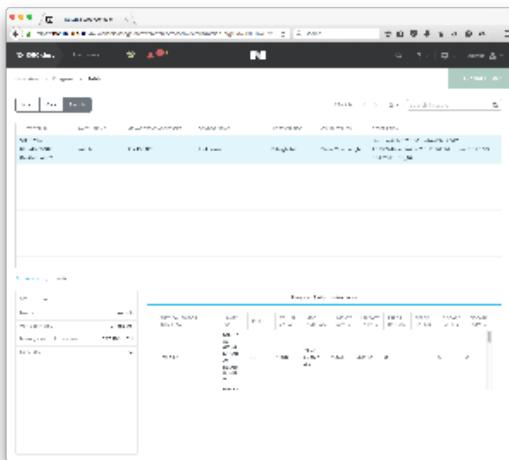
The rest of the values were not touched for this demonstration. They are usually used with SNMP v3.

- Save the configuration. The switch will now be present in the **Network Switch Configuration** menu now.
- Close the pop up window to return to the dashboard.
- Open the **Hardware** option from the **Home** dropdown menu:



- Click the **Table** button.

9. Click the **Switch** button. Configured switches are shown in the table, as indicated in the screenshot below, and can be selected in order to view interface statistics:



The switch has been added correctly, when interfaces hooked up to the Nutanix hosts are visible.

## **Switch Information Displayed on Nutanix Prism**

- Physical Interface (e.g. swp1, swp2). This will only display swp interfaces connected to Nutanix hosts by default.
- Switch ID - Unique identifier that Nutanix keeps track of each port ID (see below)
- Index - interface index, in the above demonstration swp49 maps to Index 52 because there is a loopback and two ethernet interface before the swp starts.
- MTU of interface
- MAC Address of Interface
- Unicast RX Packets (Received)
- Unicast TX Packets (Transmitted)
- Error RX Packets (Received)
- Error TX Packets (Transmitted)
- Discard RX Packets (Received)
- Discard TX Packets (Transmitted)

The Nutanix appliance will use Switch IDs that can also be viewed on the Prism CLI (by SSHing to the box). To view information from the Nutanix CLI, login using the default username **nutanix**, and the password **nutanix/4u**.

```
nutanix@NTNX-14SM15270093-D-CVM:192.168.0.184:~$ ncli network list-switch
  Switch ID          : 00051a76-f711-89b6-0000-000000003bac:::
5f13678e-6ffd-4b33-912f-f1aa6e8da982
  Name          : switch
  Switch Management Address : 192.168.0.111
```

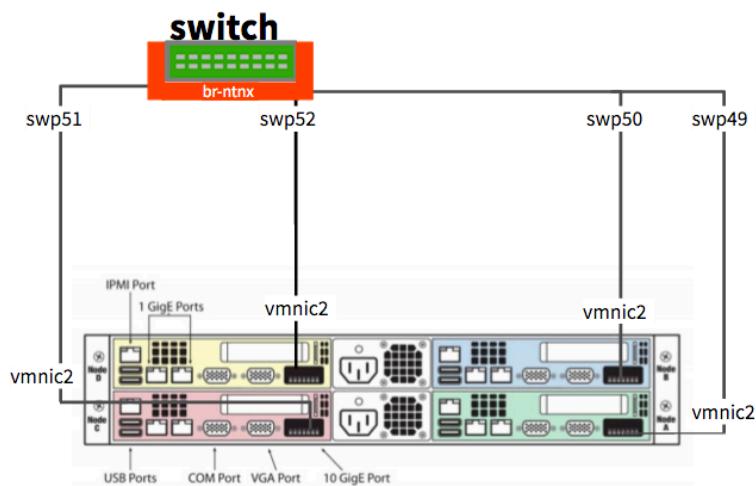
```

Description : Linux switch 3.2.65-1+deb7u2+c12.5+2 #3.
2.65-1+deb7u2+c12.5+2 SMP Mon Jun 1 18:26:59 PDT 2015 x86_64
Object ID   : enterprises.40310
Contact Information : Admin <admin@company.com>
Location Information : Raleigh, NC
Services     : 72
Switch Vendor Name : Unknown
Port IDs      : 00051a76-f711-89b6-0000-00000003bac:::
5f13678e-6ffd-4b33-912f-f1aa6e8da982:52, 00051a76-f711-89b6-0000-
00000003bac:::5f13678e-6ffd-4b33-912f-f1aa6e8da982:53, 00051a76-f711-89b6-
0000-00000003bac:::5f13678e-6ffd-4b33-912f-f1aa6e8da982:54, 00051a76-f711-
89b6-0000-00000003bac:::5f13678e-6ffd-4b33-912f-f1aa6e8da982:55

```

## Troubleshooting

To help visualize the following diagram is provided:



Nutanix Node	Physical Port	Cumulus Linux Port
Node A (Green)	vmnic2	swp49
Node B (Blue)	vmnic2	swp50
Node C (Red)	vmnic2	swp51
Node D (Yellow)	vmnic2	swp52

## Enabling LLDP/CDP on VMware ESXi (Hypervisor on Nutanix)

1. Follow the directions on one of the following websites to enable CDP:

- a. [http://kb.vmware.com/selfservice/microsites/search.do?language=en\\_US&cmd=displayKC&externalId=1003885](http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1003885)
- b. <http://wahlnetwork.com/2012/07/17/utilizing-cdp-and-lldp-with-vsphere-networking/>  
e.g. Switch CDP on:

```
root@NX-1050-A:~] esxcli network vswitch standard set -c both -v  
vSwitch0
```

Then confirm it is running:

```
root@NX-1050-A:~] esxcli network vswitch standard list -v vSwitch0  
vSwitch0  
  Name: vSwitch0  
  Class: etherswitch  
  Num Ports: 4082  
  Used Ports: 12  
  Configured Ports: 128  
  MTU: 1500  
  CDP Status: both  
  Beacon Enabled: false  
  Beacon Interval: 1  
  Beacon Threshold: 3  
  Beacon Required By:  
  Uplinks: vmnic3, vmnic2, vmnic1, vmnic0  
  Portgroups: VM Network, Management Network
```

The **both** means CDP is now running, and the lldp dameon on Cumulus Linux is capable of 'seeing' CDP devices.

2. After the next CDP interval, the Cumulus Linux box will pick up the interface via the **lldp** daemon:

```
cumulus@switch$ lldpctl show neighbor swp49  
-----  
-----  
LLDP neighbors:  
-----  
-----  
Interface:      swp49, via: CDPv2, RID: 6, Time: 0 day, 00:34:58  
Chassis:  
  ChassisID:      local NX-1050-A  
  SysName:        NX-1050-A  
  SysDescr:       Releasebuild-2494585 running on VMware ESX
```

```

MgmtIP:      0.0.0.0
Capability:   Bridge, on
Port:
  PortID:     ifname vmnic2
  PortDescr:  vmnic2
-----
```

3. Use `netshow` to look at `lldp` information:

```

cumulus@switch$ netshow lldp
-----
To view the legend, rerun "netshow" cmd with the "--legend" option
-----
Local Port      Speed      Mode          Remote Port      Remote
Host           Summary
-----  -----  -----  -----  -----
-----  -----
eth0           1G        Mgmt        ===  swp32          swoob.vsokt.
local IP: 192.168.0.111/24(DHCP)
swp49          10G(SFP+)  Access/L2  ===  vmnic2        NX-1050-
A              Untagged: br-ntnx
swp50          10G(SFP+)  Access/L2  ===  vmnic2        NX-1050-
B              Untagged: br-ntnx
swp51          10G(SFP+)  Access/L2  ===  vmnic2        NX-1050-
C              Untagged: br-ntnx
swp52          10G(SFP+)  Access/L2  ===  vmnic2        NX-1050-
D              Untagged: br-ntnx
```

## ***Enabling LLDP/CDP on Nutanix Acropolis (Hypervisor on Nutanix Acropolis)***

Nutanix Acropolis is an alternate hypervisor that Nutanix supports. Acropolis Hypervisor uses the yum packaging system and is capable of installing normal Linux lldp daemons to operating just like Cumulus Linux. LLDP should be enabled for each interface on the host. Refer to <https://community.mellanox.com/docs/DOC-1522> for setup instructions.

## ***snmpwalk the Switch from Another Linux Device***

One of the most important ways to troubleshoot is to snmpwalk the switch from another Linux device that can reach the switch running Cumulus Linux. For this demonstration, another switch running Cumulus Linux within the network is used.

1. Open `/etc/apt/sources.list` in an editor.
2. Add the following line, and save the file:

```
deb http://ftp.us.debian.org/debian/ jessie main non-free
```

3. Update the switch:

```
cumulus@switch2$ sudo apt-get update
```

4. Install the snmp and snmp-mibs-downloader packages:

```
cumulus@switch2$ sudo apt-get install snmp snmp-mibs-downloader
```

5. Verify that the "mibs :" line is commented out in /etc/snmp/snmp.conf:

```
#  
# As the snmp packages come without MIB files due to license reasons,  
loading  
# of MIBs is disabled by default. If you added the MIBs you can  
reenable  
# loading them by commenting out the following line.  
#mibs :
```

6. Perform an snmpwalk on the switch. The switch running snmpd in the demonstration is using IP address 192.168.0.111. It is possible to snmpwalk the switch from itself, following these instructions, ruling out an snmp problem vs networking problem.

```
cumulus@switch2$ snmpwalk -c public -v2c 192.168.0.111
```

## Output Examples

```
IF-MIB::ifPhysAddress.2 = STRING: 74:e6:e2:f5:a2:80  
IF-MIB::ifPhysAddress.3 = STRING: 0:e0:ec:25:b8:54  
IF-MIB::ifPhysAddress.4 = STRING: 74:e6:e2:f5:a2:81  
IF-MIB::ifPhysAddress.5 = STRING: 74:e6:e2:f5:a2:82  
IF-MIB::ifPhysAddress.6 = STRING: 74:e6:e2:f5:a2:83  
IF-MIB::ifPhysAddress.7 = STRING: 74:e6:e2:f5:a2:84  
IF-MIB::ifPhysAddress.8 = STRING: 74:e6:e2:f5:a2:85  
IF-MIB::ifPhysAddress.9 = STRING: 74:e6:e2:f5:a2:86  
IF-MIB::ifPhysAddress.10 = STRING: 74:e6:e2:f5:a2:87
```

```

IF-MIB::ifPhysAddress.11 = STRING: 74:e6:e2:f5:a2:88
IF-MIB::ifPhysAddress.12 = STRING: 74:e6:e2:f5:a2:89
IF-MIB::ifPhysAddress.13 = STRING: 74:e6:e2:f5:a2:8a
IF-MIB::ifPhysAddress.14 = STRING: 74:e6:e2:f5:a2:8b
IF-MIB::ifPhysAddress.15 = STRING: 74:e6:e2:f5:a2:8c
IF-MIB::ifPhysAddress.16 = STRING: 74:e6:e2:f5:a2:8d
IF-MIB::ifPhysAddress.17 = STRING: 74:e6:e2:f5:a2:8e
IF-MIB::ifPhysAddress.18 = STRING: 74:e6:e2:f5:a2:8f
IF-MIB::ifPhysAddress.19 = STRING: 74:e6:e2:f5:a2:90

```

Any information gathered here should verify that snmpd is running correctly on the Cumulus Linux side, reducing locations where a problem may reside.

## Troubleshooting Tips Table for snmp walks

Run snmpwalk from	If it works	If it does not work
<b>switch</b> (switch to be monitored)	snmpd is serving information correctly Problem resides somewhere else (e.g. network connectivity, Prism misconfiguration)	Is snmpd misconfigured or installed incorrectly?
<b>switch2</b> (another Cumulus Linux switch in the network)	snmpd is serving information correctly and network reachability works between <b>switch</b> and <b>switch2</b> Problems resides somewhere else (e.g. can Prism reach <b>switch</b> , Prism misconfiguration)	Network connectivity is not able to grab information? Is there an iptables rule blocking? Is the snmp walk being run correctly?
<b>Nutanix Prism CLI</b> (ssh to the cluster IP address)	snmpd is serving information correctly and network reachability works between <b>switch</b> and the <b>Nutanix Appliance</b> Problems resides somewhere else (e.g. The GUI might be misconfigured)	Is the right community name being used in the GUI? Is snmp v2c being used?

## Troubleshooting Connections without LLDP or CDP

- Find the MAC address information in the Prism GUI, located in: **Hardware -> Table -> Host -> Host NICs**
- Select a MAC address to troubleshoot (e.g. 0c:c4:7a:09:a2:43 represents vmnic0 which is tied to NX-1050-A).
- List out all the MAC addresses associated to the bridge:

```

cumulus@switch$ brctl showmacs br-ntnx
port name mac addr          vlan      is local?    ageing

```

timer					
swp9	00:02:00:00:00:06	0	no	66.94	
swp52	00:0c:29:3e:32:12	0	no	2.73	
swp49	00:0c:29:5a:f4:7f	0	no	2.73	
swp51	00:0c:29:6f:e1:e4	0	no	2.73	
swp49	00:0c:29:74:0c:ee	0	no	2.73	
swp50	00:0c:29:a9:36:91	0	no	2.73	
swp9	08:9e:01:f8:8f:0c	0	no	13.56	
swp9	08:9e:01:f8:8f:35	0	no	2.73	
swp4	0c:c4:7a:09:9e:d4	0	no	24.05	
swp1	0c:c4:7a:09:9f:8e	0	no	13.56	
swp3	0c:c4:7a:09:9f:93	0	no	13.56	
swp2	0c:c4:7a:09:9f:95	0	no	24.05	
swp52	0c:c4:7a:09:a0:c1	0	no	2.73	
swp51	0c:c4:7a:09:a2:35	0	no	2.73	
swp49	0c:c4:7a:09:a2:43	0	no	2.73	
swp9	44:38:39:00:82:04	0	no	2.73	
swp9	74:e6:e2:f5:a2:80	0	no	2.73	
swp1	74:e6:e2:f5:a2:81	0	yes	0.00	
swp2	74:e6:e2:f5:a2:82	0	yes	0.00	
swp3	74:e6:e2:f5:a2:83	0	yes	0.00	
swp4	74:e6:e2:f5:a2:84	0	yes	0.00	
swp5	74:e6:e2:f5:a2:85	0	yes	0.00	
swp6	74:e6:e2:f5:a2:86	0	yes	0.00	
swp7	74:e6:e2:f5:a2:87	0	yes	0.00	
swp8	74:e6:e2:f5:a2:88	0	yes	0.00	
swp9	74:e6:e2:f5:a2:89	0	yes	0.00	
swp10	74:e6:e2:f5:a2:8a	0	yes	0.00	
swp49	74:e6:e2:f5:a2:b1	0	yes	0.00	
swp50	74:e6:e2:f5:a2:b2	0	yes	0.00	
swp51	74:e6:e2:f5:a2:b3	0	yes	0.00	
swp52	74:e6:e2:f5:a2:b4	0	yes	0.00	
swp9	8e:0f:73:1b:f8:24	0	no	2.73	
swp9	c8:1f:66:ba:60:cf	0	no	66.94	

Alternatively, you can use grep:p

```
cumulus@switch$ brctl showmacs br-ntnx | grep 0c:c4:7a:09:a2:43
swp49      0c:c4:7a:09:a2:43          0        no           4.58
cumulus@switch$
```

vmnic1 is now hooked up to swp49. This matches what is seen in lldp:

```
cumulus@switch$ lldpctl show neighbor swp49
-----
-----
LLDP neighbors:
-----
Interface:      swp49, via: CDPv2, RID: 6, Time: 0 day, 01:11:12
  Chassis:
    ChassisID:      local NX-1050-A
    SysName:        NX-1050-A
    SysDescr:       Releasebuild-2494585 running on VMware ESX
    MgmtIP:         0.0.0.0
    Capability:    Bridge, on
  Port:
    PortID:        ifname vmnic2
    PortDescr:     vmnic2
-----
-----
cumulus@switch$
```

## **SNMP Traps**

### ***snmptrapd.conf***

The Net-SNMP trap daemon configuration file, `/etc/snmptrapd.conf`, is used to configure how incoming traps should be processed. For more information about specific configuration options within the file, run the following command in a terminal:

```
cumulus@switch:~$ man 5 snmptrapd.conf
#####
###
```

```
#
# EXAMPLE-trap.conf:
#   An example configuration file for configuring the Net-SNMP snmptrapd
agent.
#
#####
####
#
# This file is intended to only be an example. If, however, you want
```

```
# to use it, it should be placed in /etc/snmp/snmptrapd.conf.
# When the snmptrapd agent starts up, this is where it will look for it.
#
# All lines beginning with a '#' are comments and are intended for you
# to read. All other lines are configuration commands for the agent.

#
# PLEASE: read the snmptrapd.conf(5) manual page as well!
#
snmpTrapdAddr localhost
forward default {{global['snmp_server']}}}
```

## ***Generating Event Notification Traps***

The Net-SNMP agent provides a method to generate SNMP trap events, via the Distributed Management (DisMan) Event MIB, for various system events, including linkup/down, exceeding the temperature sensor threshold, CPU load, or memory threshold, or other SNMP MIBs.

## ***Enabling MIB to OID Translation***

MIB names can be used instead of OIDs, by installing the `snmp-mibs-downloader`, to download SNMP MIBs to the switch prior to enabling traps. This greatly improves the readability of the `snmpd.conf` file.

1. Open `/etc/apt/sources.list` in a text editor.
2. Add the `non-free` repository, and save the file:

```
cumulus@switch:~$ deb http://ftp.us.debian.org/debian/ jessie main non-free
```

3. Update the switch:

```
cumulus@switch:~$ apt-get update
```

4. Install the `snmp-mibs-downloader`:

```
apt-get snmp-mibs-downloader
```

5. Open the `/etc/snmp/snmp.conf` file to verify that the `mibs : line` is commented out:

```
#  
# As the snmp packages come without MIB files due to license reasons,
```

```

loading
# of MIBs is disabled by default. If you added the MIBs you can
reenable
# loading them by commenting out the following line.
#mibs :

```

6. Open the `/etc/default/snmpd` file to verify that the `export MIBS=` line is commented out:

```

# This file controls the activity of snmpd and snmptrapd

# Don't load any MIBs by default.
# You might comment this lines once you have the MIBs Downloaded.
#export MIBS=

```

7. Once the configuration has been confirmed, remove or comment out the `non-free` repository in `/etc/apt/sources.list`.

```
#deb http://ftp.us.debian.org/debian/ jessie main non-free
```

## Configuring Trap Events

The following configurations should be made in `/etc/snmp/snmp.conf`, in order to enable specific types of traps. Once configured, restart the `snmpd` service to apply the changes.

```
cumulus@switch:~$ sudo systemctl restart snmpd.service
```

## Defining Access Credentials

An SNMPv3 username is required to authorize the DisMan service. The example code below uses `cumulusUser` as the username.

```

createUser cumulusUser
iquerySecName cumulusUser
rouser cumulusUser

```

## Defining Trap Receivers

The example code below creates a trap receiver that is capable of receiving SNMPv2 traps.

```
trap2sink 192.168.1.1 public
```



Although the traps are sent to an SNMPV2 receiver, the SNMPv3 user is still required.



It is possible to define multiple trap receivers, and to use the domain name instead of IP address in the `trap2sink` directive.

## Configuring LinkUp/Down Notifications

The `linkUpDownNotifications` directive is used to configure linkup/down notifications when the operational status of the link changes.

```
linkUpDownNotifications yes
```



The default frequency for checking link up/down is 60 seconds. The default frequency can be changed using the `monitor` directive directly instead of the `linkUpDownNotifications` directive. See `man snmpd.conf` for details.

## Configuring Temperature Notifications

Temperature sensor information for each available sensor is maintained in the the `ImSensors` MIB. Each platform may contain a different number of temperature sensors. The example below generates a trap event when any temperature sensors exceeds a threshold of 68 degrees (centigrade). It monitors each `lmTempSensorsValue`. When the threshold value is checked and exceeds the `lmTempSensorsValue`, a trap is generated. The `-o lmTempSensorsDevice` option is used to instruct SNMP to also include the `ImTempSensorsDevice` MIB in the generated trap. The default frequency for the `monitor` directive is 600 seconds. The default frequency may be changed using the `-r` option.:.

```
monitor lmTemSensor -o lmTempSensorsDevice lmTempSensorsValue > 68000
```

Alternatively, temperature sensors may be monitored individually. To monitor the sensors individually, first use the `sensors` command to determine which sensors are available to be monitored on the platform.

```
#sensors

CY8C3245-i2c-4-2e
Adapter: i2c-0-mux (chan_id 2)
fan5: 7006 RPM (min = 2500 RPM, max = 23000 RPM)
fan6: 6955 RPM (min = 2500 RPM, max = 23000 RPM)
fan7: 6799 RPM (min = 2500 RPM, max = 23000 RPM)
fan8: 6750 RPM (min = 2500 RPM, max = 23000 RPM)
```

```
temp1: +34.0 C (high = +68.0 C)
temp2: +28.0 C (high = +68.0 C)
temp3: +33.0 C (high = +68.0 C)
temp4: +31.0 C (high = +68.0 C)
temp5: +23.0 C (high = +68.0 C)
```

Configure a `monitor` command for the specific sensor using the `-I` option. The `-I` option indicates that the monitored expression is applied to a single instance. In this example, there are five temperature sensors available. The following monitor directive can be used to monitor only temperature sensor three at five minute intervals.

```
monitor -I -r 300 lmTemSensor3 -o lmTempSensorsDevice.3 lmTempSensorsValue.
3 > 68000
```

## Configuring Free Memory Notifications

You can monitor free memory using the following directives. The example below generates a trap when free memory drops below 1,000,000KB. The free memory trap also includes the amount of total real memory:

```
monitor MemFreeTotal -o memTotalReal memTotalFree < 1000000
```

## Configuring Processor Load Notifications

To monitor CPU load for 1, 5 or 15 minute intervals, use the `load` directive in conjunction with the `monitor` directive. The following example will generate a trap when the 1 minute interval reaches 12%, the 5 minute interval reaches 10% or the 15 minute interval reaches 5%.

```
load 12 10 5
monitor -r 60 -o laNames -o laErrMsg "laTable" laErrorFlag !=0
```

## Configuring Disk Utilization Notifications

To monitor disk utilization for all disks, use the `includeAllDisks` directive in conjunction with the `monitor` directive. The example code below generates a trap when a disk is 99% full:

```
includeAllDisks 1%
monitor -r 60 -o dskPath -o DiskErrMsg "dskTable" diskErrorFlag !=0
```

## Configuring Authentication Notifications

To generate authentication failure traps, use the `authtrapenable` directive:

```
authtrapenable 1
```

## Supported MIBs

Below are the MIBs supported by Cumulus Linux, as well as suggested uses for them. The overall Cumulus Linux MIB is defined in `/usr/share/snmp/mibs/Cumulus-Snmp-MIB.txt`.

MIB Name	Suggested Uses
BRIDGE and Q-BRIDGE	The dot1dBasePortEntry and dot1dBasePortIfIndex tables in the BRIDGE-MIB and dot1qBase, dot1qFdbEntry, dot1qTpFdbEntry, dot1qTpFdbStatus, and the dot1qVlanStaticName tables in the Q-BRIDGE-MIB tables. You must uncomment the <code>bridge_pp.py pass_persist</code> script in <code>/etc/snmp/snmpd.conf</code> .
CUMULUS-COUNTERS-MIB	Discard counters: Cumulus Linux also includes its own counters MIB, defined in <code>/usr/share/snmp/mibs/Cumulus-Counters-MIB.txt</code> . It has the OID <code>.1.3.6.1.4.1.40310.2</code> .
CUMULUS-RESOURCE-QUERY-MIB	Cumulus Linux includes its own resource utilization MIB, which is similar to using <a href="#">cl-resource-query (see page 506)</a> . It monitors L3 entries by host, route, nexthops, ECMP groups and L2 MAC/BDPU entries. The MIB is defined in <code>/usr/share/snmp/mibs/Cumulus-Resource-Query-MIB.txt</code> , and has the OID <code>.1.3.6.1.4.1.40310.1</code> .
DISMAN-EVENT	Trap monitoring
ENTITY	From RFC 4133, the temperature sensors, fan sensors, power sensors, and ports are covered.
ENTITY-SENSOR	Physical sensor information (temperature, fan, and power supply) from RFC 3433.
HOST-RESOURCES	Users, storage, interfaces, process info, run parameters
IF-MIB	Interface description, type, MTU, speed, MAC, admin, operation status, counters
IP (includes ICMP)	IPv4, IPv4 addresses, counters, netmasks
IPv6	IPv6 counters
IP-FORWARD	IP routing table
LLDP	L2 neighbor info from <code>lldpd</code> (note, you need to <a href="#">enable the SNMP subagent (see page 198)</a> in LLDP). <code>lldpd</code> needs to be started with the <code>-x</code> option to enable connectivity to <code>snmpd</code> (AgentX).

MIB Name	Suggested Uses
LM-SENSORS MIB	Fan speed, temperature sensor values, voltages. This is deprecated since the ENTITY-SENSOR MIB has been added.
NET-SNMP-AGENT	Agent timers, user, group config
NET-SNMP-EXTEND	Agent timers, user, group config
NET-SNMP-EXTEND-MIB	(See also <a href="#">this knowledge base article</a> on extending NET-SNMP in Cumulus Linux to include data from power supplies, fans and temperature sensors.)
NET-SNMP-VACM	Agent timers, user, group config
NOTIFICATION-LOG	Local logging
SNMP-FRAMEWORK	Users, access
SNMP-MPD	Users, access
SNMP-TARGET	
SNMP-USER-BASED-SM	Users, access
SNMP-VIEW-BASED-ACM	Users, access
SNMPv2	SNMP counters (For information on exposing CPU and memory information via SNMP, see this <a href="#">knowledge base article</a> .)
TCP	TCP related information
UCD-SNMP	System memory, load, CPU, disk IO
UDP	UDP related information



The Quagga and Zebra routes MIBs are disabled in Cumulus Linux.





The ENTITY MIB does not currently show the chassis information in Cumulus Linux 3.0.

# Cumulus Networks Services Demos

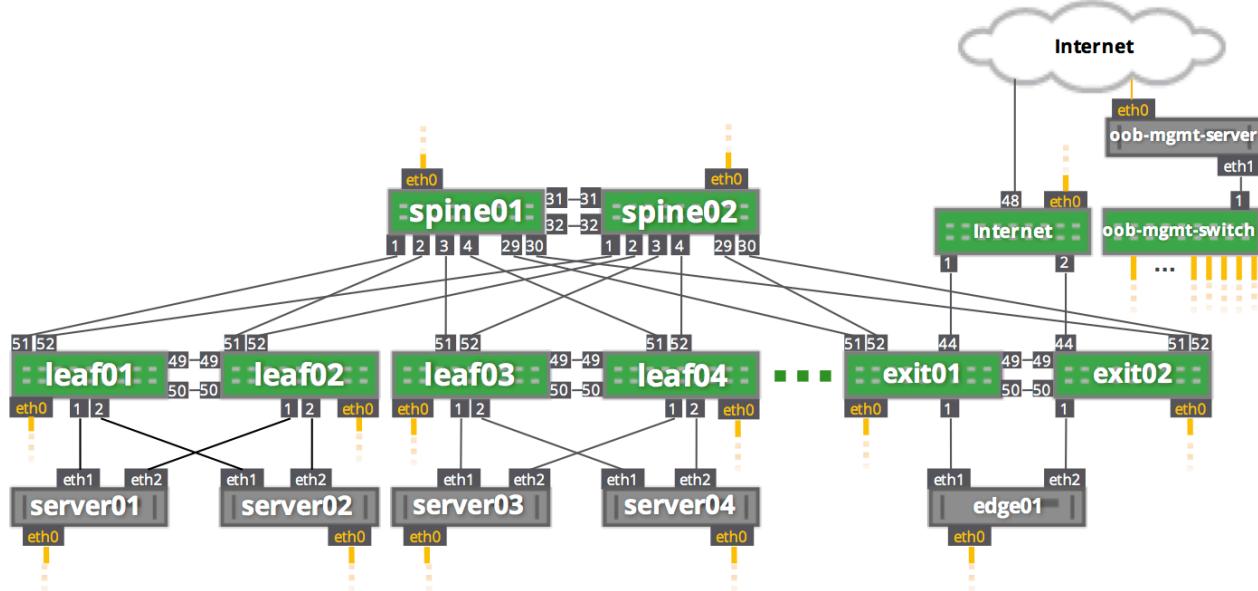
The Cumulus Networks Services team demos provide a virtual environment built using either VirtualBox or `libvirt` using Vagrant to manage the VMs. This environment utilizes the reference topology shown below. Vagrant and Cumulus VX can be used together to build virtual simulations of production networks to validate configurations, develop automation code and simulate failure scenarios.

## Contents

- Reference Topology (see page 593)
  - IP and MAC Addressing (see page 594)
  - Building the Topology (see page 595)
    - Virtual Appliance (see page 595)
    - Hardware (see page 595)
  - Demos (see page 595)

## Reference Topology

The Cumulus Networks *reference topology* includes cabling (in DOT format for dual use with [PTM \(see page 198\)](#)), MAC addressing, IP addressing, switches and servers. This topology is blessed by the Professional Services Team at Cumulus Networks to fit a majority of designs seen in the field.



## IP and MAC Addressing

Hostname	eth0 IP	eth0 MAC	Interface Count
oob-mgmt-server	192.168.0.254	any	
oob-mgmt-switch	192.168.0.1	any	Cumulus RMP
leaf01	192.168.0.11	A0:00:00:00:00:11	48x10g w/ 6x40g uplink
leaf02	192.168.0.12	A0:00:00:00:00:12	48x10g w/ 6x40g uplink
leaf03	192.168.0.13	A0:00:00:00:00:13	48x10g w/ 6x40g uplink
leaf04	192.168.0.14	A0:00:00:00:00:14	48x10g w/ 6x40g uplink
spine01	192.168.0.21	A0:00:00:00:00:21	32x40g
spine02	192.168.0.22	A0:00:00:00:00:22	32x40g
server01	192.168.0.31	A0:00:00:00:00:31	10g NICs
server02	192.168.0.32	A0:00:00:00:00:32	10g NICs
server03	192.168.0.33	A0:00:00:00:00:33	10g NICs
server04	192.168.0.34	A0:00:00:00:00:34	10g NICs
exit01	192.168.0.41	A0:00:00:00:00:41	48x10g w/ 6x40g uplink (exit leaf)
exit02	192.168.0.42	A0:00:00:00:00:42	48x10g w/ 6x40g uplink (exit leaf)

Hostname	eth0 IP	eth0 MAC	Interface Count
edge01	192.168.0.51	A0:00:00:00:00:51	10g NICs (customer edge device, firewall, load balancer, etc.)
internet	192.168.0.253	any	(represents internet provider edge device)

## ***Building the Topology***

### ***Virtual Appliance***

You can build out the reference topology in hardware or using Cumulus VX (the free Cumulus Networks virtual appliance). The [Cumulus Reference Topology using Vagrant](#) is essentially the reference topology built out inside Vagrant with VirtualBox or KVM. The installation and setup instructions for bringing up the entire reference topology on a laptop or server are on the [cldemo-vagrant GitHub repo](#).

### ***Hardware***

Any switch from the [hardware compatibility list](#) is compatible with the topology as long as you follow the interface count from the table above. Of course, in your own production environment, you don't have to use exactly the same devices and cabling as outlined above.

### ***Demos***

You can find an up to date list of all the demos in the [cldemo-vagrant GitHub repository](#), which is available to anyone free of charge.

# Docker on Cumulus Linux

Cumulus Linux 3.0 is based on Linux kernel 4.1, which supports the [Docker](#) engine. This means you can install Docker directly on a Cumulus Linux switch and you can run Docker containers natively on the switch.

```
cumulus@switch:~$ uname -r  
4.1.0-cl-1-amd64
```

## Contents

- [Installing Docker \(see page 596\)](#)
  - [Verifying the Docker Install \(see page 596\)](#)
- [Caveats when Using Docker with Cumulus Linux \(see page 596\)](#)
  - [iptables and Docker \(see page 597\)](#)
  - [Management VRF and Docker \(see page 598\)](#)

## Installing Docker

Before installing Docker, add the Debian Jessie `apt` repository to Cumulus Linux:

```
cumulus@switch:~$ echo "deb http://ftp.us.debian.org/debian/ jessie  
main contrib non-free" | sudo tee -a /etc/apt/sources.list
```

To install the Docker Engine, follow the [instructions](#) for installing it on Debian Linux, as they work for Cumulus Linux as is.



There's a community-supported Ansible playbook for installing Docker on Cumulus Linux on [GitHub](#).

## Verifying the Docker Install

Docker provides a "Hello World" testing application that can be run on Cumulus Linux to validate that Docker Engine was installed correctly.

```
cumulus@switch:~$ docker run ubuntu /bin/echo 'Hello world'  
Hello world
```

## Caveats when Using Docker with Cumulus Linux

### ***iptables and Docker***

By default, Docker Engine creates **iptables** rules to manage Docker container connectivity and provide IP masquerade (NAT). Docker Engine adds NAT entries to the *nat* table. These ACL rules do not properly sync with the hardware of Cumulus Linux, so when you run **cl-acltool -i** to install new hardware based ACLs, the Docker Engine rules get removed.

When using Docker Engine on Cumulus Linux, you must disable the **iptables** functionality within Docker Engine. To do so, use the **--iptables=false** option when you start Docker Engine. To do this automatically, you can modify the **systemd** service startup parameters:

1. Create the directory **/etc/systemd/system/docker.service.d**:

```
cumulus@switch:~$ sudo mkdir /etc/systemd/system/docker.service.d
```

2. Create the file **/etc/systemd/system/docker.service.d/noiptables.conf** and populate it with the following:

```
cumulus@switch:~$ sudo nano /etc/systemd/system/docker.service.d/noiptables.conf
```

```
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon -H fd:// --iptables=false
```

3. Reload the **systemd** configuration settings; this has no impact on currently running services.

```
cumulus@switch:~$ sudo systemctl daemon-reload
```

4. If Docker Engine was already started, reset the nat table entries, then reset the software **iptables** rules:

```
cumulus@switch:~$ sudo iptables -t nat -F
cumulus@switch:~$ sudo cl-acltool -i
```

5. Restart the Docker Engine:

```
cumulus@switch:~$ sudo systemctl restart docker.service
```

To check if the Docker Engine **iptables** rules have been installed, run **iptables -L DOCKER** to print the current software **iptables** rules. If the Docker Engine **iptables** rules are installed, you will see output similar to the following:

```
cumulus@switch:~$ sudo iptables -L DOCKER
Chain DOCKER (1 references)
target  prot opt source destination
cumulus@switch:~$
```

If the Docker Engine `iptables` rules are not installed, an error gets returned:

```
cumulus@switch:~$ sudo iptables -L DOCKER
iptables: No chain/target/match by that name.
```

## ***Management VRF and Docker***

If you have a management VRF configured on a Cumulus Linux 3.0.z switch, you cannot run Docker commands that go to another node (for example, `docker pull`) over the management VRF.

# Index

4

40G ports 159  
logical limitations 159

8

802.1p 161  
    class of service 161  
802.3ad link aggregation 261

A

ABRs 408  
    area border routers 408  
access control lists 95  
access ports 229  
ACL policy files 105  
ACL rules 164  
ACLs 95, 97, 112  
    chains 97  
    QoS 112  
active-active mode 266, 330  
    VRR 266  
    VXLAN 330  
active listener ports 123  
active-standby mode 266  
    VRR 266  
Algorithm Longest Prefix Match 387  
    routing 387  
ALPM mode 387  
    routing 387  
AOC cables 21  
apt-get 53  
area border routers 408  
    ABRs 408  
arp cache 546  
ARP requests 267  
    VRR 267  
AS\_PATH setting 437

BGP [437](#)  
ASN [421](#)  
autonomous system number [421](#)  
auto-negotiation [152](#)  
autonomous system number [421](#)  
BGP [421](#)  
autoprovisioning [59](#)

## B

bestpath [437](#)  
    BGP [437](#)  
BFD [204, 205](#)  
    Bidirectional Forwarding Detection [204](#)  
    echo function [205](#)  
BGP [419, 422, 469](#)  
    Border Gateway Protocol [419](#)  
    ECMP [422](#)  
    virtual routing and forwarding (VRF) [469](#)  
BGP peering relationships [435, 435](#)  
    external [435](#)  
    internal [435](#)  
bonds [211, 261](#)  
    LACP Bypass [261](#)  
boot recovery [503](#)  
bpdufilter [180](#)  
    and STP [180](#)  
BPDU guard [177](#)  
    and STP [177](#)  
brctl [23, 174, 216, 217](#)  
    and STP [174](#)  
bridge assurance [180](#)  
    and STP [180](#)  
bridges [215, 215, 216, 216, 216, 217, 218, 218, 222, 224, 229, 229, 235, 243](#)  
    access ports [229](#)  
    adding interfaces [216, 217](#)  
    adding IP addresses [222](#)  
    IGMP snooping [243](#)  
    MAC addresses [218](#)  
    MTU [218](#)  
    physical interfaces [216](#)  
    trunk ports [229](#)  
    untagged frames [224](#)  
VLAN-aware [215, 235](#)

## C

cable connectivity 21  
cabling 198  
    Prescriptive Topology Manager 198  
chain 97  
cl-acltool 95, 164, 547  
CLAG 267  
    and VRR 267  
clagctl 253  
class of service 161  
cl-bgp 402  
cl-cfg 128, 516  
cl-ecmpcalc 453  
cl-license 21  
cl-netstat 541  
cl-ospf 402, 409  
cl-ospf6 402, 417  
Clos topology 390  
cl-ra 402  
cl-rctl 402  
cl-resource-query 128, 506  
cl-route-check 415  
cl-support 499  
convergence 389  
    routing 389  
Cumulus Linux 17, 18, 27, 28, 31, 243, 370  
    installing 17, 31  
    reprovisioning 27  
    reserved VLAN ranges 243  
    uninstalling 28  
    upgrading 18  
    VXLAN 370  
cumulus user 77

## D

DAC cables 21  
daemons 122  
datapath 161  
datapath.conf 161  
date 73  
    setting 73

deb 58  
debugging 497  
decode-syseeprom 508  
differentiated services code point 161  
dmidecode 509  
dpkg 56  
dpkg-reconfigure 73  
DSCP 161  
    differentiated services code point 161  
DSCP marking 164  
dual-connected hosts 246  
duplex interfaces 150  
dynamic routing 207, 391  
    and PTM 207  
    quagga 391

## E

eBGP 421  
    external BGP 421  
ebtables 95, 102  
    memory spaces 102  
echo function 205, 205  
    BFD 205  
    PTM 205  
ECMP 391, 414, 422, 1  
    BGP 422  
    equal cost multi-pathing 391  
    monitoring 1  
    OSPF 414  
ECMP hashing 452, 455  
    resilient hashing 455  
EGP 392  
    Exterior Gateway Protocol 392  
equal cost multipath 452  
    ECMP hashing 452  
equal cost multi-pathing 391  
    ECMP 391  
ERSPAN 548  
    network troubleshooting 548  
Ethernet management port 19  
ethtool 161, 539  
    switch ports 161  
external BGP 421

eBGP [421](#)

## F

fast convergence [434](#)

BGP [434](#)

First Hop Redundancy Protocol [266](#)

VRR [266](#)

## G

glob [146](#)

Graphviz [198](#)

## H

hardware [507](#)

monitoring [507](#)

hardware compatibility list [17](#)

hash distribution [214](#)

HCL [17](#)

head end replication [305](#)

LNV [305](#)

high availability [391](#)

host entries [506](#)

monitoring [506](#)

hostname [19](#)

hsflowd [565](#)

hwclock [74](#)

## I

iBGP [421](#)

internal BGP [421](#)

ifdown [135](#)

ifplugged [267](#)

VRR [267](#)

ifquery [140, 535](#)

ifup [135](#)

ifupdown [134](#)

ifupdown2 [144, 227, 534, 534, 535](#)

excluding interfaces [535](#)

- logging 534
- purging IP addresses 144
- troubleshooting 534
- VLAN tagging 227
- IGMP snooping 243, 257, 376
  - MLAG 257
  - VLAN-aware bridges 243
- IGP 392
  - Interior Gateway Protocol 392
- image contents 29
- installing 17
  - Cumulus Linux 17
- interface counters 541
- interface dependencies 139
- interfaces 149, 160
  - statistics 160
- internal BGP 421
  - iBGP 421
- ip6tables 95
- IP addresses 144
  - purging 144
- iproute2 539
  - failures 539
- iptables 95
- IPv4 routes 423
  - BGP 423
- IPv6 routes 423
  - BGP 423

## L

- LACP 211, 244
  - MLAG 244
- LACP Bypass 261
- layer 3 access ports 23
  - configuring 23
- LDAP 86
- leaf-spine topology 390
- license 20
  - installing 20
- lightweight network virtualization 303, 305, 305, 363
  - head end replication 305
  - service node replication 305
- link aggregation 211

Link Layer Discovery Protocol 191

link-local IPv6 addresses 440

BGP 440

link pause 165

  datapath 165

link-state advertisement 406

link state monitoring 267

  VRR 267

LLDP 191, 198

  SNMP 198

lldpcli 193

lldpd 191, 200

LNV 303, 303, 305, 305, 363, 363

  head end replication 305

  service node replication 305

  VXLAN 303, 363

load balancing 391

logging 501, 534, 534

  ifupdown2 534

  networking service 534

logging neighbor state changes 440

  BGP 440

logical switch 244

longest prefix match 387

  routing 387

loopback interface 24

  configuring 24

LSA 406

  link-state advertisement 406

LSDB 407

  link-state database 407

lshw 509

## M

MAC entries 506

  monitoring 506

Mako templates 146, 536

  debugging 536

mangle table 165

  ACL rules 165

memory spaces 102

  ebtables 102

MLAG 244, 254, 254, 254, 257, 257, 258, 260

backup link 254  
IGMP snooping 257  
MTU 258  
peer link states 254  
protodown state 254  
STP 260  
MLD snooping 376  
monitoring 72, 497, 506, 512, 513, 539, 564, 567  
    hardware watchdog 512  
    Net-SNMP 567  
    network traffic 564  
mstpcctl 172, 231  
MTU 152, 218, 258, 538  
    bridges 218  
    failures 538  
    MLAG 258  
multi-Chassis Link Aggregation 244  
    MLAG 244  
multiple bridges 219  
mz 546  
    traffic generator 546

## N

name switch service 85  
Netfilter 95  
Net-SNMP 567  
networking service 534  
    logging 534  
network interfaces 134, 149  
    ifupdown 134  
network traffic 564  
    monitoring 564  
network troubleshooting 557  
    tcpdump 557  
network virtualization 269, 270, 370  
    VMware NSX 270  
no-as-set 437  
    BGP 437  
nonatomic updates 102  
    switchd 102  
non-blocking networks 390  
NSS 85  
    name switch service 85

NTP 74  
    time 74  
    ntpd 74

## O

ONIE 17, 28  
    rescue mode 28  
onie-select 27  
Open Network Install Environment 17  
Open Shortest Path First Protocol 406, 417  
    OSPFv2 406  
    OSPFv3 417  
open source contributions 16  
OSPF 411, 413, 414, 414, 416  
    ECMP 414  
    reconvergence 414  
    summary LSA 411  
    supported RFCs 416  
    unnumbered interfaces 413  
ospf6d.conf 418  
OSPFv2 406  
OSPFv3 417, 419, 419  
    supported RFCs 419  
    unnumbered interfaces 419  
over-subscribed networks 390

## P

packages 53  
    managing 53  
packet buffering 161  
    datapath 161  
packet queueing 161  
    datapath 161  
packet scheduling 161  
    datapath 161  
PAM 85  
    pluggable authentication modules 85  
parent interfaces 142  
password 77  
    default 77  
passwordless access 77

passwords 19  
peer-groups 436  
    BGP 436  
Per VLAN Spanning Tree 173  
    PVST 173  
ping 545  
pluggable authentication modules 85  
policy.conf 108  
port lists 146  
port speeds 150  
Prescriptive Topology Manager 198  
priority groups 161  
    datapath 161  
privileged commands 80  
protocol tuning 388, 442  
    BGP 442  
    routing 388  
protodown state 254  
    MLAG 254  
PTM 198, 205  
    echo function 205  
    Prescriptive Topology Manager 198  
ptmctl 208  
ptmd 199  
PTM scripts 201  
PVRST 173  
    Rapid PVST 173  
PVST 173  
    Per VLAN Spanning Tree 173

## Q

QoS 112  
    ACLs 112  
QSFP 542  
Quagga 207, 207, 384, 391, 394  
    and PTM 207, 207  
    configuring 394  
    dynamic routing 391  
    static routing 384  
quality of service 167

## R

Rapid PVST [173](#)  
    PVRST [173](#)  
read-only mode [441](#)  
    BGP [441](#)  
recommended configuration [44](#)  
reconvergence [414](#)  
    OSPF [414](#)  
remote access [76](#)  
repositories [58](#)  
    other packages [58](#)  
rescue mode [28](#)  
reserved VLAN ranges [243](#)  
resilient hashing [455](#)  
restart [128](#)  
    switchd [128](#)  
root user [19, 77](#)  
route advertisements [421](#)  
    BGP [421](#)  
route entries [387, 387](#)  
    ALPM [387](#)  
    limitations [387](#)  
route maps [442](#)  
    BGP [442](#)  
route reflectors [422](#)  
    BGP [422](#)  
routes [506](#)  
    monitoring [506](#)  
routing protocols [388](#)  
RSTP [173](#)

## S

sensors command [509](#)  
serial console management [19](#)  
service node replication [305](#)  
    LNV [305](#)  
services [122](#)  
sFlow [564](#)  
sFlow visualization tools [567](#)  
SFP [161, 542](#)  
    switch ports [161](#)  
single user mode [503](#)

smonctl 511  
smond 511  
snmpd 567  
sources.list 58  
SPAN 548  
    network troubleshooting 548  
spanning tree parameters 183  
Spanning Tree Protocol 172, 235  
    STP 172  
        VLAN-aware bridges 235  
SSH 76  
SSH keys 76  
static routing 382, 384  
    with ip route 382  
    with Quagga 384  
storm control 181  
    STP 181  
STP 172, 180, 181, 260  
    and bridge assurance 180  
    MLAG 260  
    Spanning Tree Protocol 172  
    storm control 181  
stub areas 412  
    OSPF 412  
sudo 77, 79  
sudoers 79, 80  
    examples 80  
summary LSA 411  
    OSPF 411  
SVI 248  
    switched virtual interface 248  
switchd 102, 126, 126, 126, 128, 516  
    configuring 126  
    counters 516  
    file system 126  
    nonatomic updates 102  
    restarting 128  
switched virtual interface 248  
    SVI 248  
switch ports 22, 159  
    configuring 22  
    logical limitations 159  
syslog 501  
systemd 257  
system management 497

## T

tcpdump 557  
    network troubleshooting 557  
templates 146  
time 73  
    setting 73  
time zone 20, 72  
topology 198, 389  
    data center 198  
traceroute 545  
traffic.conf 161  
traffic distribution 214  
traffic generator 546  
    mz 546  
traffic marking 164  
    datapath 164  
troubleshooting 497, 503, 557  
    single user mode 503  
    tcpdump 557  
trunk ports 224, 229  
tzdata 73

## U

U-Boot 17, 497  
unnumbered interfaces 413, 419  
    OSPF 413  
    OSPFv3 419  
untagged frames 224  
    bridges 224  
upgrading 18  
    Cumulus Linux 18  
user accounts 77, 77  
    cumulus 77  
    root 77  
user authentication 85  
user commands 145  
    interfaces 145

## V

virtual device counters 513, 516, 516  
monitoring 513  
poll interval 516  
VLAN statistics 516  
virtual router redundancy 264  
virtual routing and forwarding (VRF) 469, 471  
    BGP 469  
    table ID 471  
visudo 79  
VLAN 248, 513  
    statistics 513  
    switched virtual interface 248  
VLAN-aware bridges 215, 235, 235, 236, 243  
    configuring 236  
    IGMP snooping 243  
    Spanning Tree Protocol 235  
VLAN tagging 227, 227, 228  
    advanced example 228  
    basic example 227  
VLAN translation 233  
VRR 264  
    virtual router redundancy 264  
VTEP 269, 271  
vtysh 396  
    quagga CLI 396  
VXLAN 269, 271, 303, 330, 363, 370, 513  
    active-active mode 330  
    LNV 303, 363  
    no controller 370  
    statistics 513  
    VMware NSX 271

## W

watchdog 512  
    monitoring 512

## Z

zebra 392  
    routing 392  
zero touch provisioning 59, 61  
    USB 61

ZTP 59