



Cumulus NetQ 1.1.0

User Guide



Table of Contents

What's New in NetQ 1.1.0	5
NetQ Components	6
Getting Started with NetQ	8
Contents	9
Install the NetQ Telemetry Server	9
Install the NetQ Agent	10
Installing on a Cumulus Linux Switch	11
Installing on an Ubuntu, Red Hat or CentOS Server	11
Configuring the NetQ Agent on a Node	11
Configuring the Agent to Use a VRF	12
Configuring the Agent to Communicate over a Specific Port	12
Configuring NetQ Notifier on the Telemetry Server	12
Example /etc/cts/netq/netq.yml Configuration	15
Upgrading NetQ	21
Contents	21
Upgrade the NetQ Telemetry Server	21
Upgrade the NetQ Agents	22
Connect the NetQ Telemetry Server to the Network	24
Getting to Know NetQ	24
Getting Information about Network Hardware	26
Using the NetQ Shell on the NetQ Telemetry Server	27
Using the netq resolve Command	28
Sample Commands for Various Components	31
Understanding Timestamps in NetQ	32
NetQ Service Console	33
Contents	34
Connecting to the Service Console	34
Getting Service Console Information	35
Accessing the NetQ Command Line	35
Taking Preventative Steps with Your Network	36
Contents	37
netq check and netq show	37
netq show agents	38
Using NetQ with Automation	39
Using NetQ Virtual	40



Proactively Monitoring the Network Fabric	40
Contents	41
NetQ Notifier	41
Log Message Format	42
Supported Integrations	42
Configuring an Integration	44
Filtering Notifications	45
Example netq.yml File	47
Exporting to ELK	53
Exporting to Splunk	54
Precisely Locating an Issue on the Network	54
Detecting Out of Sync Nodes	55
Extending NetQ with Custom Services Using curl	56
Exporting NetQ Data	57
MLAG Troubleshooting with NetQ	58
Contents	59
All Nodes Are Up	59
Dual-connected Bond Is Down	61
VXLAN Active-active Device or Interface Is Down	63
Remote-side clagd Stopped by systemctl Command	65
Performing Network Diagnostics	67
Contents	68
Diagnosing an Event after It Occurs	68
Using NetQ as a Time Machine	70
How Far Back in Time Can You Travel?	71
Using trace in a VRF	72
Monitoring the Physical Layer	72
Managing the Layer 1 Inventory	73
Checking Peer Connections	79
Layer 1 Configuration Checks	80
Monitoring Linux Hosts with NetQ	81
Monitoring Container Environments with NetQ	82
Installing and Configuring the NetQ Virtual Environment	83
Restoring from Backups with NetQ	84
Backup Locations	85
Use Cases	85
Restoring from a Snapshot	85
Early Access Features	86



Extending NetQ with Custom Commands	87
Contents	87
Sample File with Custom Command	88
Querying the NetQ Database	90
Debugging	92
Caveats	93
Facebook Backpack Integration	93
Troubleshooting NetQ	93
Checking Agent Health	94
Error Configuring the Telemetry Server on a Node	95
cts-support	96
Index	97

©2018 Cumulus Networks. All rights reserved

CUMULUS, the Cumulus Logo, CUMULUS NETWORKS, and the Rocket Turtle Logo (the "Marks") are trademarks and service marks of Cumulus Networks, Inc. in the U.S. and other countries. You are not permitted to use the Marks without the prior written consent of Cumulus Networks. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. All other marks are used under fair use or license from their respective owners.



Cumulus NetQ is a fabric-wide, telemetry-based validation system, that enables organizations to validate network state, both during regular operations and for post-mortem diagnostic analysis. Running on Cumulus Linux switches and other certified systems — such as Ubuntu, Red Hat, and CentOS hosts — NetQ captures network data and other state information in real time, allowing cloud architects and network operations teams to operate with visibility over the entire network.

The system uses a three-pronged approach to validating networks:

- **Preventative**

NetQ easily validates potential network configuration changes in a virtualized environment or lab using check, show and trace algorithms, eliminating the need to check nodes one by one and reducing manual errors before they are rolled into production (one of the main causes of network downtime).

- **Proactive**

NetQ detects faulty network states that can result in packet loss or connectivity issues, and alerts the user with precise fault location data to allow for faster remediation, greatly improving network agility, and reducing downtime costs.

- **Diagnostic**

NetQ provides the ability to trace network paths, replay the network state at a time in the past, review fabric-wide event changelogs and diagnose the root cause of state deviations.

What's New in NetQ 1.1.0

NetQ 1.1 includes the following new features and enhancements:

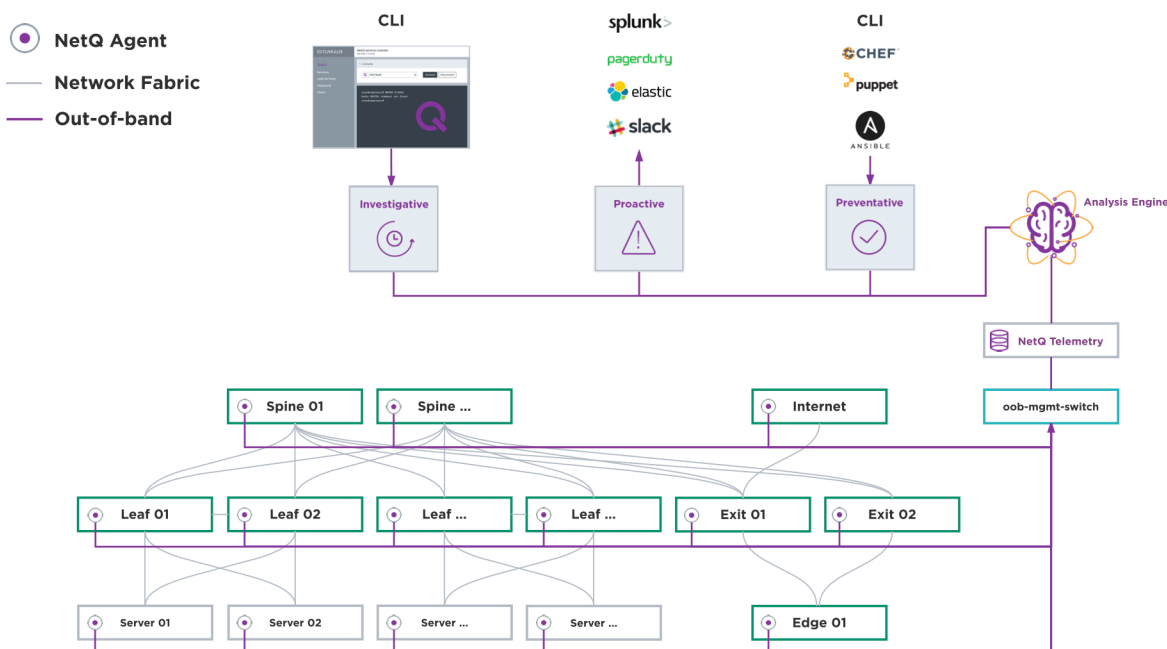
- Improved container visibility: [Docker Swarm integration \(see page 82\)](#) provides service- and application-level insights in containerized environments.
- [Layer 1 support \(see page 72\)](#): checkMonitor your physical layer, check the status of the inventory, check peer connections and investigate link mismatches and flapping.
- [EVPN](#): Fabric-wide configuration checks and traces enhancements so you can verify that your overlay control plane is configured correctly.
- [NetQ Notifier \(see page 40\)](#): Filter alerts and notifications based on severity, device and service, and redirect to the desired PagerDuty and Slack channels. Also, exporting NetQ data to [ELK and Splunk \(see page 40\)](#) is now generally available.
- [Telemetry server \(see page 8\)](#): The telemetry server is now available as a KVM-based virtual machine, which runs on Ubuntu 16.04 and Red Hat Enterprise Linux 7.
- [Standalone repository \(see page 10\)](#): NetQ packages are in their own separate repository.
- New [check and show commands](#) for more validation options.

NetQ 1.1.0 includes [early access](#) support for the following:

- [Facebook Backpack \(see page 93\)](#) chassis
- Extending NetQ with [custom commands \(see page 87\)](#)

For further information regarding bug fixes and known issues present in this release, refer to the [release notes](#).

NetQ Components



NetQ comprises the following components:

- **NetQ Agent**

The back-end Python agent installed on every monitored *node* in the network — including Cumulus Linux switches, Linux bare-metal hosts and virtual machines, or Docker containers. The agent pushes out data to the NetQ Telemetry Server periodically, and when specific **netlink** events occur. The agent monitors the following objects via **netlink**:

- interfaces
- address (IPv4 and IPv6)
- route (IPv4 and IPv6)
- link
- bridge fdb
- IP neighbor (IPv4 and IPv6)

Further, every 15 seconds, it gathers data for the following protocols:

- Bridging protocols (LLDP, STP, MLAG)
- Routing protocols (BGP, OSPF)
- Network virtualization (LNV, VXLAN data plane)
- Docker containers

It also listens to the Docker event stream to monitor Docker containers running on a host and gathers container networking information such as NAT translations, networks and container IP and MAC addresses.

- **NetQ Telemetry Server**

The database/key-value store where all network information sent from NetQ Agents running on the network is collected, aggregated and queried from.

- **NetQ Analysis Engine**

The NetQ Analysis Engine is the backend engine utilized when querying NetQ via the CLI, service console, or notifier. The engine has two parts:

- The **NetQ Agent Command Line Interface**. The NetQ CLI can be used on every node and can be used on the NetQ Telemetry Server through `netq-shell`.
- The **NetQ Notifier**. The notifier runs on the telemetry server. It responds to events pushed by the NetQ Agent, sending alerts to a configured channel, such as Slack, PagerDuty or syslog.

- **NetQ Service Console**

The Service Console provides a browser-based window for accessing the NetQ CLI from anywhere.

Getting Started with NetQ

NetQ is comprised of two main install components: the NetQ Telemetry Server, and the `cumulus-netq` metapackage which gets installed on Cumulus Linux switches. Additionally, for host network visibility and containers, you can install host OS-specific metapackages.

This section walks through the basic install and setup steps for installing and running NetQ on the following supported operating systems:

- Cumulus Linux
- Ubuntu 16.04
- Red Hat Enterprise Linux 7
- CentOS 7



Before you get started, you should review the [release notes](#) for this version.

Contents

This chapter covers ...

- [Install the NetQ Telemetry Server \(see page 9\)](#)
- [Install the NetQ Agent \(see page 10\)](#)
 - [Installing on a Cumulus Linux Switch \(see page 11\)](#)
 - [Installing on an Ubuntu, Red Hat or CentOS Server \(see page 11\)](#)
- [Configuring the NetQ Agent on a Node \(see page 11\)](#)
 - [Configuring the Agent to Use a VRF \(see page 12\)](#)
 - [Configuring the Agent to Communicate over a Specific Port \(see page 12\)](#)
- [Configuring NetQ Notifier on the Telemetry Server \(see page 12\)](#)
- [Example `/etc/cts/netq/netq.yml` Configuration \(see page 15\)](#)

Install the NetQ Telemetry Server

The NetQ Telemetry Server comprises a set of individual Docker containers that each contain a separate service for the `redis` database, the Service Console, the NetQ CLI and NetQ Notifier. It is available in one of two formats:

- VMware ESXi 6.5 virtual machine
- A QCOW/KVM image for use on Ubuntu 16.04 and Red Hat Enterprise Linux 7 hosts



Cumulus Networks recommends the telemetry server is installed on an out-of-band management network to ensure it can monitor in-band network issues without being affected itself. Ideally, you should run the telemetry server on a separate, powerful server for maximum usability and performance. For more information on system requirements, refer to the [How Far Back in Time Can You Travel](#) (see page 71) section.



The NetQ telemetry server containers are completely separate from any containers you may have on the hosts you are monitoring with NetQ. The NetQ containers will not overwrite the host containers and vice versa.

1. Download the NetQ Telemetry Server virtual machine. On the [Downloads](#) page, select *NetQ* from the **Product** menu, then click **Download** for the appropriate hypervisor — KVM or VMware.
2. Import the virtual machine into your [KVM](#) or [VMware](#) hypervisor.
3. Start the NetQ Telemetry Server. There are two default user accounts you can use to log in:
 - The primary username is *admin*, and the default password is *CumulusNetQ!*.
 - The alternate username is *cumulus*, and its password is *CumulusLinux!*.

Once the NetQ Telemetry Server is installed, you need to configure NetQ Notifier.

In addition, if you intend to use NetQ with applications like PagerDuty or Slack, you need to configure those applications to receive notifications from NetQ Notifier.



Note the external IP address of the host where the telemetry server is running, as you need this to correctly configure the NetQ Agent on every node you want to monitor. The telemetry server gets its IP address from DHCP; to get the IP address, run `ifconfig eth0` on the telemetry server.

Install the NetQ Agent

In order to manage a node with NetQ Agent and send notifications with NetQ Notifier, you need to install an OS-specific metapackage on each node. The node can be a:

- Cumulus Linux switch running version 3.3.0 or later
- Server running Red Hat RHEL 7.1, Ubuntu 16.04 or CentOS 7
- Linux virtual machine running one of the above Linux operating systems

The metapackage contains the NetQ Agent and the NetQ command line interface.

Install the metapackage on each node to monitor, then configure the NetQ Agent on the node.



Installing on a Cumulus Linux Switch

1. Edit `/etc/apt/sources.list` and add the following line:

```
cumulus@switch:~$ sudo nano /etc/apt/sources.list
deb http://apps3.cumulusnetworks.com/repos/deb CumulusLinux-3
netq-latest
```

2. Update the local `apt` repository, then install the metapackage on the switch:

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

Installing on an Ubuntu, Red Hat or CentOS Server

To install NetQ on Linux servers running Ubuntu, Red Hat or CentOS, please read the [Host Pack documentation](#).

Configuring the NetQ Agent on a Node

Once you install the NetQ packages and configure the NetQ Telemetry Server, you need to configure NetQ on each Cumulus Linux switch to monitor that node on your network.

1. To ensure useful output, ensure that `NTF` is running.
2. On the host, after you install the NetQ metapackage, restart `rsyslog` so logs are sent to the correct destination:

```
cumulus@switch:~$ sudo systemctl restart rsyslog
```

3. Link the host to the telemetry server you configured above; in the following example, the IP address for the telemetry server host is `198.51.100.10`:

```
cumulus@switch:~$ netq config add server 198.51.100.10
```

This command updated the configuration in the `/etc/cts/netq/netq.yml` file. It also enables the NetQ CLI.

- Restart the `netq` services.

```
cumulus@switch:~$ netq config agent restart
```



If you see the following error, it means you haven't added the telemetry server or the server wasn't configured:

```
cumulus@switch:~$ netq config agent start
Error: Please specify IP address of DB server
```

Configuring the Agent to Use a VRF

If you want the NetQ Agent to communicate with the telemetry server only via a [VRF](#), including a [management VRF](#), you need to specify the VRF name when configuring the NetQ Agent. For example, if the management VRF is configured and you want the agent to communicate with the telemetry server over it, configure the agent like this:

```
cumulus@switch:~$ netq config add server 198.51.100.10 vrf mgmt
```

Then you need to stop and disable the `netqd` service, then enable and start the service in the VRF:

```
cumulus@switch:~$ sudo systemctl stop netqd.service
cumulus@switch:~$ sudo systemctl disable netqd.service
cumulus@switch:~$ sudo systemctl enable netqd@mgmt.service
cumulus@switch:~$ sudo systemctl start netqd@mgmt.service
```

Configuring the Agent to Communicate over a Specific Port

By default, NetQ uses port 6379 for communication between the telemetry server and NetQ Agents. If you want the NetQ Agent to communicate with the telemetry server via a different port, you need to specify the port number when configuring the NetQ Agent like this:

```
cumulus@switch:~$ netq config add server 198.51.100.10 port 7379
```

Configuring NetQ Notifier on the Telemetry Server

NetQ Notifier listens to events from the telemetry server database. When NetQ Notifier is running on the NetQ Telemetry Server, it sends out alerts. NetQ Notifier runs on the NetQ Telemetry Server only; the NetQ Agents on the nodes only communicate with it.

You need to configure two things for NetQ Notifier:

- The events for which you want to receive notifications/alerts, like sensors or BGP session notifications.
- The integrations for where to send those notifications; by default, they are `rsyslog`, PagerDuty and Slack.

NetQ Notifier sends out alerts based on the configured log level, which is one of the following:

- `debug`: Used for debugging-related messages.
- `info`: Used for informational, high-volume messages.
- `warning`: Used for warning conditions.
- `error`: Used for error conditions.

The default log level setting is *info*, so NetQ Notifier sends out alerts for info, warning and error conditions.

By default, all notifications/alerts are enabled, and logged in `/var/log/netq-notifier.log`. You only need to edit the notifications if there is something you don't want to monitor.

NetQ Notifier is already integrated with `rsyslog`. To integrate with PagerDuty or Slack, you need to specify some parameters.

To configure alerts and integrations on the NetQ Telemetry Server:

1. Stop and disable the following services on the telemetry server, as they are not necessary (and you will enable the `netq-notifier` service in a later step):

```
cumulus@ts:~$ sudo systemctl stop cts-notifier-watch.timer cts-  
notifier-watch cts-notifier  
cumulus@ts:~$ sudo systemctl disable cts-notifier-watch.timer  
cts-notifier-watch cts-notifier
```

2. As the sudo user, open `/etc/cts/netq/netq.yml` in a text editor.

3. Configure the following in the `/etc/cts/netq/netq.yml` file:

- Change the log level: If you want a more restrictive level than info.
- Configure application notifications: To customize any notifications, uncomment the relevant section under **netq-notifier Configurations** and make changes accordingly.
- Configure PagerDuty and Slack integrations. You can see where to input the information for these integrations in the [example netq.yml file \(see page 15\)](#) below.
 - For PagerDuty, enter the API access key (also called the [authorization token](#)) and the [integration key](#) (also called the `service_key` or `routing_key`).
 - For Slack, enter the webhook URL. To get the webhook URL, in the Slack dropdown menu, click **Apps & integrations**, then click **Manage > Custom Integrations > Incoming WebHooks > select Add Configuration > select the channel to receive the notifications such as #netq-notifier in the Post to Channel dropdown > then click Add Incoming WebHook integration**. The URL produced by Slack looks similar to the one pictured below:

Webhook URL

`https://hooks.slack.com/services/sometext/moretext/evenmoretext`

Copy the URL from the **Webhook URL** field into the `/etc/cts/netq/netq.yml` file under the **Slack Notifications** section. Uncomment the lines in the sections labeled *netq-notifier*, *notifier-integrations* and *notifier-filters*, then add the webhook URL value provided by Slack:

```
netq-notifier:
  log_level: info

...

notifier-integrations:
  - name: notifier-slack-channel-1
    type: slack
    webhook: "https://hooks.slack.com/services/sometext/moretext/evenmoretext"
    severity: INFO,
    tag: "@netqts-sys"

...

notifier-filters:
  - name: default
    rule:
      output:
        - ALL
```

When you are finished editing the file, save and close it.

4. Stop then start the NetQ Notifier daemon to apply the new configuration:

```
cumulus@netq-appliance:~$ docker-compose -p "netq" -f
"${CFG_DIR}/docker/netq-base-compose.yml" stop netq-notifier
cumulus@netq-appliance:~$ docker-compose -p "netq" -f
"${CFG_DIR}/docker/netq-base-compose.yml" start netq-notifier
```



If your webhook does not immediately send a message to your channel, look for errors in syntax. Check the log file located at `/var/log/docker/cts-netq-notifier.log`.

Example `/etc/cts/netq/netq.yml` Configuration

In the following sample `/etc/cts/netq/netq.yml` file on the NetQ Telemetry Server itself. Note that the `netq.yml` looks different on a NetQ switch or host (for example, the backend server IP address and port would be uncommented and listed).

```
cumulus@netq-appliance:~$ cat /etc/cts/netq/netq.yml
## Netq configuration File.
## Configuration is also read from files in /etc/netq/config.d/ and
have
## precedence over config in /etc/cts/netq/netq.yml.

## ----- Common configurations -----

## Backend Configuration for all netq agents and apps on this host.
##
#backend:
#  server:
#  port: 6379
## ----- netq-agent configurations -----

## Netq Agent Configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
#netq-agent:
#  log_level: info
## Docker Agent Configuration
##
## docker_enable: Enable Docker monitoring. Default is True.
## docker_poll_period: Docker poll period in secs. Default is 15 secs.
##
#docker:
#  enable: true
#  poll_period: 15
```

```
## ----- netq configurations -----
## Netq configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
#netqd:
# log_level: info
## ----- netq-notifier configurations -----

## Netq Notifier configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
#netq-notifier:
# log_level: info
## Slack Notifications
##
## NetQ Notifier Filter Configuration
##
## NetQ Notifier sends notifications to integrations(syslog, slack or
pagerduty)
## based on the events that are happening across the network.
## Notifications are generated based on the filters that have been
specified in
## "notifier-filters". NetQ Agents generate an event when something
interesting
## happens on the host (switch or server) its running on. The
Notifier is always
## listening for these events and once it receives an event, it makes
it go
## through a set of filters.
##
## A filter has 3 stages:
## a) Rule: Defines a set of conditions against which an incoming
event is
## matched. Input to this stage is an incoming event and the event is
sent to
## the next stage if there is a match. If there is a match, the event
## information is passed to the action stage. The rule is a
dictionary of
## key-value pairs, where the "key" is an item associated with the
event and
## "value" is the value of that item,
## e.g. type: Link
##     hostname: leaf-01
##     ifname: swp1
## The Default rule, if none is specified or if it is empty, is to
always assume
## a match.
## Notifier-filter rules are matched sequentially and we stop only
when a match
```



```
## is found. You can make the notifier continue matching filters even
if a match
## is found, by adding "terminate_on_match: False" to the filter.
Values
## specified in the rule are matched with those received in a event
using python
## regular expressions https://docs.python.org/2/library/re.html
## We can also match for message severity and print messages only if
it is above
## the given severity. Message severity levels are: INFO, WARNING,
ERROR and
## CRITICAL in ascending order.
##
## b) Action: action to perform if the "rule" is matched. The action
stage
## take the event provided by the "rule" stage and generates a message
## dictionary with a message and its severity. Multiple actions can be
## prescribed in the "action" list. "action" is typically a python
function that
## is provided with the tool or a custom one written by the user. If
no action
## is provided, we default to a generic handler which looks at the
event and
## based on the event runs the relevant notification function.
##
## c) output: This stage takes the message dictionary provided by the
action
## stage and sends the message and severity to the right integration
to display
## the message. If the output is None the message is not sent to any
integration
## or syslog. If output is empty, the message is sent only to syslog.
Else the
## message is sent to the list of integrations specified in the
output list and
## syslog. If ALL is specified, the message is sent to all
integrations.
## Integrations are defined in notifier-integrations.
##
## The config file comes with the following default filter:
##
## notifier-filters:
## - name: default
##   rule:
##   output:
##     - ALL
##
## which is an empty rule, empty action and output to all. This
defaults to
## match all rules and then perform the default action which is to
run the
## generic handler mentioned in the action stage above.
```

```
##
## NetQ Integration Configuration
##
## The integrations refer to the external tool where you would like
## to receive
## the notification. An integration is added as a list element to
## "notifier-integrations". Each integration must have a "name" and
## "type".
## Severity is optional and lets you send messages above that level
## to the
## integration. Allowed values are: INFO, WARNING, ERROR, CRITICAL in
## increasing
## order. Currently allowed "type" are "slack" and "pagerduty". You
## can define
## multiple slack or PD integrations.
##
## For Slack integration, along with a name and "type: slack", you
## also need to
## also provide the Incoming Webhook of the channel. The webhook URL
## for your
## channel can be found or created in Slack at:
##   Apps -> Custom Integrations -> Incoming Webhooks.
## Tags are optional and are strings that are attached to the end of
## the
## notification message.
## E.g.
## notifier-integrations:
## - name: notifier-slack-channel-1
##   type: slack
##   webhook: "https://<slack-webhook1>"
##   severity: INFO,
##   tag: "@slack-tag1"
##
## For pagerDuty, along with name and "type: pagerduty", you also
## need to
## provide the "api_access_key" and "api_integration_key" from
## Pagerduty.
## A unique API Access Key which can be created on your PagerDuty
## website at:
## Configuration -> API Access -> Create New API Key
## An 'Integration Key' can be created/found on your PagerDuty
## website at:
## Configuration -> Services -> Add New Service -> New Integration ->
##   Select Integration Type as 'Use our API directly: Events API v2'.
## E.g. pagerduty integration along with slack
## notifier-integrations:
## - name: notifier-slack-channel-1
##   type: slack
##   webhook: "https://<slack-webhook1>"
##   severity: INFO
##   tag: "@slack-tag1"
## - name: notifier-pagerduty
```

```
# type: pagerduty
# severity: WARNING
# api_access_key: <API Key>
# api_integration_key: <API Integration Key>
##
## Customizing Notifications
## Here are some examples on how to customize your notifications:
##
## a) Filter notifications to integrations (Slack or PD) based on
Severity,
## i.e., WARNING to PD, INFO to Slack
# notifier-integrations:
# - name: notifier-slack-channel-1
#   type: slack
#   webhook: "https://<slack-webhook1"
#   severity: INFO <==== Set the severity type here
#   tag: "@slack-tag1"
# - name: notifier-pagerduty
#   type: pagerduty
#   severity: WARNING <==== Set the severity type here
#   api_access_key: "<API Key>"
#   api_integration_key: "<API Integration Key>"
#
##
## b) Drop all notifications coming from a switch/host say, leaf-01
# notifier-filters:
# - name: leaf-01 drop
#   rule:
#     hostname: leaf-01
#     output:
#       - None
# - name: default
#   rule:
#     output:
#       - ALL
##
## c) Drop all notifications coming from switches whose name starts
with leaf
# notifier-filters:
# - name: leaf drop
#   rule:
#     hostname: "leaf-.*"
#     output:
#       - None
# - name: default
#   rule:
#     output:
#       - ALL
##
## d) Drop all notifications coming from a particular link, e.g. leaf-
01 swp1
# notifier-filters:
```

```
# - name: leaf-01 swp1 drop
#   rule:
#     type: Link
#     hostname: leaf-01
#     ifname: swp1
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
##
## e) Send BGP Session state notifications to particular slack channel
## (slack-channel-BGP), rest to another one (slack-channel-catchall)
# notifier-filters:
# - name: BGP slack channel
#   rule:
#     type: BgpSession
#   output:
#     - slack-channel-BGP
# - name: default
#   rule:
#   output:
#     - slack-channel-catchall
##
## f) Send BgpSession notifications based on severity to different
slack channels
# notifier-filters:
# - name: BGP severity slack channel
#   rule:
#     type: BgpSession
#     severity: WARNING
#   output:
#     - slack-channel-BGP-info
# - name: default
#   rule:
#   output:
#     - slack-channel-catchall
##
## g) Drop all temperature related alerts
# notifier-filters:
# - name: temp drop
#   rule:
#     type: Temp
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
notifier-filters:
  - name: default
```

```
rule:
  output:
    - ALL
```

Upgrading NetQ

This section covers the process for upgrading NetQ. The upgrade process involves upgrading each of the various components of NetQ (the NetQ Telemetry Server, and both the Host and Cumulus Linux Agents), and then connecting the upgraded NetQ Telemetry Server to the network.



Cumulus Networks recommends only upgrading NetQ during a maintenance window.



Events generated during the upgrade process will not be available in the database. Once the upgrade process is complete, the agents will re-sync with the current state of the Host or Cumulus Linux switch with the Telemetry server.

Before upgrading NetQ, consider the following information:

- The minimum supported Cumulus Linux version for NetQ 1.1 is 3.3.0.
- NetQ 1.0 can be upgraded to NetQ 1.1 without upgrading Cumulus Linux.



NetQ 1.1 will be supported on Cumulus Linux 3.4.0 and future releases until the next NetQ version is released.

Contents

This chapter covers ...

- Upgrade the NetQ Telemetry Server (see page 21)
- Upgrade the NetQ Agents (see page 22)
 - Cumulus Linux (see page 22)
 - Ubuntu 16.04 (see page 23)
 - Red Hat Enterprise Linux 7 / CentOS 7 (see page 23)
- Connect the NetQ Telemetry Server to the Network (see page 24)


Upgrade the NetQ Telemetry Server




To install a new instance of NetQ, refer to the [Getting Started with NetQ](#) documentation.

1. Back up the current NetQ Telemetry Server data. For instructions, refer to the [NetQ Backup](#) documentation.

2. Shutdown the connectivity from the agents to the current NetQ Telemetry Server.

 This step is required to ensure agents don't attempt to communicate with the Telemetry Server during the maintenance window.


3. Shutdown the current NetQ Telemetry Server.
4. Start the new NetQ Telemetry Server.
5. Restore the data to the new NetQ Telemetry Server. For instructions, refer to the [NetQ Backup](#) documentation.

 This step can be skipped, if there is no desire to retain the previous data. NetQ agents will re-populate the current data once they connect to the new NetQ Telemetry Server.

6. Validate that the telemetry server is up and running:

```
cumulus@switch:~$ cat /etc/app-release  
APPLIANCE_VERSION=1.1.0
```

 Cumulus Networks recommends that the NetQ Agents remain disconnected from the NetQ Telemetry Server until they have been upgraded to NetQ 1.1 as well.

 The NetQ Notifier configuration syntax has changed between versions 1.0 and 1.1. Examples of the new syntax are in the [getting started guide](#) (see page 12).

Upgrade the NetQ Agents

Follow the steps for the relevant OS below to upgrade the NetQ Agents:

Cumulus Linux

1. Open the `/etc/apt/sources.list` file in a text editor.
2. Add the following line, and save the file:

```
deb      https://apps3.cumulusnetworks.com/repos/deb CumulusLinux-  
3 netq-1.1
```



3. Install the `cumulus-netq` metapack and its components:

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

Ubuntu 16.04

1. Use the `wget` tool to retrieve the public key:

```
cumulus@switch:~$ wget -O - https://apps3.cumulusnetworks.com/setup/cumulus-apps-deb.pubkey | sudo apt-key add -
```

2. Open the `/etc/apt/sources.list` file in a text editor.
3. Add the following line, and save the file:

```
deb https://apps3.cumulusnetworks.com/repos/deb xenial netq-1.1
```

4. Install the `cumulus-netq` metapack and its components:

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

Red Hat Enterprise Linux 7 / CentOS 7

1. Import the public key:

```
cumulus@switch:~$ sudo rpm --import https://apps3.cumulusnetworks.com/setup/cumulus-host-el.pubkey
```

2. Open `/etc/yum.repos.d/cumulus-host-el.repo` in a text editor.

3. Define the repository source, and save the file:

```
[cumulus-arch]
name=Cumulus Packages for RHEL
baseurl=https://apps3.cumulusnetworks.com/repos/rpm/el
/$releasever/netq-1.1/$basearch
gpgcheck=1
enabled=1

[cumulus-noarch]
name=Architecture-independent Cumulus packages for RHEL
baseurl=https://apps3.cumulusnetworks.com/repos/rpm/el
/$releasever/netq-1.1/noarch
gpgcheck=1
enabled=1

[cumulus-src]
name=Cumulus source packages for RHEL
baseurl=https://apps3.cumulusnetworks.com/repos/rpm/el
/$releasever/netq-1.1/src
gpgcheck=1
enabled=1
```

4. Install the `cumulus-netq` metapack and its components:

```
cumulus@switch:~$ sudo yum install cumulus-netq
```

Connect the NetQ Telemetry Server to the Network

1. Once the NetQ Telemetry Server and NetQ agents have been upgraded, connect the NetQ Telemetry Server to the network. For more information, refer to the [Getting Started](#) documentation.
2. Verify the NetQ Agents are OK, and running NetQ 1.1. The output should show the version as 1.1-c13u5 for NetQ 1.1:

```
cumulus@switch:~$ netq show agents
```


Getting to Know NetQ

After you've installed NetQ, running `netq example` gives you some pointers as to how it helps you solve issues across your network.

```
cumulus@oob-mgmt-server:~$ netq example
  check           :   Perform fabric-wide checks
  find-duplicates  :   Find Duplicate IP or MAC
  find-origin     :   Find Origin of Route/MAC
  regexp          :   Using Regular Expressions
  resolve         :   Annotate input with names and interesting info
  startup         :   NetQ Quickstart
  trace           :   Control Path Trace
```

```
cumulus@switch:~$ netq example trace
```

Control Path Trace

=====

Commands

=====

```
netq trace <mac> [vlan <1-4096>] from <hostname> [vrf <vrf>]
[around <text-time>] [json]
netq trace <ip> from (<hostname>|<ip-src>) [vrf <vrf>] [around
<text-time>] [json]
```

Usage

=====

`netq trace` provides control path tracing (no real packets are sent) from a specified source to a specified destination. The trace covers complete end-to-end path tracing including bridged, routed and Vxlan overlay paths. ECMP is supported as well as checking for forwarding loops, MTU consistency across all paths, and VLAN consistency across all paths. The trace also covers that the path from dest to src also exists on each hop.

```
cumulus@torc-12:~$ netq trace 27.0.0.22 from 27.0.0.21
torc-12 -- torc-12:swp3 -- spine-1:swp5 -- torc-21:lo
        -- torc-12:swp4 -- spine-2:swp5 -- torc-21:lo
```

When tracing data, only the egress information is shown as this information is gathered by looking at the routing table. In this case, there are two paths (one through spine01 and one through spine02) because the environment is leveraging equal cost routing.

You can trace by MAC as well:

```
cumulus@leaf1:~$ netq trace 00:02:00:00:00:02 vlan 1009 from leaf1
leaf1 -- leaf1:sw_clag200 -- spine1:sw_clag300 -- edge2
                                -- spine1:sw_clag300 -- edge1:VlA-1
      -- leaf1:sw_clag200 -- spine2:sw_clag300 -- edge1:VlA-1
                                -- spine2:sw_clag300 -- edge2
cumulus@leaf1:~$
```

Legend

=====

Any errors are shown in red. Bridged paths are always in WHITE, routed paths in GREEN, the VTEPs are shown in BLUE. A node in error is shown in RED.

And `netq help` shows you information about specific commands.

```
cumulus@switch:~$ netq help show interfaces
Commands:
  netq <hostname> show docker container adjacent [interfaces <remote-physical-interface>] [around <text-time>] [json]
  netq [<hostname>] show docker container name <container-name> adjacent [interfaces <remote-physical-interface>] [around <text-time>] [json]
  netq [<hostname>] show interfaces [around <text-time>] [count] [json]
  netq <hostname> show interfaces <remote-interface> [around <text-time>] [count] [json]
  netq [<hostname>] show interfaces type (bond|bridge|eth|loopback|macvlan|swp|vlan|vrf|vxlan) [around <text-time>] [count] [json]
  netq [<hostname>] show interfaces changes [between <text-time> and <text-endtime>] [json]
  netq <hostname> show interfaces <remote-interface> changes [between <text-time> and <text-endtime>] [json]
  netq [<hostname>] show interfaces type (bond|bridge|eth|loopback|macvlan|swp|vlan|vrf|vxlan) changes [between <text-time> and <text-endtime>] [json]
```



Getting Information about Network Hardware

You can get information about the hardware on the nodes in the network with `netq show inventory` command. You can get details about the ASIC, motherboard, CPU, license, memory, storage, operating system. To see a shorter summary, use the `brief` option:

```
netq@446c0319c06a:/$ netq show inventory brief
```

Node	Switch	OS	CPU	ASIC	Ports
exit01	VX	Cumulus Linux	x86_64	N/A	N/A
exit02	VX	Cumulus Linux	x86_64	N/A	N/A
leaf01	VX	Cumulus Linux	x86_64	N/A	N/A
leaf02	VX	Cumulus Linux	x86_64	N/A	N/A
leaf03	VX	Cumulus Linux	x86_64	N/A	N/A
leaf04	VX	Cumulus Linux	x86_64	N/A	N/A
server01	N/A	Ubuntu	x86_64	N/A	N/A
server02	N/A	Ubuntu	x86_64	N/A	N/A
server03	N/A	Ubuntu	x86_64	N/A	N/A
server04	N/A	Ubuntu	x86_64	N/A	N/A
spine01	VX	Cumulus Linux	x86_64	N/A	N/A
spine02	VX	Cumulus Linux	x86_64	N/A	N/A

Using the NetQ Shell on the NetQ Telemetry Server

If you need to run `netq` commands from the telemetry server, use the NetQ shell. While most other Linux commands can work from this shell, Cumulus Networks recommends you only run `netq` commands here.

```
cumulus@netq-appliance:~$ netq-shell
[<Container: a017716433>]
Welcome to Cumulus (R) Linux (R)

For support and online technical documentation, visit
http://www.cumulusnetworks.com/support

The registered trademark Linux (R) is used pursuant to a sublicense
from LMI, the exclusive licensee of Linux Torvalds, owner of the mark
on a worldwide basis.

TIP: Type `netq` to access NetQ CLI.
netq@017716433d5:/$ netq show agents
```

Node	Status	Sys Uptime	Agent Uptime
exit01	Fresh	3h ago	3h ago
exit02	Fresh	3h ago	3h ago
leaf01	Fresh	3h ago	3h ago
leaf02	Fresh	3h ago	3h ago
server01	Fresh	3h ago	3h ago
server02	Fresh	3h ago	3h ago
server03	Fresh	3h ago	3h ago
server04	Fresh	3h ago	3h ago
...			

Using the netq resolve Command

Linux commands can be piped through NetQ with the `netq resolve` command, in order to provide more contextual information and colored highlights. For example, to show routes installed by the kernel, you would run the `ip route show proto kernel` command:

```
cumulus@leaf01:~$ ip route show proto kernel
3.0.2.128/26 dev VlanA-1.103 scope link src 3.0.2.131
3.0.2.128/26 dev VlanA-1-103-v0 scope link src 3.0.2.129
3.0.2.192/26 dev VlanA-1.104 scope link src 3.0.2.195
3.0.2.192/26 dev VlanA-1-104-v0 scope link src 3.0.2.193
3.0.3.0/26 dev VlanA-1.105 scope link src 3.0.3.3
3.0.3.0/26 dev VlanA-1-105-v0 scope link src 3.0.3.1
3.0.3.64/26 dev VlanA-1.106 scope link src 3.0.3.67
3.0.3.64/26 dev VlanA-1-106-v0 scope link src 3.0.3.65
169.254.0.8/30 dev peerlink-1.4094 scope link src 169.254.0.10
192.168.0.0/24 dev eth0 scope link src 192.168.0.15
```

You can enhance the output to display the node names and interfaces by piping the output through `netq resolve` so the output looks like this:



```
cumulus@leaf01:~$ ip route show proto kernel | netq resolve
10.0.0.0/22 (
multiple:
) dev eth0 scope link src 10.0.0.165 (
cel-smallxp-13
:
eth0
)
3.0.2.128/26 (
server02
:
torbond1.103
) dev VlanA-1.103 scope link src 3.0.2.131 (
leaf02
:
VlanA-1.103
)
3.0.2.128/26 (
server02
:
torbond1.103
) dev VlanA-1-103-v0 scope link src 3.0.2.129 (
leaf02
:
VlanA-1-103-v0
)
3.0.2.192/26 (
leaf02
:
VlanA-1-104-v0
) dev VlanA-1.104 scope link src 3.0.2.195 (
leaf02
:
VlanA-1.104
)
3.0.2.192/26 (
leaf02
:
VlanA-1-104-v0
) dev VlanA-1-104-v0 scope link src 3.0.2.193 (
leaf02
:
VlanA-1-104-v0
```

```

)
3.0.3.0/26 (
server01
:
torbond1.105
) dev VlanA-1.105 scope link src 3.0.3.3 (
leaf02
:
VlanA-1.105
)
3.0.3.0/26 (
server01
:
torbond1.105
) dev VlanA-1-105-v0 scope link src 3.0.3.1 (
leaf02
:
VlanA-1-105-v0
)
3.0.3.64/26 (
server02
:
torbond1.106
) dev VlanA-1.106 scope link src 3.0.3.67 (
leaf02
:
VlanA-1.106
)
3.0.3.64/26 (
server02
:
torbond1.106
) dev VlanA-1-106-v0 scope link src 3.0.3.65 (
leaf01
:
VlanA-1-106-v0
)
169.254.0.8/30 (
leaf02
:
peerlink-1.4094
) dev peerlink-1.4094 scope link src 169.254.0.10 (
leaf02
:
peerlink-1.4094

```

```

)
192.168.0.0/24 (
server02
:
eth0
) dev eth0 scope link src 192.168.0.15 (
leaf01
:
eth0
)

```

Sample Commands for Various Components

NetQ provides network validation for the entire stack, providing algorithmic answers to many questions, both simple and intractable, that pertain to your network fabric.

Component	Problem	Solution
Host	Where is this container located? Open ports? What image is being used? Which containers are part of this service? How are they connected?	netq show docker container netq show docker container service
Overlay	Is my overlay configured correctly? Can A reach B? Is my control plane configured correctly?	netq check show vxlan netq check evpn Inv netq trace overlay
L3	Is OSPF working as expected? Is BGP working as expected? Can IP A reach IP B?	netq check show ospf netq check show bgp netq trace l3
L2	Is MLAG configured correctly? Is there an STP loop? Is VLAN or MTU misconfigured? How does MAC A reach B?	netq check show clag netq show stp netq check show vlan netq check mtu netq trace L2
OS	Are all switches licensed correctly? Do all switches have NetQ agents running?	netq check license netq check show agents

Component	Problem	Solution
Interfaces	Is my link down? Are all bond links up? What optics am I using? What's the peer for this port? Which ports are empty? Is there a link mismatch? Are links flapping?	netq show check interfaces
Hardware	Have any components crashed? What switches do I have in the network?	netq check sensors netq show sensors all netq show inventory brief

Understanding Timestamps in NetQ

Every event or entry in the NetQ database is stored with a timestamp of when the event was captured by the NetQ agent on the node. This timestamp is based on time on the node where the agent is running, and is pushed in UTC format. Thus, it is important to ensure that all nodes are [NTP synchronized \(see page 55\)](#). Without this NTP sync, events may be displayed out of order or, worse, not displayed when looking for events that occurred at a particular time or within a time window.

Interface state, IP addresses, routes, ARP/ND table (IP neighbor) entries and MAC table entries carry a timestamp that represents the time the event happened (such as when a route is deleted or an interface comes up) — *except* the first time the NetQ agent is run. If the network has been running and stable when a NetQ agent is brought up for the first time, then this time reflects when the agent was started. Subsequent changes to these objects are captured with an accurate time of when the event happened.

Data that is captured and saved based on polling, and just about all other data in the NetQ database as of NetQ 1.1, including control plane state (such as BGP or MLAG), has a timestamp of when the information was *captured* rather than when the event *actually happened*, though NetQ does try to compensate for it if the data extracted provides additional information to compute a more precise time of the event; for example, BGP uptime can be used to determine when the event actually happened in conjunction with the timestamp.

When retrieving the timestamp, JSON output always returns the time in microseconds since the epoch. Non-JSON output displays how long ago in the past the event occurred. The closer the event is to the present, the more granular is the time shown. For example, if an event happened less than an hour ago, NetQ displays the information with a timestamp with microseconds of granularity. However, the farther you are from the event, this granularity is coarser. This is shown in the two outputs below:


```
cumulus@leaf01:mgmt-vrf:~$ netq leaf01 show interfaces swp51
Matching link records are:
Node          Interface      Type      State
Details                               Last Changed
-----
leaf01        swp51          swp       up      LLDP: spine01:
swp1,         2h ago
MTU: 1500

cumulus@leaf01:mgmt-vrf:~$ netq leaf01 show interfaces swp52
Matching link records are:
Node          Interface      Type      State
Details                               Last Changed
-----
leaf01        swp52          swp       up      LLDP: spine02:
swp1,         2h ago
MTU: 1500
```



Remember that the time stored in the database is the one with microseconds since the epoch and is what is returned (as a float) in the JSON output.

One more important point to note. If a NetQ agent is restarted on a node, it doesn't update all the timestamps for existing objects to this new restart time. Those times are preserved to those at the agent's original start time, unless the node is rebooted between the agent stopping and restarting; in which case, the time is once again the time of agent restart.

NetQ Service Console

The NetQ Telemetry Server provides access to the NetQ Service Console, a graphical user interface (GUI) for NetQ. The service console in turn provides terminal access to any node in the fabric.



The Cumulus NetQ Service Console utilizes elements of Portainer. You can read the Portainer license file [here](#).

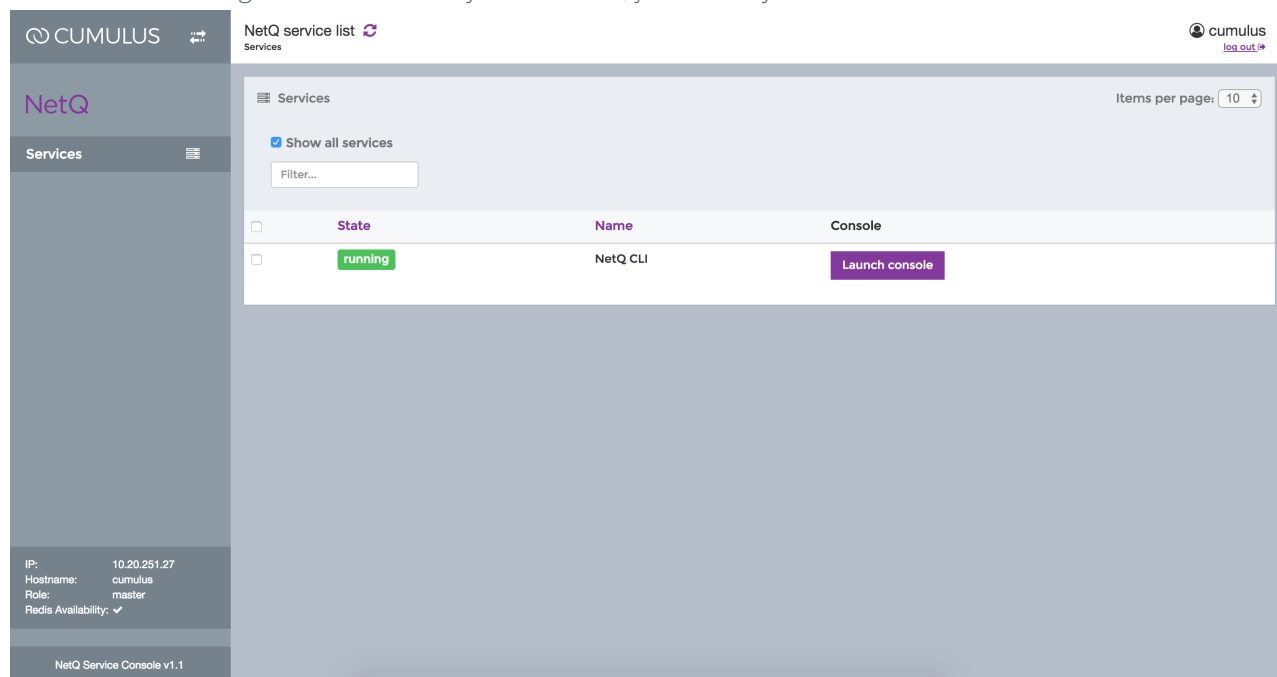
Contents

This chapter covers ...

- [Connecting to the Service Console \(see page 34\)](#)
- [Getting Service Console Information \(see page 35\)](#)
- [Accessing the NetQ Command Line \(see page 35\)](#)

Connecting to the Service Console

To connect to the service console, open a browser, and go to the IP address of the [telemetry server \(see page 8\)](#). You are prompted to log in with the username and password for the service console. You can use the same credentials that you use to access the telemetry server VM. The service console user accounts are managed in the telemetry server itself, just like any Linux user account.



The screenshot shows the NetQ Service Console web interface. The left sidebar contains the Cumulus logo, the NetQ logo, and a 'Services' menu item. The main content area is titled 'NetQ service list' and shows a table of services. The table has columns for 'State', 'Name', and 'Console'. One service is listed: 'NetQ CLI' with a state of 'running' and a 'Launch console' button. The bottom of the interface shows system information: IP: 10.20.251.27, Hostname: cumulus, Role: master, Redis Availability: ✓, and NetQ Service Console v1.1.

State	Name	Console
running	NetQ CLI	Launch console

Getting Service Console Information

The lower lefthand corner of the service console window displays information about



- **IP:** The IP address of the telemetry server VM. In the default configuration, the IP field is empty. To have this field display the IP address, edit `/etc/cts/redis/host.conf` and set the `HOST_IP` variable to the telemetry server's IP address, then restart the `netq-gui` service with `sudo systemctl restart netq-gui.service`.
- **Hostname:** The hostname of the telemetry server VM. If you change the hostname and don't restart the `netq-gui` service, the hostname won't change in the service console.
- **Role:** The role that the NetQ database is in, which currently can be *master* only, as there is only one Redis server, which is the primary.
- **Redis availability:** Indicates whether or not the Redis database on the telemetry server VM is reachable.

Accessing the NetQ Command Line

The service console provides access to a standard Bash shell, so you can run NetQ commands — or any Linux command — directly on a given node.



The console is connected to the NetQ CLI container within the telemetry server; it is not connected to the shell of the telemetry server itself. As such, the `netq-shell` command does not work in the console; it is intended to run regular NetQ commands.

In the Services window of the console, click **Launch console**.



Cumulus NetQ 1.1.0 User Guide

The screenshot shows the Cumulus NetQ service console interface. On the left, there's a sidebar with the Cumulus logo and 'NetQ' text. Below it, 'Services' is listed. At the bottom left, system information is displayed: IP: 10.20.251.27, Hostname: cumulus, Role: master, and Redis Availability: ✓. The main console area has a title bar 'NetQ service console' with a 'log out' link. Below the title bar is a 'Console' tab and a 'Back to Services' link. The console content shows a welcome message: 'Welcome to Cumulus (R) NetQ Command Line Interface', a tip: 'TIP: Type `netq help` to get started.', and a prompt: 'root@cumulus:/#'.

You can run any NetQ commands within the console, such as `netq show agents`:

The screenshot shows the Cumulus NetQ service console interface with the 'netq show agents' command executed. The output is a table with the following columns: Hostname, Status, NTP Sync, Version, Sys Uptime, Agent Uptime, and Reinitialize time. The table lists various network devices and their status.

Hostname	Status	NTP Sync	Version	Sys Uptime	Agent Uptime	Reinitialize time
noc-pr	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:27s	13h:36m:16s	7h:16m:16s
noc-se	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:26s	13h:36m:12s	7h:16m:13s
spine-1	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:28s	13h:36m:6s	7h:16m:0s
spine-2	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:28s	13h:36m:1s	7h:16m:16s
tor-1	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:26s	13h:35m:38s	7h:16m:1s
tor-2	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:25s	13h:35m:34s	7h:16m:5s
torc-11	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:38s	13h:35m:56s	7h:16m:9s
torc-12	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:37s	13h:35m:52s	7h:15m:58s
torc-21	Rotten	-	-	-	-	-
torc-22	Fresh	yes	1.1-cl3u5-1504544477.b01a48d	13h:38m:36s	13h:35m:43s	7h:16m:8s

When you're finished with the session, click **Back to Services** to close the console.

Taking Preventative Steps with Your Network

NetQ provides quality assurance capabilities to detect erroneous or undesired network configurations before the changes are rolled into production. NetQ can be used to test existing or design topologies, validate configuration changes, and review the state of the network in real time, allowing it to integrate effectively with CI/CD environments. NetQ commands can also be run in an automation tool; depending on the outcome of the automation tests, the script can either continue the deployment, or roll back the changes until the issues are addressed.

In addition, [NetQ Virtual \(see page 83\)](#) provides users with a Cumulus VX topology to serve as a virtual representation of your production network; once the network is verified in NetQ Virtual, the topology can then be rolled into production.

PETE: Just about every output on this page needs to be updated. The output of check, show etc have all changed.

Contents

This chapter covers ...

- [netq check and netq show \(see page 37\)](#)
 - [netq show agents \(see page 38\)](#)
- [Using NetQ with Automation \(see page 39\)](#)
- [Using NetQ Virtual \(see page 40\)](#)

netq check and netq show

The `netq check` and `netq show` commands validate network state before and after configuration changes. Based on results returned by NetQ, you or your automation script can either roll back the configuration change or continue deploying it:

```
cumulus@leaf01:~$ netq check
agents      : netq agent
bgp         : BGP info
clag        : Multi-chassis LAG (CLAG) info
license     : License
lnv         : Lightweight Network Virtualization info
mtu         : Link MTU
ospf        : OSPF info
sensors     : Temperature/Fan/PSU sensors
vlan        : VLAN
vxlan       : VxLAN dataplane info
```

Here are some example check commands:



```
cumulus@leaf01:~$ netq check agents
Checked nodes: 25, Rotten nodes: 0
```

NetQ check enables users to review the state of the network at specific moments in time by specifying the `around text-time` option.

```
cumulus@leaf01:~$ netq check bgp vrf DataVrf1081
Total Nodes: 25, Failed Nodes: 1, Total Sessions: 52 , Failed
Sessions: 0
```

```
cumulus@leaf01:~$ netq check lnv around 10m
Checked Nodes: 9, Warning Nodes: 0, Failed Nodes: 0
```

```
cumulus@leaf01:~$ netq check sensors around 14m
Total Nodes: 25, Failed Nodes: 0, Checked Sensors: 221, Failed
Sensors: 0
```

The `netq show` command displays a wide variety of content from the network:

```
cumulus@leaf01:~$ netq show
agents      : netq agent
bgp         : BGP info
changes     : How this information has changed with time
clag        : Multi-chassis LAG (CLAG) info
docker      : Docker Info
interfaces  : Network interface
inventory   : Inventory information
ip          : IPv4 related info
ipv6        : IPv6 related info
lldp        : LLDP based neighbor info
lnv         : Lightweight Network Virtualization info
macs        : Mac table entries
sensors     : Temperature/Fan/PSU sensors
services    : System services
```

netq show agents

To get the health of the NetQ agents running in the fabric, run `netq show agents`. A *Fresh* status indicates the agent is running as expected. The agent sends a heartbeat every 30 seconds, and if 3 consecutive heartbeats are missed, its status changes to *Rotten*.

```
cumulus@leaf01:~$ netq show agents
```

Node	Status	Sys Uptime	Agent Uptime
-----	-----	-----	-----
leaf01	Fresh	2h ago	2h ago
leaf02	Fresh	2h ago	2h ago
leaf03	Fresh	2h ago	2h ago
leaf04	Fresh	2h ago	2h ago
oob-mgmt-server	Fresh	2h ago	2h ago
server01	Fresh	2h ago	2h ago
server02	Fresh	2h ago	2h ago
server03	Fresh	2h ago	2h ago
server04	Fresh	2h ago	2h ago
spine01	Fresh	2h ago	2h ago
spine02	Fresh	2h ago	2h ago

Using NetQ with Automation

Using NetQ for preventative care of your network pairs well with automation scripts and playbooks to prevent errors on your network before deploying the configuration to production.

NetQ works with Ansible, Chef and Puppet.

For example, you can use NetQ in your Ansible playbook to help you configure your network topology. The playbook could pull in BGP data in JSON format before it starts creating the topology:

```
- hosts: localhost leaf spine
  gather_facts: False
  tasks:
    - name: Gather BGP Adjacency info in JSON format
      local_action: command netq show bgp json
      register: result
      #delegate_to: localhost
      run_once: true
```

Based on the outcome, the playbook can then respond appropriately. Later, it can check IP addresses to verify the connections:



```
#ipv6 address check
- name: run ipv6check on broken_dict
  command: netq show ipv6 addresses {{item.key}} {{item.value}}
json
  with_dict: "{{broken_dict}}"
  register: command_outputs
  delegate_to: localhost
  run_once: true
```

Using NetQ Virtual

The NetQ Virtual environment provides another way for you to verify your network configuration before deploying it into production. For more information, see [Installing and Configuring the NetQ Virtual Environment \(see page 83\)](#).

Proactively Monitoring the Network Fabric

NetQ continually and algorithmically checks for various network events (see below) and sends real-time alerts via *NetQ Notifier* to notify users that a network event occurs. When alerted, you can determine precisely where the fault occurred so you can remediate quickly.

You can create filters for how to handle notifications and you can also ignore notifications.

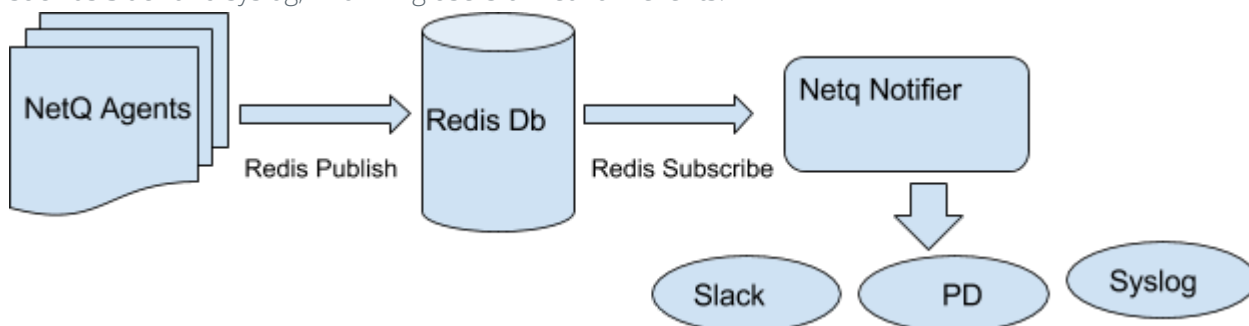
Contents

This chapter covers ...

- [NetQ Notifier \(see page 41\)](#)
 - [Log Message Format \(see page 42\)](#)
 - [Supported Integrations \(see page 42\)](#)
 - [Configuring an Integration \(see page 44\)](#)
 - [Filtering Notifications \(see page 45\)](#)
 - [Example netq.yml File \(see page 47\)](#)
 - [Exporting to ELK \(see page 53\)](#)
 - [Exporting to Splunk \(see page 54\)](#)
 - [Precisely Locating an Issue on the Network \(see page 54\)](#)
 - [Detecting Out of Sync Nodes \(see page 55\)](#)
- [Extending NetQ with Custom Services Using curl \(see page 56\)](#)
- [Exporting NetQ Data \(see page 57\)](#)

NetQ Notifier

The NetQ Notifier's role within the NetQ suite of applications is to deliver alerts to users through mediums such as Slack and syslog, informing users of network events.



Notifications can be generated for the following network events:

- Agent node state

- Backend connections
- Fan
- License
- NTP
- OS
- Port
- PSU
- Services
- Temperature

When a notification arrives, what should you do next? Typically, you could run `netq check` commands; see [Performing Network Diagnostics \(see page 67\)](#) for more information. For a thorough example, read about troubleshooting [MLAG node failures \(see page 58\)](#).

Log Message Format

Messages have the following structure:

```
<timestamp> <node> <service>[PID]: <level> <type>: <message>
```

For example:

```
2017-08-28T22:43:32.794669+00:00 spine01 netq-notifier[13232]: INFO:  
filter#default: BGP: leaf01 peerlink-1.4094: session state changed  
from failed to established
```

Supported Integrations

NetQ supports the ability to send notifications to the following applications:

- **PagerDuty:** NetQ Notifier sends notifications to PagerDuty as PagerDuty events.

<input type="checkbox"/>	Status	Urgency	Title	Created	Service	Assigned To
<input type="checkbox"/>	Resolved	Low	filter#default: NetQ Agent: spine-1: state changed from fresh to rotten SHOW DETAILS (1 resolved alert) #10659	on Aug 31, 2017 at 3:08 PM	Puneet - Netq Notifier integration	--
<input type="checkbox"/>	Resolved	Low	filter#default: Service: noc-se clagd (vrf default): state changed from ok to warning SHOW DETAILS (1 resolved alert) #10658	on Aug 31, 2017 at 3:08 PM	Puneet - Netq Notifier integration	--
<input type="checkbox"/>	Resolved	Low	filter#default: BGP: tor-2 uplink-1: session state changed from established to failed SHOW DETAILS (1 resolved alert) #10657	on Aug 31, 2017 at 3:08 PM	Puneet - Netq Notifier integration	--
<input type="checkbox"/>	Resolved	Low	filter#default: BGP: torc-12 uplink-1: session state changed from established to failed SHOW DETAILS (1 resolved alert) #10656	on Aug 31, 2017 at 3:08 PM	Puneet - Netq Notifier integration	--




If NetQ generates multiple notifications, on the order of 50/second (which could happen when a node reboots or when one peer in an MLAG pair disconnects), PagerDuty does not see all these notifications. You may see warnings in the `netq-notifier.log` file like this:


```
2017-09-20T20:39:48.222458+00:00 rdsq1 netq-notifier[1]:
WARNING: Notifier: notifier-pagerduty: Request failed
with exception: Code: 429, msg: {"status":"throttle
exceeded","message":"Requests for this service are
arriving too quickly. Please retry later."}
```


This is a known limitation in PagerDuty at this time.


- **Slack:** NetQ Notifier sends notifications to Slack as incoming webhooks for a Slack channel you configure. For example:


1:42 ☆


Notifier: Backend connection up
From netq-notifier running on vx-1 | Today at 1:36 PM


NetQ Agent: Agent on these nodes are rotten: vx-1, centos
From netq-notifier running on vx-1 | Today at 1:36 PM


NetQ Agent: Agent on these nodes are rotten: vx-1, centos
From netq-notifier running on vx-1 | Today at 1:36 PM


netq notifier APP 2:20 PM


filter#default: NetQ Agent: ubuntu: state changed from fresh to rotten
From netq-notifier running on vx-1 | Today at 2:20 PM

- **rsyslog:** Using `rsyslog`, NetQ Notifier sends alerts and events to the `/var/log/netq-notifier.log` file by default, but notifications can also be sent to ELK/Logstash or Splunk.

```
2017-08-30T20:15:23.610955+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: Notifier: Queue size is now at 50
2017-08-30T21:56:11.861046+00:00 cel-smallxp-13 netq-notifier[18258]: WARNING: filter#default: Service: cel-bs01-fc4 bgpd (vrf default): state changed from ok to error
2017-08-30T23:17:03.644854+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: Notifier: Queue size is now at 50
2017-08-31T00:47:35.389123+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: Notifier: Queue size is now at 50
2017-08-31T02:17:36.946193+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: Notifier: Queue size is now at 50
2017-08-31T03:58:42.836327+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc102 smond (vrf default): service restarted
2017-08-31T05:15:11.292104+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Link: qct-ly8-04 swp2: state change from down to up
2017-08-31T06:59:48.443736+00:00 cel-smallxp-13 netq-notifier[18258]: WARNING: filter#default: Sensor: qct-ly8-04 fan6: state changed from ok to low
2017-08-31T08:48:08.867436+00:00 cel-smallxp-13 netq-notifier[18258]: WARNING: filter#default: Link: act-5712-12 swp36: state change from up to down
2017-08-31T10:05:36.842427+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc401 smond (vrf default): service restarted
2017-08-31T11:15:09.189561+00:00 cel-smallxp-13 netq-notifier[18258]: WARNING: filter#default: Link: dell-s6000-22 swp1s0: state change from up to down
2017-08-31T13:44:31.388453+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc401 smond (vrf default): service restarted
2017-08-31T14:46:57.604810+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc202 smond (vrf default): service restarted
2017-08-31T16:20:26.888155+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc301 smond (vrf default): service restarted
2017-08-31T17:22:56.880300+00:00 cel-smallxp-13 netq-notifier[18258]: INFO: filter#default: Service: cel-bs01-lc401 smond (vrf default): service restarted
2017-08-31T18:47:57.845644+00:00 cel-smallxp-13 netq-notifier[8037]: INFO: filter#default: Service: cel-bs01-lc101 netqd (vrf default): service restarted
2017-08-31T20:26:03.030005+00:00 cel-smallxp-13 netq-notifier[8037]: WARNING: filter#default: NetQ Agent: cel-bs01-fc2: state changed from fresh to rotten
2017-08-31T21:48:34.233133+00:00 cel-smallxp-13 netq-notifier[8037]: WARNING: filter#default: Sensor: qct-ly8-04 fan6: state changed from ok to low
2017-08-31T23:58:39.619907+00:00 cel-smallxp-13 netq-notifier[8037]: WARNING: filter#default: Sensor: qct-ly8-04 fan6: state changed from ok to low
```

- **Splunk:** NetQ integrates with Splunk using `rsyslog`, a standard mechanism to capture log files in Linux. Splunk provides plugins to handle `rsyslog` inputs.

i	Time	Event
>	9/2/17 4:35:01.000 AM	<14>Sep 2 04:35:01 vx-1 netq-notifier[8019]: INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted host = vx-1 ; source = tcp:51415 ; sourcetype = syslog
>	9/2/17 4:30:00.000 AM	<14>Sep 2 04:30:00 vx-1 netq-notifier[8019]: INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted host = vx-1 ; source = tcp:51415 ; sourcetype = syslog
>	9/2/17 4:29:45.000 AM	<14>Sep 2 04:29:45 vx-1 netq-notifier[8019]: INFO: filter#default: Service: ubuntu rsyslog (vrf default): service restarted host = vx-1 ; source = tcp:51415 ; sourcetype = syslog
>	9/2/17 4:28:45.000 AM	<14>Sep 2 04:28:45 vx-1 netq-notifier[8019]: INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted host = vx-1 ; source = tcp:51415 ; sourcetype = syslog
>	9/2/17 4:28:30.000 AM	<14>Sep 2 04:28:30 vx-1 netq-notifier[8019]: INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted host = vx-1 ; source = tcp:51415 ; sourcetype = syslog

- **ELK/Logstash:** NetQ integrates with ELK/Logstash using `rsyslog`. ELK also provides plugins to handle `rsyslog` inputs.

```
{
  "severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:28:45.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:28:45", "facility_label": "user-level"
}
{"severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu rsyslog (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:29:45.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:29:45", "facility_label": "user-level"
}
{"severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:30:00.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:30:00", "facility_label": "user-level"
}
{"severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu netq-notifier (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:35:01.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:35:01", "facility_label": "user-level"
}
{"severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu netqd (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:38:18.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:38:18", "facility_label": "user-level"
}
{"severity": 6, "pid": "8019", "program": "netq-notifier", "message": "INFO: filter#default: Service: ubuntu netq-agent (vrf default): service restarted\n", "type": "syslog", "priority": 14, "logsource": "vx-1", "@timestamp": "2017-09-02T04:38:18.000Z", "@version": "1", "host": "192.168.50.110", "facility": "1", "severity_label": "Informational", "timestamp": "Sep 2 04:38:18", "facility_label": "user-level"
}
```

Configuring an Integration

You need to define to which applications NetQ sends notifications. By default, NetQ sends notifications only to `syslog`.

To configure PagerDuty or Slack, you need to edit the `/etc/cts/netq/netq.yml` configuration file.

```
cumulus@switch:~$ sudo nano /etc/cts/netq/netq.yml

...

## a) Filter notifications to integrations (Slack or PD) based on
Severity,
## i.e., WARNING to PD, INFO to Slack
# notifier-integrations:
# - name: notifier-slack-channel-1
#   type: slack
#   webhook: "https://<slack-webhook1"
#   severity: INFO <==== Set the severity type here
#   tag: "@slack-tag1"
# - name: notifier-pagerduty
#   type: pagerduty
#   severity: WARNING <==== Set the severity type here
#   api_access_key: "<API Key>"
#   api_integration_key: "<API Integration Key>"
#

...
```

You need to do some extra steps to be able to export NetQ data to [ELK \(see page 53\)](#) or [Splunk \(see page 54\)](#) (see below).

After you modify the NetQ configuration, you must restart the `netq-notifier` service on the telemetry server:

```
cumulus@switch:~$ sudo systemctl restart netq-notifier.service
```

Filtering Notifications

By default, NetQ sends all notifications in response to network events. You can filter this according to your needs.

A filter has three components, a *rule*, an *action* and *output*:

- **Rule:** A set of conditions against which an incoming event is matched. If an incoming event matches the rule, the event information is passed to the action. The rule is a dictionary of key-value pairs, where the "key" is an item associated with the event and "value" is the value of that item. For example:

```
rule:
hostname: leaf01
ifname: swp1
```

If the default rule is not specified or if it is empty, a match always results. You can make NetQ Notifier continue matching filters even if a match is found, by adding `terminate_on_match: False` to the filter. Values specified in the rule are matched with those received in a event using [Python regular expressions](#). NetQ also matches for message severity and sends a notification only if the event is above the given severity. Message severity levels are: INFO, WARNING, ERROR and CRITICAL in ascending order.

- **Action:** The action to perform if the rule matches. The action takes the event provided by the rule stage and generates a message dictionary with a message and its severity. Multiple actions can be prescribed in the action list. An action is typically a Python function that is provided with NetQ or you can write a custom one yourself. If no action is provided, NetQ defaults to a generic handler that looks at the event, and based on that event runs the relevant notification function.
- **Output:** The integrations that will receive the notification. The output contains the message and severity. If the output is *None* the notification is not sent to any integration, including `syslog`. If the output is empty, the message is sent only to `syslog`. Otherwise, the notification is sent to the list of integrations specified in the output list as well as to `syslog`. If ALL is specified, the notification is sent to all integrations.

For example, to send BGP session state notifications to particular Slack channel, in this case, *slack-channel-BGP*, do the following:

```
cumulus@switch:~$ sudo nano /etc/cts/netq/netq.yml

...

## e) Send BGP Session state notifications to particular slack channel
## (slack-channel-BGP), rest to another one (slack-channel-catchall)
# notifier-filters:
# - name: BGP slack channel
# rule:
# type: BgpSession
# output:
# - slack-channel-BGP

...
```

To drop notifications, set the output to *None* for the given rule in the `/etc/cts/netq/netq.yml` file. For example, you can drop all notifications from leaf01 by configuring the following:

```
cumulus@switch:~$ sudo nano /etc/cts/netq/netq.yml

...

## b) Drop all notifications coming from a switch/host say, leaf01
# notifier-filters:
# - name: leaf01 drop
#   rule:
#     hostname: leaf01
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL

...
```

Example netq.yml File

/etc/netq/netq.yml file contents

```
cumulus@switch:~$ cat /etc/cts/netq/netq.yml
## Netq configuration File.
## Configuration is also read from files in /etc/netq/config.d/ and
have
## precedence over config in /etc/cts/netq/netq.yml.
## ----- Common configurations -----
## Backend Configuration for all netq agents and apps on this host.
##
backend:
  server: 10.0.0.165
#  port: 6379
## ----- netq-agent configurations -----
## Netq Agent Configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
#netq-agent:
#  log_level: info
## Docker Agent Configuration
##
## docker_enable: Enable Docker monitoring. Default is True.
## docker_poll_period: Docker poll period in secs. Default is 15 secs.
##
#docker:
#  enable: true
```

```
# poll_period: 15
## ----- netq configurations -----
## Netq configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
#netqd:
# log_level: info
## ----- netq-notifier configurations -----
## Netq Notifier Configuration
##
## log_level: Could be debug, info, warning or error. Default is info.
##
# netq-notifier:
# log_level: debug
## NetQ Notifier Filter Configuration
##
## NetQ Notifier sends notifications to integrations(syslog, slack or
pagerduty)
## based on the events that are happening across the network.
## Notifications are generated based on the filters that have been
specified in
## "notifier-filters". NetQ Agents generate an event when something
interesting
## happens on the host (switch or server) its running on. The
Notifier is always
## listening for these events and once it receives an event, it makes
it go
## through a set of filters.
##
## A filter has 3 stages:
## a) Rule: Defines a set of conditions against which an incoming
event is
## matched. Input to this stage is an incoming event and the event is
sent to
## the next stage if there is a match. If there is a match, the event
## information is passed to the action stage. The rule is a
dictionary of
## key-value pairs, where the "key" is an item associated with the
event and
## "value" is the value of that item,
## e.g. type: Link
##     hostname: leaf-01
##     ifname: swp1
## The Default rule, if none is specified or if it is empty, is to
always assume
## a match.
## Notifier-filter rules are matched sequentially and we stop only
when a match
## is found. You can make the notifier continue matching filters even
if a match
```



```
## is found, by adding "terminate_on_match: False" to the filter.
Values
## specified in the rule are matched with those received in a event
using python
## regular expressions https://docs.python.org/2/library/re.html
## We can also match for message severity and print messages only if
it is above
## the given severity. Message severity levels are: INFO, WARNING,
ERROR and
## CRITICAL in ascending order.
##
## b) Action: action to perform if the "rule" is matched. The action
stage
## take the event provided by the "rule" stage and generates a message
## dictionary with a message and its severity. Multiple actions can be
## prescribed in the "action" list. "action" is typically a python
function that
## is provided with the tool or a custom one written by the user. If
no action
## is provided, we default to a generic handler which looks at the
event and
## based on the event runs the relevant notification function.
##
## c) output: This stage takes the message dictionary provided by the
action
## stage and sends the message and severity to the right integration
to display
## the message. If the output is None the message is not sent to any
integration
## or syslog. If output is empty, the message is sent only to syslog.
Else the
## message is sent to the list of integrations specified in the
output list and
## syslog. If ALL is specified, the message is sent to all
integrations.
## Integrations are defined in notifier-integrations.
##
## The config file comes with the following default filter:
##
## notifier-filters:
## - name: default
##   rule:
##     output:
##       - ALL
##
## which is an empty rule, empty action and output to all. This
defaults to
## match all rules and then perform the default action which is to
run the
## generic handler mentioned in the action stage above.
##
## NetQ Integration Configuration
```

```
##
## The integrations refer to the external tool where you would like
## to receive
## the notification. An integration is added as a list element to
## "notifier-integrations". Each integration must have a "name" and
## "type".
## Severity is optional and lets you send messages above that level
## to the
## integration. Allowed values are: INFO, WARNING, ERROR, CRITICAL in
## increasing
## order. Currently allowed "type" are "slack" and "pagerduty". You
## can define
## multiple slack or PD integrations.
##
## For Slack integration, along with a name and "type: slack", you
## also need to
## also provide the Incoming Webhook of the channel. The webhook URL
## for your
## channel can be found or created in Slack at:
## Apps -> Custom Integrations -> Incoming Webhooks.
## Tags are optional and are strings that are attached to the end of
## the
## notification message.
## E.g.
# notifier-integrations:
# - name: notifier-slack-channel-1
#   type: slack
#   webhook: "https://<slack-webhook1>"
#   severity: INFO,
#   tag: "@slack-tag1"
##
## For pagerDuty, along with name and "type: pagerduty", you also
## need to
## provide the "api_access_key" and "api_integration_key" from
## Pagerduty.
## A unique API Access Key which can be created on your PagerDuty
## website at:
## Configuration -> API Access -> Create New API Key
## An 'Integration Key' can be created/found on your PagerDuty
## website at:
## Configuration -> Services -> Add New Service -> New Integration ->
## Select Integration Type as 'Use our API directly: Events API v2'.
## E.g. pagerduty integration along with slack
# notifier-integrations:
# - name: notifier-slack-channel-1
#   type: slack
#   webhook: "https://<slack-webhook1>"
#   severity: INFO
#   tag: "@slack-tag1"
# - name: notifier-pagerduty
#   type: pagerduty
#   severity: WARNING
```

```
#   api_access_key: <API Key>
#   api_integration_key: <API Integration Key>
##
## Customizing Notifications
## Here are some examples on how to customize your notifications:
##
## a) Filter notifications to integrations (Slack or PD) based on
Severity,
## i.e., WARNING to PD, INFO to Slack
# notifier-integrations:
# - name: notifier-slack-channel-1
#   type: slack
#   webhook: "https://<slack-webhook1"
#   severity: INFO <==== Set the severity type here
#   tag: "@slack-tag1"
# - name: notifier-pagerduty
#   type: pagerduty
#   severity: WARNING <==== Set the severity type here
#   api_access_key: "<API Key>"
#   api_integration_key: "<API Integration Key>"
#
##
## b) Drop all notifications coming from a switch/host say, leaf-01
# notifier-filters:
# - name: leaf-01 drop
#   rule:
#     hostname: leaf-01
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
##
## c) Drop all notifications coming from switches whose name starts
with leaf
# notifier-filters:
# - name: leaf drop
#   rule:
#     hostname: "leaf-.*"
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
##
## d) Drop all notifications coming from a particular link, e.g. leaf-
01 swp1
# notifier-filters:
# - name: leaf-01 swp1 drop
#   rule:
```

```
#     type: Link
#     hostname: leaf-01
#     ifname: swp1
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
##
## e) Send BGP Session state notifications to particular slack channel
## (slack-channel-BGP), rest to another one (slack-channel-catchall)
# notifier-filters:
# - name: BGP slack channel
#   rule:
#     type: BgpSession
#   output:
#     - slack-channel-BGP
# - name: default
#   rule:
#   output:
#     - slack-channel-catchall
##
## f) Send BgpSession notifications based on severity to different
slack channels
# notifier-filters:
# - name: BGP severity slack channel
#   rule:
#     type: BgpSession
#     severity: WARNING
#   output:
#     - slack-channel-BGP-info
# - name: default
#   rule:
#   output:
#     - slack-channel-catchall
##
## g) Drop all temperature related alerts
# notifier-filters:
# - name: temp drop
#   rule:
#     type: Temp
#   output:
#     - None
# - name: default
#   rule:
#   output:
#     - ALL
notifier-filters:
  - name: default
    rule:
    output:
```

- ALL

Exporting to ELK

To export NetQ Notifier data to ELK via Logstash, on the host running the NetQ Telemetry Server and NetQ Notifier, configure the notifier to send the logs to a Logstash instance. In the following example, Logstash is on a host with the IP address 192.168.50.30, using port 51414:

```
# rsyslog - logstash configuration
sed -i '/$netq_notifier_log/a if $programname == "netq-notifier" then
@@192.168.50.30:51414' /etc/rsyslog.d\
/50-netq-notifier.conf
```

Then restart `rsyslog`:

```
root@ts_host:~# systemctl restart rsyslog
```

On the server running Logstash, create a file in `/etc/logstash/conf.d/` called `notifier_logstash.conf`, and paste in the following text, using the IP address and port you specified earlier:

```
root@ts_host:~# vi /etc/logstash/conf.d/notifier_logstash.conf

input {
  syslog {
    type => syslog
    port =>
51414
  }
}
output {
  file {
    path => "/tmp/logstash_notifier.
log"
  }
}
```

Then restart Logstash:

```
root@logstash_host:~# systemctl restart logstash
```

NetQ Notifier logs now appear in `/tmp/logstash_notifier.log` on the Logstash host.

Exporting to Splunk

To export NetQ Notifier data to Splunk, on the host running the NetQ Telemetry Server and NetQ Notifier, configure the notifier to send the logs to Splunk. In the following example, Splunk is on a host with the IP address 192.168.50.30, using port 51414:

```
# rsyslog - splunk configuration
sed -i `/$netq_notifier_log/a if $programname == "netq-notifier" then
@@192.168.50.30:51415' /etc/rsyslog.d\
/50-netq-notifier.conf
```

Then restart `rsyslog`:

```
root@ts_host:~# systemctl restart rsyslog
```

To configure Splunk, do the following:

1. In Splunk in a browser, choose **Add Data > monitor > TCP > Port**, and set it to `51415`.
2. Click **Next**, then choose **Source Type (syslog) > Review > Done**.

NetQ Notifier messages now appear in Splunk.

Precisely Locating an Issue on the Network

NetQ helps you locate exactly where you have an issue on your network. Use `netq check` or `netq trace` to locate a fault, then run `netq show changes` to see what could have caused it.

For example, checking the state of the VLANs on your network, you can see where some nodes have mismatched VLANs with their peers:

```
cumulus@leaf01:~$ netq check vlan
Checked Nodes: 25, Checked Links: 775, Failed Nodes: 3, Failed Links: 6 Vlan
and/or PVID mismatch found on following links
Hostname Interface Vlans Peer Peer Interface Peer Vlans Error
-----
-----
server01 torbond1 103-106,1000-1005 leaf02 hostbond2 101-106,1000-1005 VLAN
set Mismatch
server01 torbond1 103-106,1000-1005 leaf01 hostbond2 101-106,1000-1005 VLAN
set Mismatch
server02 torbond1 102-106,1000-1005 leaf02 hostbond3 101-106,1000-1005 VLAN
set Mismatch
server02 torbond1 102-106,1000-1005 leaf01 hostbond3 101-106,1000-1005 VLAN
set Mismatch
server03 torbond1 102-106,1000-1005 leaf04 hostbond2 101-106,1000-1005 VLAN
set Mismatch
server03 torbond1 102-106,1000-1005 leaf03 hostbond2 101-106,1000-1005 VLAN
set Mismatch
```

Detecting Out of Sync Nodes

NetQ 1.1 has added commands to assist in determining if any nodes are out of sync. Use `netq check ntp` to determine if any nodes are out of sync, and `netq show services ntp` and `netq show ntp` to review the records:

```
cumulus@switch:~$ netq check ntp
Total Nodes: 18, Checked Nodes: 18, Rotten Nodes: 7, Unknown Nodes:
0, failed NTP Nodes: 8
Hostname          NTP Sync      Connect Time
-----
act-5712-12       Rotten        2017-09-01 09:15:30
act-6712-06       Rotten        2017-09-01 09:16:02
act-7712-04       Rotten        2017-09-01 09:16:05
cel-smallxp-13    no            2017-08-26 01:15:00
dell-s4000-10     Rotten        2017-09-01 09:14:53
dell-s6000-22     Rotten        2017-09-01 09:15:29
mlx-2410-02       Rotten        2017-09-01 09:16:23
qct-ly8-04        Rotten        2017-09-01 09:14:56
```

```
cumulus@switch:~$ netq show services ntp
Matching services records are:
Node          Service      PID  VRF      Enabled      Active
Monitored     Status      Up Time    Last Changed
-----
leaf01        ntp          913  default  yes          yes
no            ok           2h ago    2h ago
leaf02        ntp          911  default  yes          yes
no            ok           2h ago    2h ago
leaf03        ntp          909  default  yes          yes
no            ok           2h ago    2h ago
leaf04        ntp          910  default  yes          yes
no            ok           2h ago    2h ago
oob-mgmt-server ntp          729  default  yes          yes
no            ok           2h ago    2h ago
spine01       ntp          909  default  yes          yes
no            ok           2h ago    2h ago
spine02       ntp          909  default  yes          yes
no            ok           2h ago    2h ago
```

```
cumulus@switch:~$ netq show ntp
Hostname          NTP Sync      Current Server      Stratum
-----
act-5712-12       -             -                   -
act-6712-06       -             -                   -
```

```
act-7712-04      -      -      -
cel-bs01-fc1     yes     chimera.buffero  2
cel-bs01-fc2     yes     104.156.99.226  2
cel-bs01-fc4     yes     104.156.99.226  2
cel-bs01-lc101   yes     chimera.buffero  2
cel-bs01-lc102   yes     secure.visionne  2
cel-bs01-lc201   yes     chimera.buffero  2
cel-bs01-lc202   yes     secure.visionne  2
cel-bs01-lc301   yes     chimera.buffero  2
cel-bs01-lc401   yes     104.156.99.226  2
cel-bs01-lc402   yes     chimera.buffero  2
cel-smallxp-13   no      -      16
dell-s4000-10    -      -      -
dell-s6000-22    -      -      -
mlx-2410-02      -      -      -
qct-ly8-04       -      -      -
{code}
```



These commands require `systemd` in order to run correctly.

Extending NetQ with Custom Services Using curl

You can extend NetQ to monitor parameters beyond what it monitors by default. For example, you can create a service that runs a series of pings to a known host or between two known hosts to ensure that connectivity is valid. Or you can create a service that curls a URL and sends the output to `/dev/null`. This method works with the [NetQ time machine](#) (see [page 70](#)) capability regarding `netq show services`.

1. As the `sudo` user on a node running the NetQ agent, edit the `/etc/netq/config.d/netq-agent-commands.yml` file.
2. Create the custom service. In the example below, the new service is called `web`. You need to specify:
 - The *period* in seconds.
 - The *key* that identifies the name of the service.
 - The command will *run* always. If you do not specify *always* here, you must enable the service manually using `systemctl`.
 - The *command* to run. In this case we are using `curl` to ping a web server.

```
cumulus@leaf01:~$ sudo vi /etc/netq/config.d/netq-agent-commands.
yml

user-commands:
- service: 'misc'
  commands:
    - period: "60"
      key: "config-interfaces"
      command: "/bin/cat /etc/network/interfaces"
```



```
- period: "60"
  key: "config-ntp"
  command: "/bin/cat /etc/ntp.conf"
- service: "zebra"
  commands:
    - period: "60"
      key: "config-quagga"
      command: ["/usr/bin/vtysh", "-c", "show running-config"]

- service: "web"
  commands:
    - period: "60"
      key: "webping"
      run: "always"
      command: ['/usr/bin/curl https://cumulusnetworks.com/ -o
/dev/null']
```

3. After you save and close the file, restart the NetQ agent:

```
cumulus@leaf01:~$ netq config agent restart
```

4. You can verify the command is running by checking the `/var/run/netq-agent-running.json` file:

```
cumulus@leaf01:~$ cat /var/run/netq-agent-running.json
{"commands": [{"service": "smond", "always": false, "period":
30, "callback": {}, "command": "/usr/sbin/smonctl -j", "key":
"smonctl-json"}, {"service": "misc", "always": false, "period":
30, "callback": {}, "command": "/usr/sbin/switchd -lic", "key":
"cl-license"}, {"service": "misc", "always": false, "period":
30, "callback": {}, "command": null, "key": "ports"},
{"service": "misc", "always": false, "period": 60, "callback":
null, "command": "/bin/cat /etc/network/interfaces", "key":
"config-interfaces"}, {"service": "misc", "always": false,
"period": 60, "callback": null, "command": "/bin/cat /etc/ntp.
conf", "key": "config-ntp"}, {"service": "lldpd", "always":
false, "period": 30, "callback": {}, "command": "/usr/sbin
/lldpctl -f json", "key": "lldp-neighbor-json"}, {"service":
"mstpd", "always": false, "period": 15, "callback": {},
"command": "/sbin/mstpcctl showall json", "key": "mstpcctl-bridge-
json"}], "backend": {"server": "10.0.0.165"}}
```

5. And you can see the service is running on the host when you run `netq show services`:

```
cumulus@leaf01:~$ netq show services web
```

Exporting NetQ Data

Data from the NetQ Telemetry Server can be exported in a number of ways. First, you can use the `json` option to output check and show commands to JSON format for parsing in other applications.

For example, you can check the state of BGP on your network with `netq check bgp`:

```
cumulus@leaf01:~$ netq check bgp
Total Nodes: 25, Failed Nodes: 2, Total Sessions: 228 , Failed
Sessions: 2,
Node          Peer Name  Peer Hostname Reason          Time
-----
exit01        swp6.2     spine01       Rotten Agent    15h ago
spine01       swp3.2     exit01        Idle            15h ago
```

When you show the output in JSON format, this same command looks like this:

```
cumulus@leaf01:~$ netq check bgp json
{
  "failedNodes": [
    {
      "node": "exit-1",
      "reason": "Idle",
      "peerId": "firewall-1",
      "neighbor": "swp6.2",
      "time": "15h ago"
    },
    {
      "node": "firewall-1",
      "reason": "Idle",
      "peerId": "exit-1",
      "neighbor": "swp3.2",
      "time": "15h ago"
    }
  ],
  "summary": {
    "checkedNodeCount": 25,
    "failedSessionCount": 2,
    "failedNodeCount": 2,
    "totalSessionCount": 228
  }
}
```

MLAG Troubleshooting with NetQ

This chapter outlines a few scenarios that illustrate how you use NetQ to troubleshoot MLAG on Cumulus Linux switches. Each starts with a log message that indicates the current of MLAG state.

NetQ can monitor many aspects of an MLAG configuration, including:

- Verifying the current state of all nodes
- Verifying the dual connectivity state
- Checking that the peer link is part of the bridge
- Verifying whether MLAG bonds are not bridge members
- Verifying whether the VXLAN interface is not a bridge member
- Checking for remote-side service failures caused by `systemctl`
- Checking for VLAN-VNI mapping mismatches
- Checking for layer 3 MTU mismatches on peerlink subinterfaces
- Checking for VXLAN active-active address inconsistencies
- Verifying that STP priorities are the same across both peers

Contents

This chapter covers ...

- [All Nodes Are Up \(see page 59\)](#)
- [Dual-connected Bond Is Down \(see page 61\)](#)
- [VXLAN Active-active Device or Interface Is Down \(see page 63\)](#)
- [Remote-side clagd Stopped by systemctl Command \(see page 65\)](#)

All Nodes Are Up

When the MLAG configuration is running smoothly, NetQ Notifier sends out a message that all nodes are up:

```
2017-05-22T23:13:09.683429+00:00 noc-pr netq-notifier[5501]: INFO:
CLAG: All nodes are up
```

Running `netq show clag` confirms this:

```
cumulus@noc-pr:~$ netq show clag
Matching CLAG session records are:
Node          Peer          SysMac          State Backup
#Bonds #Dual Last Changed
-----
mlx-2700-03   torc-11(P)    44:38:39:ff:ff:01 up    up
8           8           26s ago
noc-pr(P)     noc-se        00:01:01:10:00:01 up    up
9           9           39m ago
noc-se        noc-pr(P)     00:01:01:10:00:01 up    up
9           9           40m ago
torc-11(P)    mlx-2700-03   44:38:39:ff:ff:01 up    up
8           8           27s ago
```

```

torc-21(P)      torc-22      44:38:39:ff:ff:02 up    up
8      8      2h ago
torc-22      torc-21(P)      44:38:39:ff:ff:02 up    up
8      8      2h ago

```

You can also verify a specific node is up:

```

cumulus@noc-pr:~$ netq mlx-2700-03 show clag
Matching CLAG session records are:
Node           Peer           SysMac           State Backup
#Bonds #Dual Last Changed
-----
mlx-2700-03    torc-11(P)      44:38:39:ff:ff:01 up    up
8      8      45s ago

```

Similarly, checking the MLAG state with NetQ also confirms this:

```

cumulus@noc-pr:~$ netq check clag
Checked Nodes: 6, Failed Nodes: 0

```

When you're directly on the switch, you can run `clagctl` to get the state:

```

cumulus@mlx-2700-03:/var/log# sudo clagctl

The peer is alive
Peer Priority, ID, and Role: 4096 00:02:00:00:00:4e primary
Our Priority, ID, and Role: 8192 44:38:39:00:a5:38 secondary
Peer Interface and IP: peerlink-3.4094 169.254.0.9
VxLAN Anycast IP: 36.0.0.20
Backup IP: 27.0.0.20 (active)
System MAC: 44:38:39:ff:ff:01

CLAG Interfaces
Our Interface    Peer Interface    CLAG Id Conflicts    Proto-
Down Reason
-----
vx-38            vx-38            -            -            -
vx-33            vx-33            -            -            -
hostbond4        hostbond4        1            -            -
hostbond5        hostbond5        2            -            -
vx-37            vx-37            -            -            -
vx-36            vx-36            -            -            -
vx-35            vx-35            -            -            -
vx-34            vx-34            -            -            -

```

Dual-connected Bond Is Down

When dual connectivity is lost in an MLAG configuration, you'll receive messages from NetQ Notifier similar to the following:

```
2017-05-22T23:14:40.290918+00:00 noc-pr netq-notifier[5501]: WARNING:
LINK: 1 link(s) are down. They are: mlx-2700-03 hostbond5
2017-05-22T23:14:53.081480+00:00 noc-pr netq-notifier[5501]: WARNING:
CLAG: 1 node(s) have failures. They are: mlx-2700-03
2017-05-22T23:14:58.161267+00:00 noc-pr netq-notifier[5501]: WARNING:
CLAG: 2 node(s) have failures. They are: mlx-2700-03, torc-11
```

To begin your investigation, show the status of the `clagd` service:

```
cumulus@noc-pr:~$ netq mlx-2700-03 show services clagd

Matching services records are:
Hostname      Service  PID   VRF      Enabled  Active  Monitored
Status  Up Time   Last Changed
-----
-----
mlx-2700-03 clagd      5802  default  yes      yes     yes
warning  1h ago    2m ago
```

Checking the MLAG status provides the reason for the failure:

```
cumulus@noc-pr:~$ netq check clag
Checked Nodes: 6, Warning Nodes: 2
Node          Reason
-----
-----
-----
mlx-2700-03   Link Down: hostbond5
torc-11       Singly Attached Bonds: hostbond5
```

You can retrieve the output in JSON format for importing the output into another tool:

```
cumulus@noc-pr:~$ netq check clag json
{
  "warningNodes": [
    { "node": "mlx-2700-03", "reason": "Link Down: hostbond5" },
    { "node": "torc-11", "reason": "Singly Attached Bonds: hostbond5" }
  ],
  "failedNodes": [],
  "summary":
```

```
{ "checkedNodeCount": 6, "failedNodeCount": 0, "warningNodeCount": 2 }
}
```

After you fix the issue, you can show the MLAG state to see if all the nodes are up. The notifications from NetQ Notifier indicate all nodes are UP, and the `netq check` flag also indicates there are no failures.

```
cumulus@noc-pr:~$ netq show clag
Matching CLAG session records are:
Node           Peer           SysMac           State Backup
#Bonds #Dual Last Changed
-----
mlx-2700-03     torc-11(P)      44:38:39:ff:ff:01 up    up
8             7             52s ago
noc-pr(P)       noc-se          00:01:01:10:00:01 up    up
9             9             27m ago
noc-se          noc-pr(P)       00:01:01:10:00:01 up    up
9             9             27m ago
torc-11(P)      mlx-2700-03     44:38:39:ff:ff:01 up    up
8             7             50s ago
torc-21(P)      torc-22         44:38:39:ff:ff:02 up    up
8             8             1h ago
torc-22         torc-21(P)      44:38:39:ff:ff:02 up    up
8             8             1h ago
```

When you're directly on the switch, you can run `clagctl` to get the state:

```
cumulus@mlx-2700-03:/var/log# sudo clagctl

The peer is alive
Peer Priority, ID, and Role: 4096 00:02:00:00:00:4e primary
Our Priority, ID, and Role: 8192 44:38:39:00:a5:38 secondary
Peer Interface and IP: peerlink-3.4094 169.254.0.9
VxLAN Anycast IP: 36.0.0.20
Backup IP: 27.0.0.20 (active)
System MAC: 44:38:39:ff:ff:01

CLAG Interfaces
Our Interface   Peer Interface   CLAG Id Conflicts   Proto-
Down Reason
-----
vx-38          vx-38            -                    -
vx-33          vx-33            -                    -
hostbond4      hostbond4        1                    -
hostbond5      -                2                    -
vx-37          vx-37            -                    -
vx-36          vx-36            -                    -
vx-35          vx-35            -                    -
```

vx-34	vx-34	-	-	-
-------	-------	---	---	---

VXLAN Active-active Device or Interface Is Down

When a VXLAN active-active device or interface in an MLAG configuration is down, log messages also include VXLAN and LNV checks.

```
2017-05-22T23:16:51.517522+00:00 noc-pr netq-notifier[5501]: WARNING:
VXLAN: 2 node(s) have failures. They are: mlx-2700-03, torc-11
2017-05-22T23:16:51.525403+00:00 noc-pr netq-notifier[5501]: WARNING:
LINK: 2 link(s) are down. They are: torc-11 vx-37, mlx-2700-03 vx-37
2017-05-22T23:16:54.194681+00:00 noc-pr netq-notifier[5501]: WARNING:
LNV: 1 node(s) have failures. They are: torc-22
2017-05-22T23:16:59.448755+00:00 noc-pr netq-notifier[5501]: WARNING:
LNV: 3 node(s) have failures. They are: tor-2, torc-21, torc-22
2017-05-22T23:17:04.703044+00:00 noc-pr netq-notifier[5501]: WARNING:
CLAG: 2 node(s) have failures. They are: mlx-2700-03, torc-11
```

To begin your investigation, show the status of the `clagd` service:

```
cumulus@noc-pr:~$ netq mlx-2700-03 show service clagd
Matching services records are:
Node      Service  PID  VRF      Enabled  Active  Monitored
Status    Up Time  Last Changed
-----
mlx-2700-03 clagd    5802  default  yes      yes     yes
error     2h ago  3m ago
```

Checking the MLAG status provides the reason for the failure:

```
cumulus@noc-pr:~$ netq check clag
Checked Nodes: 6, Warning Nodes: 2, Failed Nodes: 2
Node      Reason
-----
mlx-2700-03 Protodown Bonds: vx-37:vxlan-single
torc-11    Protodown Bonds: vx-37:vxlan-single
```

You can retrieve the output in JSON format for importing the output into another tool:

```
cumulus@noc-pr:~$ netq check clag json
{
  "failedNodes": [
```

```
{ "node": "mlx-2700-03", "reason": "Protodown Bonds: vx-37:vxlan-
single" }
,
{ "node": "torc-11", "reason": "Protodown Bonds: vx-37:vxlan-single" }
],
"summary":
{ "checkedNodeCount": 6, "failedNodeCount": 2, "warningNodeCount": 2 }
}
```

After you fix the issue, you can show the MLAG state to see if all the nodes are up:

```
cumulus@noc-pr:~$ netq show clag
Matching CLAG session records are:
Node          Peer          SysMac          State Backup
#Bonds #Dual Last Changed
-----
mlx-2700-03    torc-11(P)      44:38:39:ff:ff:01 up    up
8            7            52s ago
noc-pr(P)      noc-se          00:01:01:10:00:01 up    up
9            9            27m ago
noc-se         noc-pr(P)       00:01:01:10:00:01 up    up
9            9            27m ago
torc-11(P)     mlx-2700-03     44:38:39:ff:ff:01 up    up
8            7            50s ago
torc-21(P)     torc-22         44:38:39:ff:ff:02 up    up
8            8            1h ago
torc-22        torc-21(P)      44:38:39:ff:ff:02 up    up
8            8            1h ago
```

When you're directly on the switch, you can run `clagctl` to get the state:

```
cumulus@mlx-2700-03:/var/log# sudo clagctl

The peer is alive
Peer Priority, ID, and Role: 4096 00:02:00:00:00:4e primary
Our Priority, ID, and Role: 8192 44:38:39:00:a5:38 secondary
Peer Interface and IP: peerlink-3.4094 169.254.0.9
VxLAN Anycast IP: 36.0.0.20
Backup IP: 27.0.0.20 (active)
System MAC: 44:38:39:ff:ff:01

CLAG Interfaces
Our Interface    Peer Interface    CLAG Id Conflicts    Proto-
Down Reason
-----
vx-38            vx-38            -                    -
vx-33            vx-33            -                    -
```


hostbond4	hostbond4	1	-	-
hostbond5	hostbond5	2	-	-
vx-37	-	-	-	vxlan-
single				
vx-36	vx-36	-	-	-
vx-35	vx-35	-	-	-
vx-34	vx-34	-	-	-

Remote-side clagd Stopped by systemctl Command

In the event the `clagd` service is stopped via the `systemctl` command, NetQ Notifier sends messages similar to the following:

```
2017-05-22T23:51:19.539033+00:00 noc-pr netq-notifier[5501]: WARNING:
VXLAN: 1 node(s) have failures. They are: torc-11
2017-05-22T23:51:19.622379+00:00 noc-pr netq-notifier[5501]: WARNING:
LINK: 2 link(s) flapped and are down. They are: torc-11 hostbond5,
torc-11 hostbond4
2017-05-22T23:51:19.622922+00:00 noc-pr netq-notifier[5501]: WARNING:
LINK: 23 link(s) are down. They are: torc-11 VlanA-1-104-v0, torc-11
VlanA-1-101-v0, torc-11 VlanA-1, torc-11 vx-33, torc-11 vx-36, torc-
11 vx-37, torc-11 vx-34, torc-11 vx-35, torc-11 swp7, torc-11 VlanA-1-
102-v0, torc-11 VlanA-1-103-v0, torc-11 VlanA-1-100-v0, torc-11 VlanA-
1-106-v0, torc-11 swp8, torc-11 VlanA-1.106, torc-11 VlanA-1.105,
torc-11 VlanA-1.104, torc-11 VlanA-1.103, torc-11 VlanA-1.102, torc-
11 VlanA-1.101, torc-11 VlanA-1.100, torc-11 VlanA-1-105-v0, torc-11
vx-38
2017-05-22T23:51:27.696572+00:00 noc-pr netq-notifier[5501]: INFO:
LINK: 15 link(s) are up. They are: torc-11 VlanA-1.106, torc-11 VlanA-
1-104-v0, torc-11 VlanA-1.104, torc-11 VlanA-1.103, torc-11 VlanA-
1.101, torc-11 VlanA-1-100-v0, torc-11 VlanA-1.100, torc-11 VlanA-
1.102, torc-11 VlanA-1-101-v0, torc-11 VlanA-1-102-v0, torc-11 VlanA-
1.105, torc-11 VlanA-1-103-v0, torc-11 VlanA-1-106-v0, torc-11 VlanA-
1, torc-11 VlanA-1-105-v0
2017-05-22T23:51:30.863789+00:00 noc-pr netq-notifier[5501]: WARNING:
LNV: 1 node(s) have failures. They are: torc-11
2017-05-22T23:51:36.156708+00:00 noc-pr netq-notifier[5501]: WARNING:
CLAG: 2 node(s) have failures. They are: mlx-2700-03, torc-11
2017-05-22T23:51:36.183638+00:00 noc-pr netq-notifier[5501]: WARNING:
LNV: 2 node(s) have failures. They are: spine-2, torc-11
2017-05-22T23:51:41.444670+00:00 noc-pr netq-notifier[5501]: WARNING:
LNV: 1 node(s) have failures. They are: torc-11
```

Showing the MLAG state reveals which nodes are down:

```
cumulus@noc-pr:~$ netq show clag
Matching CLAG session records are:
```

Node	Peer	SysMac	State	Backup
#Bonds	#Dual	Last Changed		
-----	-----	-----	-----	-----
mlx-2700-03		44:38:39:ff:ff:01	down	down
8	0	33s ago		
noc-pr(P)	noc-se	00:01:01:10:00:01	up	up
9	9	1h ago		
noc-se	noc-pr(P)	00:01:01:10:00:01	up	up
9	9	1h ago		
torc-11		44:38:39:ff:ff:01	down	n/a
0	0	32s ago		
torc-21(P)	torc-22	44:38:39:ff:ff:02	up	up
8	8	2h ago		
torc-22	torc-21(P)	44:38:39:ff:ff:02	up	up
8	8	2h ago		

Checking the MLAG status provides the reason for the failure:

```
cumulus@noc-pr:~$ netq check clag
Checked Nodes: 6, Warning Nodes: 1, Failed Nodes: 2
Node          Reason
-----
-----
mlx-2700-03    Peer Connectivity failed
torc-11        Peer Connectivity failed
```

You can retrieve the output in JSON format for importing the output into another tool:

```
cumulus@noc-pr:~$ netq check clag json
{
  "failedNodes": [
    { "node": "mlx-2700-03", "reason": "Peer Connectivity failed" },
    { "node": "torc-11", "reason": "Peer Connectivity failed" },
  ],
  "summary": {
    "checkedNodeCount": 6, "failedNodeCount": 2, "warningNodeCount": 1 }
}
```

When you're directly on the switch, you can run `clagctl` to get the state:

```
root@mlx-2700-03:/var/log# clagctl

The peer is not alive
Our Priority, ID, and Role: 8192 44:38:39:00:a5:38 primary
```



```
Peer Interface and IP: peerlink-3.4094 169.254.0.9
VxLAN Anycast IP: 36.0.0.20
Backup IP: 27.0.0.20 (inactive)
System MAC: 44:38:39:ff:ff:01
```

CLAG Interfaces

Our Interface	Peer Interface	CLAG Id	Conflicts	Proto- Down Reason
vx-38	-	-	-	-
vx-33	-	-	-	-
hostbond4	-	1	-	-
hostbond5	-	2	-	-
vx-37	-	-	-	-
vx-36	-	-	-	-
vx-35	-	-	-	-
vx-34	-	-	-	-

Performing Network Diagnostics

NetQ provides users with the ability to go back in time to replay the network state, see fabric-wide event changelogs and root cause state deviations. The NetQ Telemetry Server maintains data collected by NetQ agents in a time-series database, making fabric-wide events available for analysis. This enables you to replay and analyze network-wide events for better visibility and to correlate patterns. This allows for root-cause analysis and optimization of network configs for the future.

Contents

This chapter covers ...

- [Diagnosing an Event after It Occurs \(see page 68\)](#)
- [Using NetQ as a Time Machine \(see page 70\)](#)
 - [How Far Back in Time Can You Travel? \(see page 71\)](#)
- [Using trace in a VRF \(see page 72\)](#)

Diagnosing an Event after It Occurs

NetQ provides a number of commands to enable you to diagnose past events.

NetQ Notifier records network events and sends them to `syslog`, or another third-party service like PagerDuty or Slack. You can use `netq show changes` to look for any changes made to the runtime configuration that may have triggered the alert, then use `netq trace` to track the connection between the nodes.

The `netq trace` command traces the route of an IP or MAC address from one endpoint to another. It works across bridged, routed and VXLAN connections, computing the path using available data instead of sending real traffic — this way, it can be run from anywhere. It performs MTU and VLAN consistency checks for every link along the path.

For example, say you get an alert about a BGP session failure. You can quickly run `netq check bgp` to determine what sessions failed:

```
cumulus@leaf01:~$ netq check bgp
Total Nodes: 25, Failed Nodes: 4, Total Sessions: 228 , Failed
Sessions: 6,
Node      Neighbor  Peer ID  Reason  Time
-----
exit01    swp7.2    spine02  Idle    53m ago
exit01    swp7.3    spine02  Idle    53m ago
exit02    swp6.4    spine01  Idle    53m ago
spine01    swp4.4    exit02   Idle    53m ago
spine02    swp3.2    exit01   Idle    53m ago
spine02    swp3.3    exit01   Idle    53m ago
```

You can run a trace from spine01 to leaf02, which has the IP address 10.1.20.252:

```
cumulus@leaf01:~$ netq trace 10.1.20.252 from spine01 around 5m
spine01 -- spine01:swp1 -- leaf01:vlan20
        -- spine01:swp2 -- leaf02:vlan20
```

Then you can check what's changed on the network to help you identify the problem. Notice the nodes in a *Failed* state filter to the top of the list:

```
cumulus@leaf01:~$ netq show bgp changes
Matching BGP Session records are:
Node          Neighbor          VRF
ASN           Peer ASN   State  PfxRx      DbState  Last Changed
-----
leaf04        swp52(spine02)    default
64516         65000        Estd    6          Add      5h ago
leaf03        swp52(spine02)    default
64515         65000        Estd    5          Add      5h ago
leaf01        swp52(spine02)    default
64513         65000        Estd    5          Add      5h ago
leaf02        swp52(spine02)    default
64514         65000        Estd    6          Add      5h ago
spine02       swp2(leaf02)      default
65000         64514        Estd    2          Add      5h ago
spine02       swp3(leaf03)      default
65000         64515        Estd    2          Add      5h ago
spine02       swp1(leaf01)      default
65000         64513        Estd    2          Add      5h ago
spine02       swp4(leaf04)      default
65000         64516        Estd    2          Add      5h ago
leaf04        swp51(spine01)    default
64516         65000        Estd    6          Add      5h ago
spine01       swp2(leaf02)      default
65000         64514        Estd    2          Add      5h ago
leaf02        swp51(spine01)    default
64514         65000        Estd    6          Add      5h ago
leaf01        swp51(spine01)    default
64513         65000        Estd    5          Add      5h ago
spine01       swp1(leaf01)      default
65000         64513        Estd    2          Add      5h ago
spine01       swp4(leaf04)      default
65000         64516        Estd    2          Add      5h ago
leaf03        swp51(spine01)    default
64515         65000        Estd    5          Add      5h ago
spine01       swp3(leaf03)      default
65000         64515        Estd    2          Add      5h ago
```

Using NetQ as a Time Machine

With NetQ, you can travel back to a specific point in time or a range of times to help you isolate errors and issues.

For example, if you think you had an issue with your sensors last night, you can check the sensors on all your nodes around the time you think the issue occurred:

```
cumulus@leaf01:~$ netq check sensors around 12h
Total Nodes: 25, Failed Nodes: 0, Checked Sensors: 221, Failed Sensors:
0
```

Or you can specify a range of times using the `between` option. The units of time you can specify are second (`s`), minutes (`m`), hours (`h`) and days (`d`). Always specify the most recent time first, then the more distant time. For example, to see the changes made to the network between the past minute and 5 minutes ago, you'd run:

```
cumulus@leaf01:~$ netq show changes between 1m and 5m
No changes to specified interfaces found
No changes to interface addresses found
Matching MAC table records are:
```

Origin Port	MAC	DbState	Last Changed	VLAN	Node Name	Egress
1	44:38:39:00:00:17	Add	3m ago	20	leaf02	bond-
swp1						
1	44:38:39:00:00:17	Add	3m ago	20	leaf01	bond-
swp1						
1	44:38:39:00:00:32	Add	4m ago	20	leaf03	bond-
swp2						
1	44:38:39:00:00:32	Add	4m ago	20	leaf04	bond-
swp2						
1	44:38:39:00:00:15	Del	4m ago	20	leaf01	bond-
swp2						
1	44:38:39:00:00:15	Del	4m ago	20	leaf02	bond-
swp2						
1	44:38:39:00:00:32	Del	4m ago	20	leaf03	bond-
swp2						
1	44:38:39:00:00:32	Del	4m ago	20	leaf04	bond-
swp2						
1	44:38:39:00:00:17	Del	4m ago	20	leaf02	bond-
swp1						

```

1      44:38:39:00:00:17    20      leaf01      bond-
swp1      Del      4m ago
Matching IP route records are:
Origin Table      IP
Node      Nexthops      DbState      Last Changed
-----
0      default      ff02::1:ff00:5c/128
spine01      swp1      Del      3m ago
0      default      ff02::1:ff00:12/128
leaf02      eth0      Del      3m ago
No changes to IP neighbor table found
No changes to BGP sessions found
No changes to CLAG session found
No changes to LNV session found

```

You can travel back in time 5 minutes and run a trace from spine02 to exit01, which has the IP address 27.0.0.1:

```

cumulus@leaf01:~$ netq trace 27.0.0.1 from spine02 around 5m
Detected Routing Loop. Node exit01 (now via Local Node exit01 and
Ports swp6 <==> Remote Node/s spine01 and Ports swp3) visited twice.
Detected Routing Loop. Node spine02 (now via mac:00:02:00:00:00:15)
visited twice.
spine02 -- spine02:swp3 -- exit01:swp6.4 -- exit01:swp3 -- exit01
-- spine02:swp7 -- spine02

```

How Far Back in Time Can You Travel?

The NetQ Telemetry Server stores an amount of data limited by a few factors:

- The size of the network: The larger the network, the more complex it is because of the number of routes and nodes.
- The amount of memory in the telemetry server. The more memory, the more data you can retrieve.
- The types of nodes you are monitoring with NetQ. You can monitor just network switches, or switches and hosts, or switches, hosts and containers.
- The number of changes in the network over time.

In general, you can expect to be able to query to a point back in time follows:

Using NetQ to Monitor ...	Data Point	Small Network	Medium Network	Large Network
Switches only	Telemetry server memory minimum	8G	16G	24G
	Years of data retrievable	25.5	17.4	15.6
Switches and Linux hosts		16G	32G	48G

Using NetQ to Monitor ...	Data Point	Small Network	Medium Network	Large Network
	Telemetry server memory minimum			
	Years of data retrievable	4.3	2.7	2.4
Switches, Linux hosts and containers	Telemetry server memory minimum	32G	64G	96G
	Years of data retrievable	2.9	1.5	1.2

The sizing numbers in this table rely on the following assumptions and definitions:

- The types of configuration and operational data being recorded:
 - Switches and hosts: Interfaces; MLAG; LLDP-enabled links; IPv4/v6 addresses, neighbors and routes; BGP sessions; link flaps per day; IPv4/v6 route flaps per day; BGP and MLAG session flaps.
 - Containers: Exposed ports, networks, container flaps per day.
- A small network has 20 racks with 40 leaf nodes, 10 spine nodes and 40 hosts per rack.
- A medium network has 60 racks with 120 leaf nodes, 30 spine nodes and 40 hosts per rack.
- A large network has 100 racks with 200 leaf nodes, 50 spine nodes and 40 hosts per rack.
- The hosts are dual-attached.
- The network is oversubscribed 4:1.
- Adding more memory to the telemetry server allows you to go back even further in time, in a near linear fashion. So doubling the memory should double the range.

Using trace in a VRF

The `netq trace` command works with VRFs as well:

```
cumulus@leaf01:~$ netq trace 10.1.20.252 from spine01 vrf default
around 5m
spine01 -- spine01:swp1 -- leaf01:vlan20
        -- spine01:swp2 -- leaf02:vlan20
```


Monitoring the Physical Layer

NetQ provides the ability to monitor at layer 1 — the physical cabling connecting the nodes of the network fabric. This includes the ability to:

- Manage the inventory: show all optics, determine all the plugged and empty ports, figure out optics expenses by auditing by vendor
- Validate configurations: check peer connections, discover any misconfigured ports, peers, or unsupported modules, check for link flaps
- Investigate errors: including CRC errors

NetQ uses LLDP to collect port information. It can also identify peer ports for DACs and AOCs without using LLDP or even if the link is not UP.

Managing the Layer 1 Inventory

NetQ provides detailed information about the cabling on a given node:

```
cumulus@cel-smallxp-13:~$ netq show interfaces physical
Matching cables records are:
```

Hostname	Interface	State	Speed	AutoNeg	Module	
Vendor	Part No		Last	Changed		
act-5712-12	swp36	down	1G	off	SFP	
AVAGO	AFBR-5715PZ-JU1		9:06:10	ago		
act-5712-12	swp27	up	10G	off	SFP	
OEM	SFP-10GB-LR		17:13:23	ago		
act-5712-12	swp13	up	10G	off	SFP	
JDSU	PLRXPLSCS4322N		17:13:44	ago		
act-5712-12	swp52	up	40G	off	QSFP+	
Mellanox	MC2210130-002		17:13:28	ago		
act-5712-12	swp34	down	10G	off	empty	n
/a	n/a		17:13:47	ago		
act-5712-12	swp37	up	1G	off	SFP	FINISAR
CORP.	FCLF8522P2BTL		17:13:48	ago		
act-5712-12	swp17	up	1G	off	SFP	FINISAR
CORP.	FTLF1318P3BTL		17:13:42	ago		
act-5712-12	swp35	down	1G	off	SFP	CISCO-
AGILENT	QFBR-5766LP		9:06:10	ago		
act-5712-12	eth0	up	1G	on	RJ45	n
/a	n/a		17:13:51	ago		
act-5712-12	swp8	up	10G	off	SFP	
Mellanox	MC2609130-003		17:13:54	ago		
act-5712-12	swp51s3	up	10G	off	QSFP+	
CISCO	AFBR-7IER05Z-CS1		17:13:32	ago		
act-5712-12	swp50s2	up	10G	off	QSFP+	
Mellanox	MC2609130-003		17:13:39	ago		

```

act-5712-12      swp21      up      10G      off      SFP
FIBERSTORE      SFP-10GLR-31      17:13:27 ago
act-5712-12      swp42      up      1G      off      SFP
OEM              SFP-GLC-T      17:13:17 ago
act-5712-12      swp5       up      10G      off      SFP
Mellanox         MC2609130-003      17:13:41 ago
act-5712-12      swp39      up      1G      off      SFP      FINISAR
CORP.      FCLF8522P2BTL      17:13:55 ago
act-5712-12      swp7       up      10G      off      SFP
Mellanox         MC2609130-003      17:13:52 ago
act-5712-12      swp45      up      10G      off      SFP
Mellanox         MC3309130-001      17:13:14 ago
act-5712-12      swp9       up      10G      off      SFP      CISCO-
AVAGO      AFBR-7IER05Z-CS1      17:13:54 ago
act-5712-12      swp48      up      10G      off      SFP
Mellanox         MC3309130-001      17:13:19 ago
act-5712-12      swp2       down    1G      off      SFP      FINISAR
CORP.      FCLF8520P2BTL      13:04:25 ago
act-5712-12      swp41      up      1G      off      SFP      FINISAR
CORP.      FCLF8522P2BTL      17:13:17 ago
act-5712-12      swp50s3    up      10G      off      QSFP+
Mellanox         MC2609130-003      17:13:40 ago

...

```

By running the `netq NODE show interfaces physical module` command, you can see detailed information about the modules on a given node:

```

cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
module
Matching cables records are:
Hostname      Interface Module      Vendor      Part
No      Serial No      Transceiver      Connector      Length
Last Changed
-----
-----
-----
act-5712-12      swp36      SFP      AVAGO      AFBR-5715PZ-
JU1  AM1113SK1A6      1000Base-SX,Mult LC      550m,  9:10:
28 ago

imode,      270m

50um (M5),Multim

ode,

62.5um (M6),Shor

twave laser w/o

```

```

OFC (SN),interme
diate distance (
I)
act-5712-12      swp27      SFP      OEM      SFP-10GB-
LR      ACSLR130408      10G Base-LR      LC      10km,  17:
17:41 ago

10000m
act-5712-12      swp13      SFP      JDSU
PLRXPLSCS4322N  CG03UF45M      10G Base-SR,Mult LC
80m,  17:18:02 ago

imode,      30m,

50um (M5),Multim      300m

ode,

62.5um (M6),Shor

twave laser w/o

OFC (SN),interme
diate distance (
I)
act-5712-12      swp52      QSFP+      Mellanox      MC2210130-
002      MT1539VS03755      40G Base-CR4      n/a      2m      17:
17:46 ago
act-5712-12      swp34      empty      n/a      n
/a      n/a      n/a      n/a      n
/a      17:18:05 ago
act-5712-12      swp37      SFP      FINISAR CORP.
FCLF8522P2BTL      PTN1VH2      1000Base-T      RJ45
100m  17:18:05 ago
act-5712-12      swp17      SFP      FINISAR CORP.
FTLF1318P3BTL      PUC00GG      1000Base-LX,Long LC
10km,  17:17:59 ago

wave laser (LC),      10000m

Longwave laser (
LL),Single Mode

(SM),long distan
ce (L)

```



```

act-5712-12      swp35      SFP      CISCO-AGILENT      QFBR-
5766LP          AGS10335337      1000Base-SX      LC      550m,
9:10:28 ago

270m
act-5712-12      eth0      RJ45      n/a      n
/a              n/a              n/a              n/a      n
/a      17:18:09 ago
act-5712-12      swp8      SFP      Mellanox      MC2609130-
003      MT1507VS05177      1000Base-CX,Copp Copper pigtail      3m      17:
18:12 ago

er Passive,Twin

Axial Pair (TW)
act-5712-12      swp51s3      QSFP+      CISCO      AFBR-7IER05Z-
CS1 AVE1823402U      n/a      n/a      5m      17:17:
49 ago
act-5712-12      swp50s2      QSFP+      Mellanox      MC2609130-
003      MT1507VS05177      40G Base-CR4,Twi n/a      3m      17:
17:57 ago

n Axial Pair (TW
)
...

```

To see empty ports on a node, use the `netq NODE show interfaces physical empty` command:

```

cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
empty
Matching cables records are:
Hostname      Interface State Speed  AutoNeg Module
Vendor        Part No      Last Changed
-----
act-5712-12   swp34       down   10G    off    empty   n
/a            n/a         17:19:10 ago
act-5712-12   swp4        down   10G    off    empty   n
/a            n/a         17:19:14 ago
act-5712-12   swp46       down   10G    off    empty   n
/a            n/a         17:18:42 ago
act-5712-12   swp32       down   10G    off    empty   n
/a            n/a         17:19:03 ago
act-5712-12   swp3        down   10G    off    empty   n
/a            n/a         17:19:15 ago
act-5712-12   swp31       down   10G    off    empty   n
/a            n/a         17:19:12 ago

```

Similarly, to see plugged in ports, run the `netq NODE show interfaces physical plugged` command:

```
cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
plugged
Matching cables records are:
Hostname      Interface State Speed   AutoNeg Module
Vendor        Part No      Last Changed
-----
act-5712-12   swp36       down   1G      off    SFP
AVAGO         AFBR-5715PZ-JU1  9:12:54 ago
act-5712-12   swp27       up     10G     off    SFP
OEM           SFP-10GB-LR      17:20:07 ago
act-5712-12   swp13       up     10G     off    SFP
JDSU          PLRXPLSCS4322N  17:20:28 ago
act-5712-12   swp52       up     40G     off    QSFP+
Mellanox      MC2210130-002   17:20:12 ago
act-5712-12   swp37       up     1G      off    SFP      FINISAR
CORP.         FCLF8522P2BTL  17:20:32 ago
act-5712-12   swp17       up     1G      off    SFP      FINISAR
CORP.         FTLF1318P3BTL  17:20:26 ago
act-5712-12   swp35       down   1G      off    SFP      CISCO-
AGILENT       QFBR-5766LP     9:12:54 ago
act-5712-12   eth0        up     1G      on     RJ45     n
/a            n/a            17:20:35 ago
act-5712-12   swp8        up     10G     off    SFP
Mellanox      MC2609130-003   17:20:38 ago
act-5712-12   swp51s3     up     10G     off    QSFP+
CISCO         AFBR-7IER05Z-CS1 17:20:16 ago
act-5712-12   swp50s2     up     10G     off    QSFP+
Mellanox      MC2609130-003   17:20:23 ago
act-5712-12   swp21       up     10G     off    SFP
FIBERSTORE    SFP-10GLR-31    17:20:11 ago
act-5712-12   swp42       up     1G      off    SFP
OEM           SFP-GLC-T       17:20:01 ago
act-5712-12   swp5        up     10G     off    SFP
Mellanox      MC2609130-003   17:20:25 ago
act-5712-12   swp39       up     1G      off    SFP      FINISAR
CORP.         FCLF8522P2BTL  17:20:39 ago
act-5712-12   swp7        up     10G     off    SFP
Mellanox      MC2609130-003   17:20:36 ago
act-5712-12   swp45       up     10G     off    SFP
Mellanox      MC3309130-001   17:19:58 ago
act-5712-12   swp9        up     10G     off    SFP      CISCO-
AVAGO         AFBR-7IER05Z-CS1 17:20:38 ago
act-5712-12   swp48       up     10G     off    SFP
Mellanox      MC3309130-001   17:20:03 ago
act-5712-12   swp2        down   1G      off    SFP      FINISAR
CORP.         FCLF8520P2BTL  13:11:09 ago
```

```

act-5712-12      swp41      up      1G      off      SFP      FINISAR
CORP.      FCLF8522P2BTL      17:20:01 ago
act-5712-12      swp50s3      up      10G      off      QSFP+
Mellanox      MC2609130-003      17:20:24 ago
act-5712-12      swp43      up      10G      off      SFP
OEM      SFP-H10GB-CU1M      17:20:02 ago
act-5712-12      swp40      up      1G      off      SFP      FINISAR
CORP.      FCLF8522P2BTL      17:20:00 ago
act-5712-12      swp24      up      1G      off      SFP      FINISAR
CORP.      FTLF1318P3BTL      17:20:08 ago

...

```

By searching on specific vendors, you can run a cost analysis of your network:

```

cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
vendor AVAGO
Matching cables records are:
Hostname      Interface State Speed      AutoNeg Module
Vendor      Part No      Last Changed
-----
act-5712-12      swp36      down      1G      off      SFP
AVAGO      AFBR-5715PZ-JU1      9:13:53 ago
act-5712-12      swp29      up      1G      off      SFP
AVAGO      AFCT-5715PZ-JU1      17:21:04 ago

```

You can also search on part numbers using `netq NODE show interfaces physical model PARTNUMBER:`

```

cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
model SFP-H10GB-CU1M
Matching cables records are:
Hostname      Interface State Speed      AutoNeg Module
Vendor      Part No      Last Changed
-----
act-5712-12      swp43      up      10G      off      SFP
OEM      SFP-H10GB-CU1M      17:22:10 ago
act-5712-12      swp44      up      10G      off      SFP
OEM      SFP-H10GB-CU1M      17:22:06 ago
act-5712-12      swp14      up      10G      off      SFP
OEM      SFP-H10GB-CU1M      17:22:36 ago

```

Checking Peer Connections

NetQ checks peer connections using LLDP. For DACs and AOCs, NetQ determines the peers using their serial numbers in the port EEPROMs, even if the link is not UP.

```
cumulus@cel-smallxp-13:~$ netq act-5712-12 show interfaces physical
peer
Matching cables records are:
Hostname          Interface Peer Hostname      Peer Interface State
Message
-----
act-5712-12      swp27    act-5712-12      swp53s0
up
act-5712-12      swp13    cel-red-08       swp6
up
act-5712-12      swp52    dell-s6000-22    swp32
up
act-5712-12      swp34                                down  Port
cage empty
act-5712-12      swp37    dell-s4000-10    swp37
up
act-5712-12      swp17    cel-red-08       swp1
up

...

act-5712-12      swp11    act-5712-12      swp51s1
up
act-5712-12      swp10    act-5712-12      swp51s2
up
act-5712-12      swp3                                down  Port
cage empty
act-5712-12      swp49    act-6712-06      swp32
up
act-5712-12      swp12    act-5712-12      swp51s3
up
act-5712-12      swp23    cel-red-08       swp5
up
act-5712-12      swp31                                down  Port
cage empty
act-5712-12      swp38    dell-s4000-10    swp38
up
act-5712-12      swp47    cel-red-08       swp45
up
act-5712-12      swp51s0  act-5712-12      swp9
up
```

```

act-5712-12      swp50s1  act-5712-12      swp7
up
act-5712-12      swp53s2                                down  Peer
port unknown
act-5712-12      swp53s3                                down  Peer
port unknown
act-5712-12      swp25                                  down  Peer
port unknown
...

```

You can get peer data for a specific port:

```

cumulus@cel-smallxp-13:~$ netq cel-smallxp-13 show interfaces
physical swp31 peer
Matching cables records are:
Hostname          Interface Peer Hostname      Peer Interface State
Message
-----
cel-smallxp-13    swp31      cel-smallxp-13    swp32          up

```

Layer 1 Configuration Checks

You can verify that the following configurations are the same on both ends of two peer interfaces:

- Admin state
- Operational state
- Autonegotiation setting
- Link speed

You can also determine whether a link is flapping or if verify whether both peers are the correct peers. If NetQ can't determine the peer, the port is marked as *unverified*.

To do a layer 1 configuration check, you run the `netq check interfaces` command, which only checks physical interfaces, not bridges, bonds or other software constructs.

```

cumulus@cel-smallxp-13:~$ netq check interfaces
Checked Nodes: 18, Failed Nodes: 8
Checked Ports: 741, Failed Ports: 1, Unverified Ports: 414
Hostname          Interface Peer Hostname      Peer Interface Message
-----
act-5712-12      -          -          Rotten
Agent
act-6712-06      -          -          Rotten
Agent
act-7712-04      -          -          Rotten
Agent

```


cel-smallxp-13	swp2	cel-smallxp-13	swp1	State
mismatch (up, down)				
dell-s4000-10	-	-		Rotten
Agent				
dell-s6000-22	-	-		Rotten
Agent				
mlx-2410-02	-	-		Rotten
Agent				
qct-ly8-04	-	-		Rotten
Agent				

Use the *and* keyword to check the connections between two peers:

```
cumulus@cel-smallxp-13:~$ netq check interfaces cel-smallxp-13 swp2
and mlx-2410-02 swp54
Checked Nodes: 1, Failed Nodes: 1
Checked Ports: 1, Failed Ports: 1, Unverified Ports: 0
Hostname          Interface Peer Hostname    Peer Interface Message
-----
cel-smallxp-13    swp2      mlx-2410-02    swp54      Incorrect
peer specified. Real peer is cel-
smallxp-13 swp1
cumulus@cel-smallxp-13:~$ netq check interfaces cel-smallxp-13 swp1
and mlx-2410-02 swp54
Checked Nodes: 1, Failed Nodes: 0
Checked Ports: 1, Failed Ports: 0, Unverified Ports: 0
```

If a link is flapping, NetQ indicates this in a message:

```
cumulus@cel-smallxp-13:~$ netq check interfaces
Checked Nodes: 18, Failed Nodes: 8
Checked Ports: 741, Failed Ports: 1, Unverified Ports: 414
Hostname          Interface Peer Hostname    Peer Interface Message
-----
dell-s6000-22      -          -              Link
flapped 11 times in last 5
mins
```

Monitoring Linux Hosts with NetQ

Running NetQ on Linux hosts provides unprecedented network visibility, giving the network operator a complete view of the entire infrastructure's network connectivity instead of just from the network devices.

The NetQ Agent is supported on the following Linux hosts:

- CentOS 7
- Red Hat Enterprise Linux 7.1
- Ubuntu 16.04

You need to install the OS-specific NetQ metapack on every host you want to monitor with NetQ. For more information, see the [Host Pack user guide](#).



Monitoring Container Environments with NetQ

The NetQ Agent monitors Docker and Mesos Universal Container Runtime containers the same way it monitors [physical servers \(see page 81\)](#). There is no special implementation. The NetQ Agent pulls Docker data from the container as it would pull data from a Cumulus Linux switch or Linux host.

For more information, see the [Host Pack user guide](#).



Installing and Configuring the NetQ Virtual Environment

A virtual NetQ demo environment is available on the [Cumulus Networks GitHub site](#), allowing you to try out NetQ on your own, or to test/validate updates to your network before deploying them into production.

The environment uses a series of Cumulus VX virtual machines built using the Cumulus Networks [reference topology](#), which requires Vagrant and a hypervisor like VirtualBox. The [GitHub site](#) provides information on downloading and installing the hypervisor.

Restoring from Backups with NetQ

NetQ automatically takes snapshots of the NetQ Telemetry Server at five minute intervals. These snapshots can be used to restore to a previous configuration, or to diagnose existing issues with the configuration. For information regarding how long snapshot data is stored, refer to the [How Far Back in Time Can You Travel](#) section.



There are no configuration steps required for setting up backups. NetQ snapshots occur automatically.

Backup Locations

Backup snapshots can be found in two file locations on the NetQ Telemetry Server:

- `/var/log/backup`: The latest, or master, snapshot.
- `/var/backup`: Directory of previous snapshots.

Use Cases

There are several use-cases in which restoring from a snapshot may be warranted. These include:

- Upgrading the physical server to increase available resources.
- Migrating from one physical server to another.
- A NetQ Telemetry Server crash.

Restoring from a Snapshot

The following steps outline the process for restoring the NetQ Telemetry Server from a snapshot:

1. Extract the GZip snapshot you wish to restore into a file called `appendonly.aof`. The example command below uses the master snapshot:

```
root@cumulus:~# gzip -d < /var/backup/appendonly.aof_master_2017-06-06_054601.gz > appendonly.aof
```

The snapshot filename has several parts:

- `appendonly.aof`: The base file name.
 - `_master_`: Defines this file as the current master snapshot.
 - `2017-06-06_054601`: The date and time the snapshot was taken.
2. Shutdown the NetQ stack:

```
root@cumulus:~# sudo systemctl stop netq-appliance
```

3. Copy the extracted `appendonly.aof` file into the data directory:

```
root@cumulus:~# cp appendonly.aof /var/data/redis/master
/appendonly.aof
```

4. Remove the `dump.rmb` file from the master directory, if the file is present:

```
root@cumulus:~# rm -f /var/data/redis/master/dump.rdb
```

5. Use the `grep` command to confirm the Redis configuration is still set correctly:

```
root@cumulus:~# grep appendonly /etc/cts/redis/*conf
/etc/cts/redis/redis.conf:appendonly yes
/etc/cts/redis/redis.conf:appendfilename "appendonly.aof"
root@cumulus:~# grep 'save "' /etc/cts/redis/*conf
/etc/cts/redis/redis.conf:save ""
```

6. Restart the NetQ Stack:

```
root@cumulus:~# sudo systemctl start netq-appliance
```

Early Access Features

NetQ has [early access](#) features that provide advanced access to new functionality before it becomes generally available. The following features are early access in NetQ 1.1:

In NetQ 1.1, early access features are bundled into the `netq-apps` package; there is no specific EA package like there typically is with Cumulus Linux. You can enable early access features by running the `netq config add experimental` command.



For NetQ 1.1, the only feature enabled by this command is for the [custom commands](#) (see [page 87](#)).

There is nothing to enable for Facebook Backpack; however, the chassis is considered to be an early access feature, so do not run NetQ on it in production.

You can disable the early access features by running the `netq config del experimental` command.

Extending NetQ with Custom Commands

NetQ provides the ability to codify playbooks and extend NetQ with custom commands for use cases specific to your network.

The summary of steps required to do this is as follows:

- The extensions must be written in [Python](#) or [Cython](#).
- The commands that need to be added must use `network doctopt`.
- The .py file (or the compiled .so if using Cython) is now copied to `/usr/lib/python2.7/dist-packages/netq_apps/modules/addons`.
- Enable the add-ons with the `netq config add addons` command
- Check that your command works by typing `netq <TAB>`

Contents

This chapter covers ...

- [Sample File with Custom Command](#) (see [page 88](#))
 - [Command Specification With Help](#) (see [page 88](#))
 - [Associating the Command with the Function](#) (see [page 90](#))
 - [Using the cli and netq Parameters](#) (see [page 90](#))
 - [Return Values](#) (see [page 90](#))
- [Querying the NetQ Database](#) (see [page 90](#))
 - [The Imports](#) (see [page 91](#))
 - [netq_show](#) (see [page 92](#))
 - [Route](#) (see [page 92](#))
 - [The Function Handler](#) (see [page 92](#))

- The Query Functions (see page 92)
- Debugging (see page 92)
- Caveats (see page 93)

Sample File with Custom Command

To help you get started, here is the Hello World of NetQ command extension:

Sample Hello World

```
'''
hello: A netq app hello world module
Usage:
    netq hello [json]
Options:
    hello                : Hello world experimental
'''
import json
from netq_apps.modules import NetqModule, RC_SUCCESS, RC_FAIL
app = NetqModule()

@app.route('hello')
def cli_hello_world(cli, netq):
    '''My very own hello'''
    jsonify = cli.get('json')
    if jsonify:
        print json.dumps({'greeting': 'Hello World'})
    else:
        print 'Hello World'
    return RC_SUCCESS
```

Let's break down each part of the code.

Command Specification With Help

The lines at the start of the file within the triple quotes (") constitute what is called the *docstring* of the file or module. `network-docopt`, the Python library that builds the command parser for NetQ, uses the information provided in the *docstring*. Specifically, everything between **Usage** and **Options** is considered a command specification. In this case, `netq hello` is the only command specified in the file. The command MUST start with the word `netq`. Every `netq` command follows the following structure:

```
netq [<hostname>] <verb> <object> <filters>
```

For example, here is the sample for `show vlan`:

```
netq [<hostname>] show vlan [<1-4096>] [around <text-time>] [json]
```


The `<hostname>` option is used to filter results to just the specified host; hostname can also be a regular expression. The `<verb>` is `show`, the `<object>` is `vlan` and the remaining parameters are filters to viewing the data.

For example, if you wanted to extend hello world by passing an optional greeting, modify the usage to be:

```
netq hello <text-greeting>
```

network-docopt understands a few parameter types and validates them before passing them to your code. Some common ones are:

- **<hostname>**: A host known to NetQ
- **<remote-interface>**: An interface on the specified host known to NetQ
- **<text>**: Any free text, but has to be a single word or delimited within quotes
- **<ip>**, **<ip/prefixlen>**: IPv4 or IPv6 address, with prefix length in the second case
- **<ipv4>**, **<ipv4/prefixlen>**: IPv4 address, with prefix length in the second case
- **<ipv6>**, **<ipv6/prefixlen>**: IPv6 address, with prefix length in the second case
- **<wildcard>**: All the remaining text
- Valid number range: Such as **<1-4096>** to limit the allowed range

So in the VLAN example above, specifying a VLAN value outside the 1-4096 range results in an error, with command unknown and a help message indicating that you need to specify a value between 1 and 4096. For hosts and interfaces used with `<hostname>` and `<remote-interface>`, NetQ automatically provides tab completion.

To display meaningful help associated with a keyword, add the help for the command via the **Options** section. In the example code above, the object `hello` has the help text "Hello world experimental". This text is displayed when the user types `netq <TAB>`, as shown in the following example:

```
cumulus@switch:~$ netq
<hostname> : Type first char of netq host for dynamic completion
check : Perform fabric-wide checks
config : Configuration
example : Show examples of usage and workflow
hello : Hello world experimental
help : Show usage info
resolve : Annotate input with names and interesting info
show : Show fabric-wide info about specified object
trace : Control Path Trace
cumulus@torc-11:mgmt-vrf:~$ netq
```



Any help you provide here overrides the help provided for the keyword by a module loaded previously.

Associating the Command with the Function

After configuring the command, you need to associate or *bind* that command with the function to be called when a user runs the command. This is done by using decorators to functions similar to how other CLI builders or web servers work.

First, create an instance of the class `NetqModule()` called *app*. Then associate the function to the appropriate command via the decorator `@app.route`. As shown in the example above, the function `cli_hello_world()` is decorated to indicate that it is the function to call for the command `hello`. The function takes two parameters: *cli* and *netq*. Usage of these parameters is discussed in the next section.

Keep in mind the following when matching the command to the function:

- If a prior binding has already been assigned to a command, the newer binding will fail. By default, modules in the core NetQ code take precedence over early access modules, which take precedence over the modules defined in addons directory.
- The command string can be as small as possible. For example, the commands `netq hello json` and `netq hello` can be handled by different functions or by the same function. The NetQ command parser does a longest match first to determine which of the competing functions is assigned to execute a command. In NetQ 1.1, the command parser supports up to three string matches. In other words, `show ip address` is supported, but `show ip address json` is not. Such longer command strings bound to a function either silently fail or a shorter string version is matched.

Using the cli and netq Parameters

The function that is called to execute a command expects to receive two parameters, *cli* and *netq*, in the order shown in the example above.

cli is a dictionary containing the parameters provided by the user on the command line. *netq* contains the timestamps provided by the user, if any. Any other object within NetQ can be ignored. The timestamps are provided to query NetQ objects around a specific time or in a time window.

The example shows how to extract the value provided by the user at the command line from *cli*. Since *json* is a keyword, getting the key *json* from *cli* lets you to determine if the user specified *json* at the command line or not. If the user did not specify *json* at the command line, `cli.get('json')` returns *None*, whereas if the user did specify *json*, then `cli.get('json')` returns the string "json". Thus, if the user wants to specify a parameter along with a keyword, for example, as shown in `netq show macs [vlan <1-4096>]`, then the value of the VLAN to search for a MAC address can be found using `cli.get('<1-4096>')`, not via `cli.get('vlan')`.

Return Values

The function returns either *RC_SUCCESS* if successful or *RC_FAIL* if not. The code snippet shows how to import these values from the standard NetQ libraries.

Querying the NetQ Database

While the code snippet above was sufficient to illustrate the general skeleton, if you want to extend the commands, you typically will want to add meaningful functionality such as querying the database and displaying some more meaningful information. For example, consider a new command called `show ip-routes`, which displays the route information available in the database, but with a different set of fields than shown via `show ip routes`. The code to do so is shown below.

```

"""
routes.py: NetQ app module for processing IPv4/v6 routes
Usage:
    netq <hostname> show myroutes [vrf <vrf>] [json]
Options:
    myroutes                                : IPv4/v6 routes
"""
from __future__ import absolute_import
from collections import OrderedDict

from netq_apps.modules import NetqModule, RC_SUCCESS
from netq_apps.cmd.netq import netq_show

from netq_lib.orm.redisdb.models import Route

app = NetqModule()

@app.route('show myroutes')
@netq_show
def cli_show_myroutes(cli, netq, context):
    '''MY very own show routes'''
    hostname = cli.get('<hostname>') or '*'
    vrf = cli.get('<vrf>') or '*'
    context.col_sizes = [16, 8, 32, 26, 16]
    entries = Route.query.filter(timestamp=netq.start_time,
                                endtimestamp=netq.end_time,
                                hostname=hostname, vrf=vrf)

    for entry in entries:
        out = OrderedDict()
        if isinstance(entry, tuple):
            route = entry[0]
        else:
            route = entry
        if not route.nexthops:
            route.nexthops = [['None', 'Local']]
        nexthops = ', '.join(
            '%s: %s' % (nh[0], nh[1]) if nh[0] != 'None' else '%s' %
nh[1]
            for nh in sorted(route.nexthops)
        )

        out['Hostname'] = route.hostname
        out['Protocol'] = route.protocol
        out['Prefix'] = route.prefix
        out['Nexthops'] = nexthops
        out['Last Changed'] = route.timestamp
        yield out

```

Much of this code is similar to the hello world example, but the new items are discussed below.

The Imports

There are two additional imports, one for *netq_show* and the other for *Route*.

netq_show

netq_show is the decorator that takes care of wrapping the output in a format native to NetQ. For example, it generates the JSON for you automatically, so that you don't have to write a JSON output generator just to support JSON and you don't have to worry about supporting the tabular format, displaying rotten nodes in a different color and so on. All you have to do is generate output in the form of an `OrderedDict` and `yield` for every entry. The `OrderedDict` ensures that the columns are displayed in the order provided in the code. The column headers are generated from the dictionary key, as are the JSON keys.

By wrapping the code with the *netq_show*, all these display complexities are covered for you.

Route

Route is the database object that holds all the pertinent information about a route. Its contents are defined in the `/usr/lib/python2.7/dist-packages/netq_lib/orm/redisdb/models.py` file. There are other database objects defined in the file, but this example only involves the *Route* object.

The Function Handler

The function that satisfies the command `show myroutes` is *cli_show_myroutes*, and because of the decorator, takes an additional input parameter, *context*. It's mainly used to pass things between the main NetQ command module and the specific modules, such as this one. This particular case uses the *context* to update the column sizes to be used in the display.

The Query Functions

The meat of the code is the query. Objects are queried using the model of `<object>.query.<query function>`. This particular example uses *filter* as the query function, as shown by the `Route.query.filter()` call. The filter function produces output filtered by the parameters specified in the keyword arguments passed. For example, the *hostname* keyword argument restricts the results returned by the query function to only those on the specified host. The list of keys that can be specified for an object are listed under the object's definition in the aforementioned `models.py` file under the function `key_fmt()`. A look at that function for the *Route* object shows that the key fields are: hostname, prefix, route type, routing table id, ipv4/v6 route and, if the entry is originated on this node, the protocol that added this route and the VRF name qualifier. The values returned include all the key fields plus the fields shown in the `val_fmt()` function for the object.

The other useful query functions are:

- `query.get()`: which returns just the first element matching the parameters specified.
- `query.latest()`: which returns the latest element matching the parameters specified, and does not take any time parameters.
- `query.count()`: which returns a count of the matching elements instead of the elements themselves.

The filter query functions return an iterator and thus is lazy about retrieving data from the back end. You can stop whenever you want in the iteration. `query.get()` and `query.latest()` both return a single object of the type the query is on while `query.count()` returns an integer.

Debugging

Inevitably when writing code, coding errors need to be debugged and the fixes tried again. When a module doesn't load or returns an error, it is reported in the `netqd.log`, usually kept under `/var/log` (unless you modified the location). Deploying the module on one node doesn't mean it is automatically available on all nodes. You must copy it to all the required nodes.

To reload the modules after making fixes, run the command `netq config reload parser`.

Caveats

This feature is an early access feature, and must be treated as such. There may be obscure failures which will require Cumulus Networks engineering intervention to investigate. Finally, please save the modules you write. If you reinstall the `netq-apps` package, your modules may get overwritten when you install the new package. One of the next releases of NetQ should provide the ability to store these modules under `/usr/local/lib`, to keep them from being affected by package management.

Facebook Backpack Integration

NetQ can run within a [Facebook Backpack chassis](#), but it is considered to be an [early access](#) (see page 86) feature.

Keep the following issues in mind if you intend to use NetQ with a Backpack chassis:

- You must assign a unique hostname to every node that runs the NetQ Agent. By default, all the fabric cards in the chassis have the same hostname.
- The NetQ Agent must be installed on every line card.
- No information is returned about the ASIC when you run `netq show inventory asic`. This is a known issue.
- Since the chassis sensor information is shared among, every line card and fabric card can report the same sensor data. By default, sensor data is disabled on a chassis. To enable sensor data on a line card, edit `/etc/cts/netq/netq.yml` or `/etc/cts/netq/config.d/user.yml` and set the `send_chassis_sensor_data` keyword to `true`, then restart the NetQ Agent with `netq config agent restart`. This prevents any duplication of data in the NetQ database.

```
cumulus@backpack-lc101:~$ sudo nano /etc/cts/netq/netq.yml

...

netq-agent:
  send_chassis_sensor_data: true

...
```

Troubleshooting NetQ

To aid in troubleshooting issues with NetQ, there are several configuration and log files on the **telemetry server** that can provide insight into the root cause of the issue:

File	Description
/etc/cts/netq/netq.yml	The NetQ Telemetry Server configuration file.
/var/log/docker/cts-redis.log	The Redis log file.
/var/log/docker/cts-docker-compose.log	The backup log file.
/var/log/netqd.log	The NetQ daemon log file for the NetQ CLI.
/var/log/netq-notifier.log	The NetQ Notifier log file.

A **node** running the NetQ Agent has the following configuration and log files:

File	Description
/etc/netq/netq.yml	The NetQ configuration file.
/var/log/netq-agent.log	The NetQ Agent log file.
/etc/netq/config.d/netq-agent-commands.yml	Contains key-value command pairs and relevant custom configuration settings.
/run/netq-agent-running.json	Contains the full command list that will be pushed when the agent starts.

Checking Agent Health

Checking the health of the NetQ agents is a good way to start troubleshooting NetQ on your network. If any agents are rotten, meaning three heartbeats in a row were not sent, then you can investigate the rotten node. In the example below, the NetQ Agent on server01 is rotten, so you know where to start looking for problems:

```
netq@446c0319c06a:/$ netq check agents
Checked nodes: 12,
```

```
Rotten nodes: 1
netq@446c0319c06a:/$ netq show agents
Node          Status      Sys Uptime    Agent Uptime
-----
exit01
Fresh
      8h ago      4h ago
exit02
Fresh
      8h ago      4h ago
leaf01
Fresh
      8h ago      4h ago
leaf02
Fresh
      8h ago      4h ago
leaf03
Fresh
      8h ago      4h ago
leaf04
Fresh
      8h ago      4h ago
server01
Rotten
      4h ago      4h ago
server02
Fresh
      4h ago      4h ago
server03
Fresh
      4h ago      4h ago
server04
Fresh
      4h ago      4h ago
spine01
Fresh
      8h ago      4h ago
spine02
Fresh
      8h ago      4h ago
```



Error Configuring the Telemetry Server on a Node

If you get an error when you run the `netq config add server` command on a node, it's usually due to one of two reasons:

- The hostname or IP address for the telemetry server was input incorrectly when you ran `netq config add server`. Check what you input and try again.
- The telemetry server isn't responding. Try pinging the IP address you entered and see if the ping works.

cts-support

The `cts-support` command generates an archive of useful information for troubleshooting issues with NetQ. The Cumulus Networks support team may request the output of this command when assisting with any issues that you could not solve with your own troubleshooting. Run the following command on the telemetry server:

```
cumulus@ts:~$ cts-support
```




Index