Cumulus NetQ

Cumulus® NetQ is a network operations tool set that provides actionable insight into and operational intelligence about the health of the entire Linux-based data center — from the container, virtual machine, or host, all the way to the switch and port. Working hand-in-hand with Cumulus Linux, NetQ 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, providing cloud architects and network operations teams the ability to operate with visibility into the entire network. It is integrated with container orchestrators and the Netlink interface to make this happen. With NetQ, network operations changes from a manual, reactive, box-by-box approach to an automated, informed and agile one.

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

- **Preventative Validation**: NetQ easily validates potential network configuration changes in a virtualized environment or lab using check, show and trace algorithms. NetQ eliminates the need to check switches or servers one by one and can reduce manual errors before they are rolled into production (one of the main causes of network downtime).
- **Proactive Alerting**: NetQ detects faulty network states that can result in packet loss or connectivity issues, and alerts the user with precise fault location data for faster remediation, greatly improving network agility and reducing downtime costs.
- **Diagnostic Troubleshooting**: NetQ provides the ability to trace network paths, replay the network state at a time in the past, review fabric-wide event change logs, and diagnose the root cause of state deviations.

NetQ also offers Image and Provisioning Management (IPM), which makes it possible to get your new Cumulus Linux switches up and running quickly by providing bootstrapping and lifecycle management capabilities, including image and ZTP configuration management. IPM contains local storage and distribution services for the Cumulus Linux network operating system (NOS) and provisioning scripts used to deploy and upgrade Cumulus Linux and NetQ. With IPM, network deployment changes from a tedious box-by-box installation process to a consistent and predictable one.

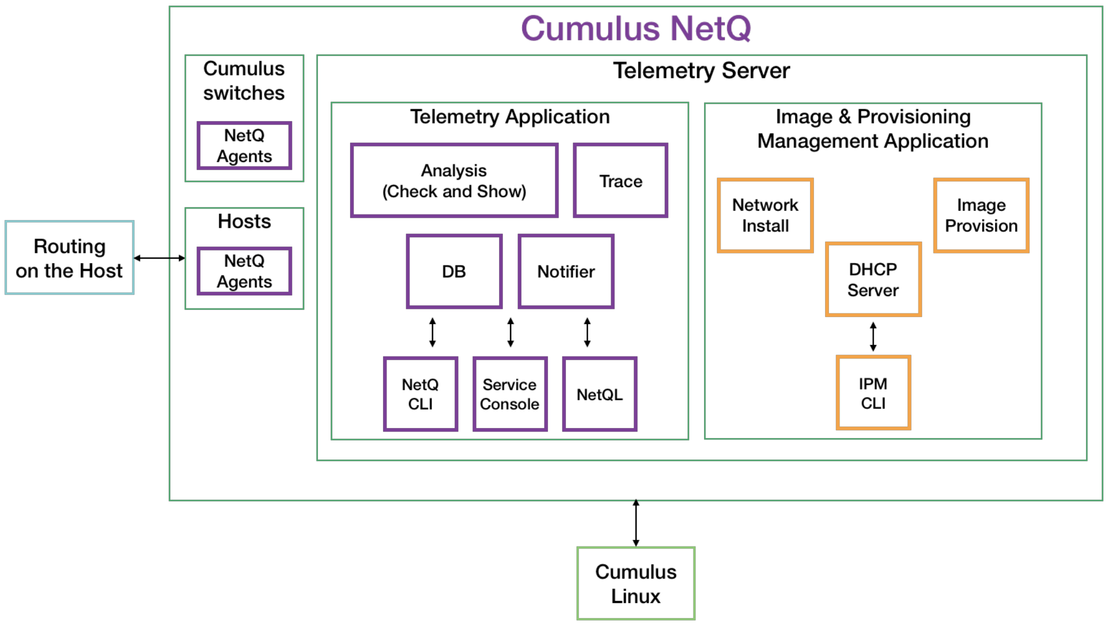
Contents

This topic describes...

Cumulus NetQ Components

NetQ contains the following applications and key components:

- Telemetry
  - NetQ Switch Agents (data collector)
  - NetQ Host Agents (data collector)
  - NetQ Telemetry Server (storage and processing)
- Image and Provisioning Management
  - ONIE (Open Network Install Environment)
  - DHCP (Dynamic Host Control Protocol) server
  - ZTP (Zero-Touch Provisioning)
- Various User Interfaces



NetQ interfaces with Cumulus Linux and Cumulus Routing on the Host products.

Telemetry Components

Each of the Telemetry components used to gather, store and process data about the network state are described here.

NetQ Agents

NetQ Agents are back-end Python software installed on every monitored *node* in the network — including Cumulus® Linux® switches, Linux bare-metal hosts and virtual machines, and Docker containers. The NetQ Agents push network data every 15 seconds and event information immediately to the NetQ Telemetry Server.

Switch Agents

The NetQ Agents running on Cumulus Linux switches gather the following network data via Netlink:

- Interfaces
- IP addresses (v4 and v6)
- IP routes (v4 and v6)
- Links
- Bridge FDB (forwarding database)
- Neighbors (IPv4 and IPv6)

for the following protocols:

- Bridging protocols: LLDP, STP, MLAG
- Routing protocols: BGP, OSPF
- Network virtualization: LNV, VXLAN, EVPN

The NetQ Agent is supported on Cumulus Linux 3.3.0 and later.

Host Agents

The NetQ Agents running on hosts gather the same information as that for switches, plus the following network data:

- NAT (network address translation) changes
- Network IP and MAC addresses
- Container IP and MAC addresses

The NetQ Agent obtains container information by listening to the Docker Swarm or Kubernetes orchestration tools.

The NetQ Agent is supported on hosts running Ubuntu 16.04, Red Hat® Enterprise Linux 7, and CentOS 7 Operating Systems.

NetQ Telemetry Server

The NetQ Telemetry Server provides the validation and notification capabilities of NetQ. It hosts a database/key-value store, NetQ Notifier, Analysis Engine, and Trace Engine.

- The database contains all network state and event information received from NetQ Agents
- NetQ Notifier responds to events pushed by the NetQ Agents, sending alerts to `rsyslog`, or to a third-party notification application (Slack, PagerDuty, Splunk, or ELK Logstash)
- The Analysis Engine validates service network-wide and displays details on the services
- The Trace Engine tracks faults across available paths between two switches or servers in the network

NetQ Telemetry User Interfaces

NetQ telemetry functionality is available through several user interfaces:

- NetQ Command Line Interface (CLI) provides access to network telemetry data
- NetQ Service Console provides a browser-based window for accessing the NetQ CLI from anywhere.
- NetQL provides direct access to the data base using a custom query language

All of these interfaces query the Telemetry Server database for network state and event information.

Data Center Network Deployments

There are three key deployment types that are commonly deployed for network management in the data center:

- Out-of-Band Management
- Inband Management
- High Availability

A summary of each type is provided here.

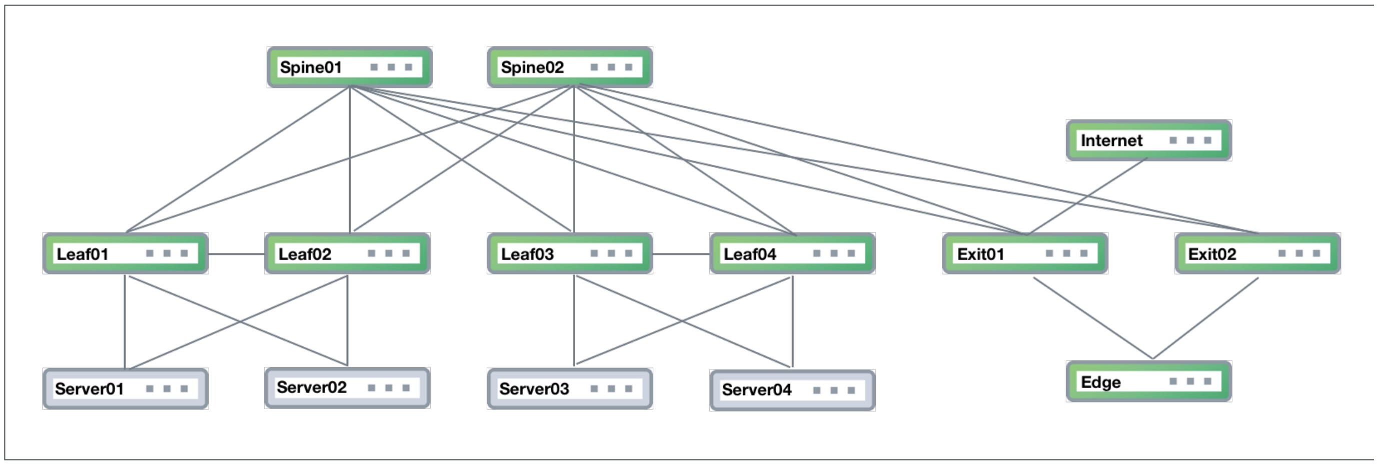
Out-of-Band Management Deployment

Cumulus Networks recommends deploying NetQ on an out-of-band (OOB) management network to separate network management traffic from standard network data traffic, but it is not required. This figure shows a sample CLOS-based network fabric design for a data center using an OOB management network overlaid on top, where NetQ is deployed.

The physical *network* hardware includes:

- **Spine** switches: where data is aggregated and distributed; also known as an aggregation switch, end-of-row (EOR) switch or distribution switch
- **Leaf** switches: where servers connect to the network; also known as a Top of Rack (TOR) or access switch
- **Server** hosts: where applications are hosted and data served to the user through the network
- **Exit** switch: where connections to outside the data center occur; also known as Border Leaf or Service Leaf
- **Edge** server (optional): where the firewall is the demarcation point, peering may occur through the exit switch layer to Internet (PE) devices
- **Internet** device (PE): where provider edge (PE) equipment communicates at layer 3 with the network fabric

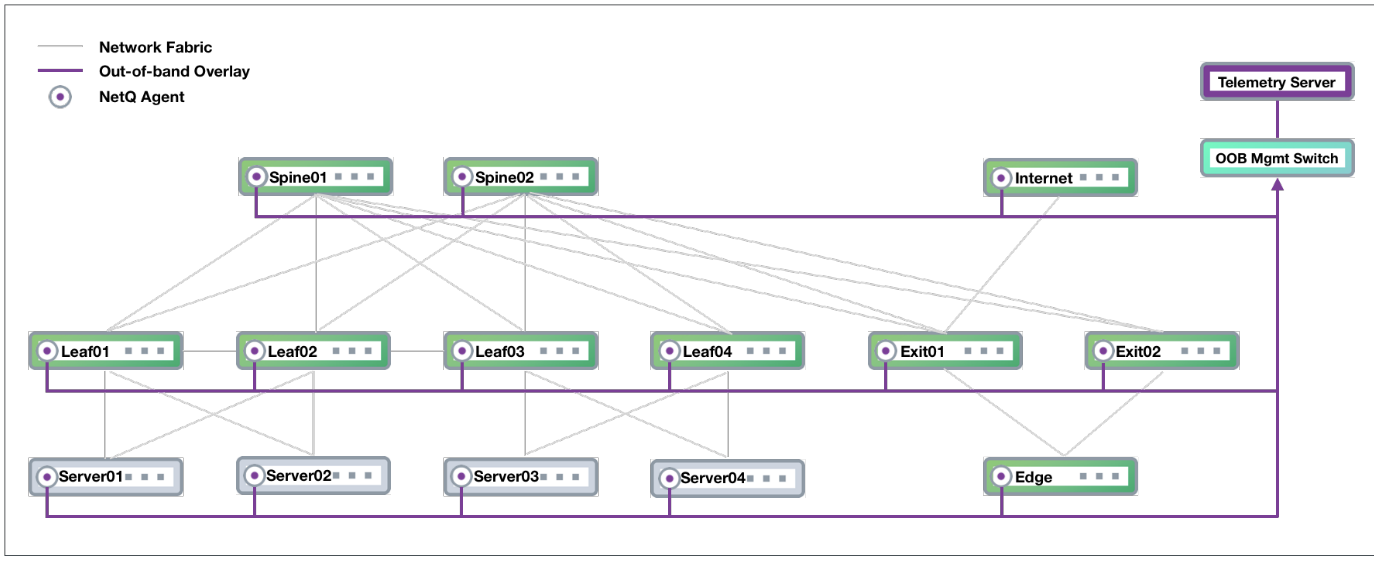
The diagram shows physical connections (in the form of grey lines) between Spine 01 and four Leaf devices and two Exit devices, and Spine 02 and the same four Leaf devices and two Exit devices. Leaf 01 and Leaf 02 are connected to each other over a peerlink and act as an MLAG pair for Server 01 and Server 02. Leaf 03 and Leaf 04 are connected to each other over a peerlink and act as an MLAG pair for Server 03 and Server 04. The Edge is connected to both Exit devices, and the Internet node is connected to Exit 01.



The physical *management* hardware includes:

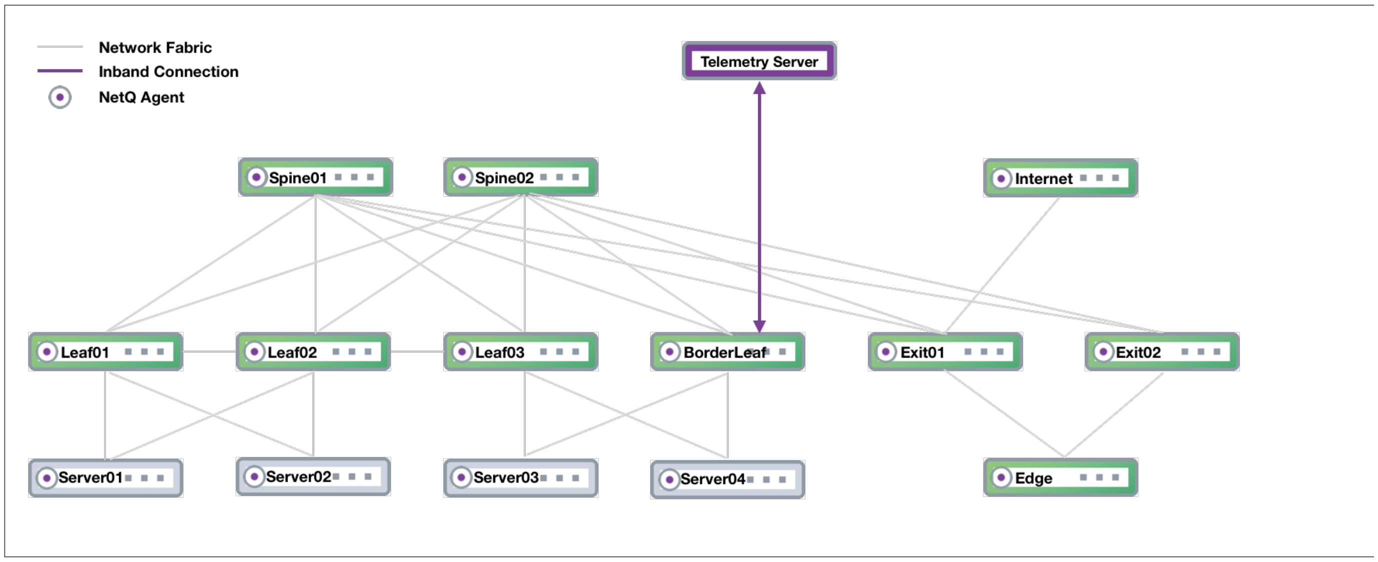
- OOB Mgmt Switch: aggregation switch that connects to all of the network devices through communications with the NetQ Agent on each node
- NetQ Telemetry Server: hosts the telemetry software, database and user interfaces (refer to description above).

These switches are connected to each of the physical network devices through a virtual network overlay, shown with purple lines.



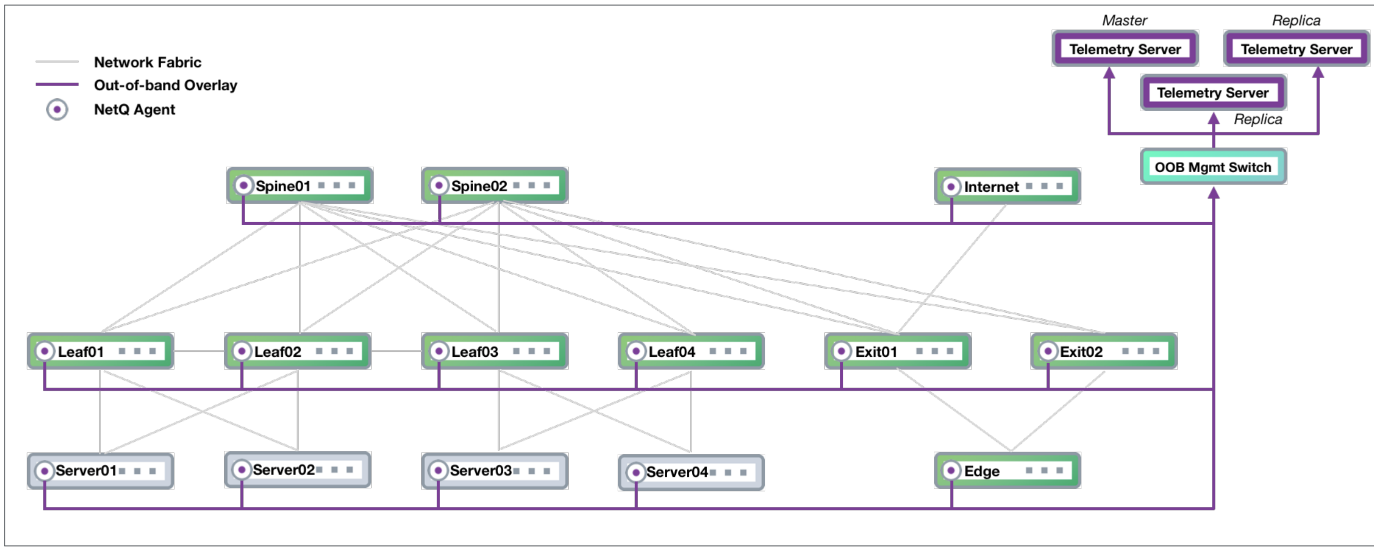
Inband Management Deployment

While not the preferred deployment method, you might choose to implement NetQ within your data network. In this scenario, there is no overlay and all traffic to and from the NetQ Agents and the Telemetry Server traverses the data paths along with your regular network traffic. The roles of the switches in the CLOS network are the same, except that the Telemetry Server performs the aggregation function that the OOB management switch performed. If your network goes down, you might not have access to the Telemetry Server for troubleshooting.



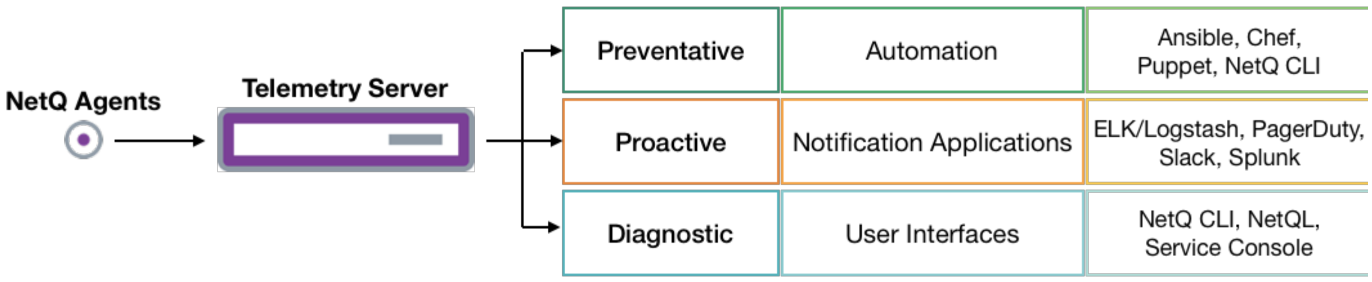
High Availability Deployment

NetQ supports a high availability deployment for users who prefer a solution in which the collected data and processing provided by the Telemetry Server remains available through alternate equipment should the TS fail for any reason. In this configuration, three TSs are deployed, with one as the master and two as replicas. Data from the NetQ Agents is sent to all three switches so that if the master TS fails, one of the replicas automatically becomes the master and continues to store and provide the telemetry data. This example is based on an OOB management configuration, and modified to support high availability for NetQ.



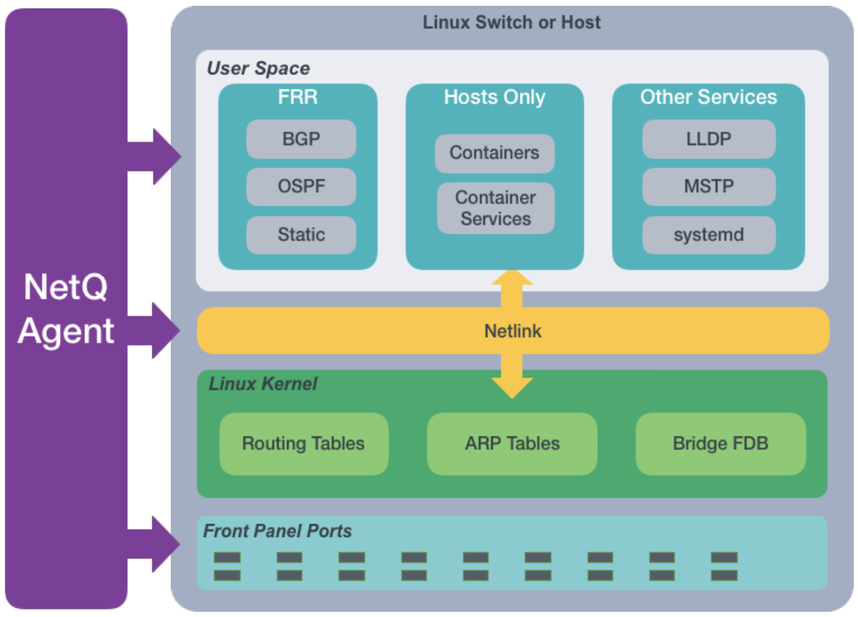
Telemetry Operation

In any of the above deployments, NetQ offers preventative management and proactive monitoring capabilities in the form of integration with automation scripts and notification applications. It also provides multiple interfaces for diagnosing the network performance and configuration.


NetQ Agents
Telemetry Server
Preventative
Automation
Ansible, Chef, Puppet, NetQ CLI
Proactive
Notification Applications
ELK/Logstash, PagerDuty, Slack, Splunk
Diagnostic
User Interfaces
NetQ CLI, NetQL, Service Console

The NetQ Agent

From a software perspective, a network switch has software associated with the hardware platform, the operating system, and communications. For data centers, the software on a Linux network switch would be similar to the diagram shown here.



The NetQ Agent interacts with the various components and software on switches and hosts and provides the gathered information to the NetQ Telemetry Server (TS). You can view the data using the NetQ CLI or Service Console. NetQL (an early access feature) enables you to query the data in a more customized fashion.

The NetQ Agent polls the user space for information about the performance of the various routing protocols and services that are running on the switch. Cumulus Networks supports BGP and OSPF Free Range Routing (FRR) protocols as well as static addressing. Cumulus Linux also supports LLDP and MSTP among other protocols, and a variety of services such as systemd and sensors. For hosts, the NetQ Agent also polls for performance of containers managed with Docker or Kubernetes orchestrators. All of this information is used to provide the current health of the network and verify it is configured and operating correctly.

For example, if the NetQ Agent learns that an interface has gone down, a new BGP neighbor has been configured, or a container has moved, it provides that information to the TS. That information can then be used to notify users of the operational state change through various channels. By default, data is logged in NetQ and visible in `rsyslog`, but you can configure the Notifier component in NetQ to send the information to a third-party notification application as well. NetQ supports ELK/Logstash, PagerDuty, Slack, and Splunk integrations.

The NetQ Agent interacts with the Netlink communications between the Linux kernel and the user space, listening for changes to the network state, configurations, routes and MAC addresses. NetQ uses this information to enable notifications about these changes so that network operators and administrators can respond quickly when changes are not expected or favorable.

For example, if a new route is added or a MAC address removed, NetQ Agent records these changes and sends that information to the NetQ Telemetry Server. Based on the configuration of the Notifier component, these changes can be sent to a variety of locations for end user response.

The NetQ Agent also interacts with the hardware platform to obtain performance information about various physical components, such as fans and power supplies, on the switch. Operational states and temperatures are measured and reported, along with cabling information to enable management of the hardware and cabling, and proactive maintenance.

For example, as thermal sensors in the switch indicate that it is becoming very warm, various levels of alarms are generated. These are then communicated through notifications according to the Notifier configuration.

Chassis are also supported with four or eight line cards (a switch on a circuit board), where each line card has its own NetQ Agent to provide the same level of information as if it were a standalone switch.

The Telemetry Server

Once the collected data is sent to and stored in the NetQ Telemetry Server database, you can:

- Validate configurations, identifying misconfigurations in your current network, in the past, or prior to deployment,
- Monitor communication paths throughout the network,
- Notify users of issues and management information,
- Anticipate impact of connectivity changes,

- and so forth.

Validate Configurations

The NetQ CLI enables validation of your network health through three major sets of commands. They extract the information from the analysis engine, trace engine, and notifier. The analysis engine is continually validating the connectivity and configuration of the devices and protocols running on the network. Using the check and show commands displays the status of the various components and services on a network-wide and complete software stack basis. For example, you can perform a network-wide check on BGP with a single `netq check bgp` command. The command lists any devices that have misconfigurations or other operational errors in seconds. When errors or misconfigurations are present, using the `netq show bgp` command displays the BGP configuration on each device so that you can compare and contrast each device, looking for potential causes. Check and Show commands are available for numerous components and services as shown in the following table.

Component or Service	Check	Show	Component or Service	Check	Show
Agents	★	★	LLDP		★
BGP	★	★	LNV	★	★
CLAG (MLAG)	★	★	MACs		★
Docker		★	MTU	★	★
EVPN	★	★	NTP	★	★
Interfaces	★	★	OSPF	★	★
Inventory		★	Sensors	★	★
IPv4/v6		★	Services		★
Kubernetes		★	VLAN	★	★
License	★	★	VXLAN	★	★

Monitor Communication Paths

The trace engine is used to validate the available communication paths between two network devices. The corresponding `netq trace` comman d enables you to view all of the paths between the two devices and if there are any breaks in the paths. For example, you can check the paths between two hosts. This example output shows eight successful paths between server02 and server03, all with an MTU of 9152.

```
cumulus@ts:~$ netq trace 10.244.40.193 from 10.244.6.6
Number of Paths: 8
Number of Paths with Errors: 0
Number of Paths with Warnings: 0
Path MTU: 9152
   server02:eth1 -- leaf01:swp51 -- spine01:swp3 -- leaf03:swp1 --
server03:cali320be3ff5dd
                                   -- spine01:swp4 -- leaf04:swp1 --
server03:cali320be3ff5dd
                   -- leaf01:swp52 -- spine02:swp3 -- leaf03:swp1 --
server03:cali320be3ff5dd
                                   -- spine02:swp4 -- leaf04:swp1 --
server03:cali320be3ff5dd
     server02:eth2 -- leaf02:swp51 -- spine01:swp3 -- leaf03:swp1 --
server03:cali320be3ff5dd
                                   -- spine01:swp4 -- leaf04:swp1 --
server03:cali320be3ff5dd
                   -- leaf02:swp52 -- spine02:swp3 -- leaf03:swp1 --
server03:cali320be3ff5dd
                                   -- spine02:swp4 -- leaf04:swp1 --
server03:cali320be3ff5dd
```

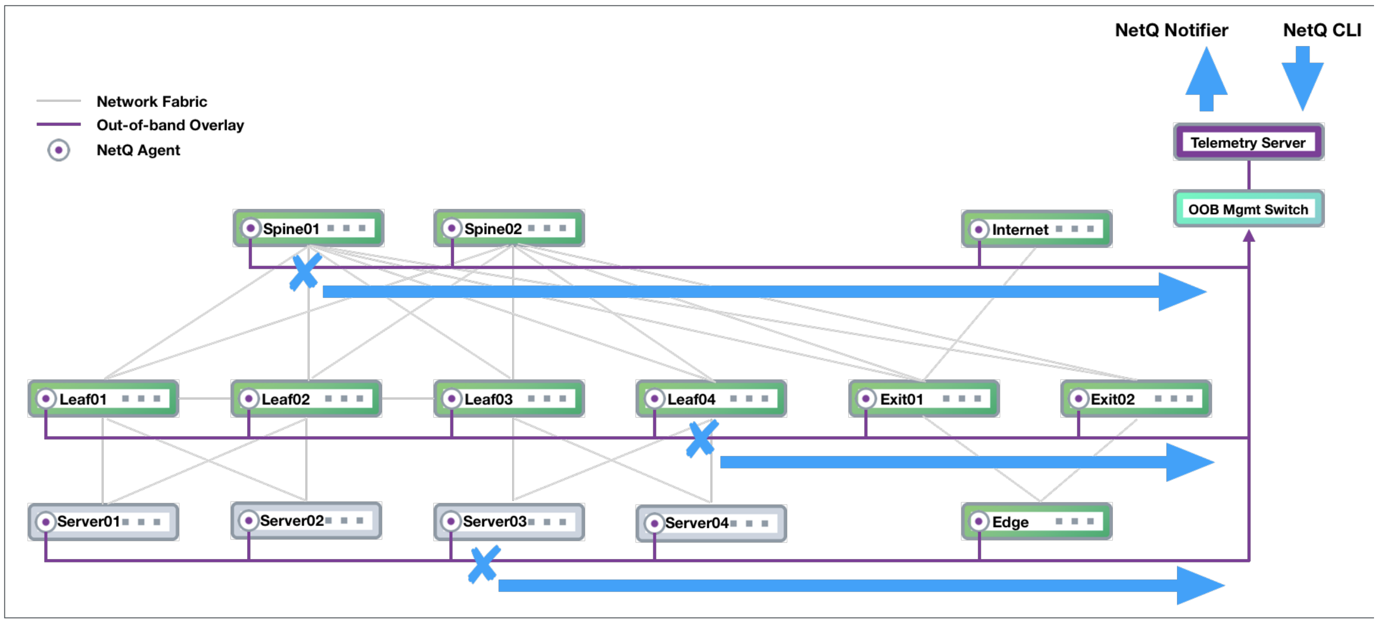
This output is read as:

- Path 1 traverses the network from server02 interface eth1 to leaf01 interface swp51 to spine01 interface swp3 to leaf03 interface swp1 to server03.
- Path 2 traverses the network from server02 interface eth1 to leaf01 interface swp51 to spine01 interface **swp4** to **leaf04** interface swp1 to server03.
- Path 3 traverses the network from server02 interface eth1 to leaf01 interface **swp52** to spine01 interface **swp3** to **leaf03** interface swp1 to server03.
- Path 4 traverses the network from server02 interface eth1 to leaf01 interface swp52 to spine01 interface **swp4** to **leaf04** interface swp1 to server03.
- Path 5 traverses the network from server02 interface **eth2** to leaf01 interface swp51 to spine01 interface **swp3** to **leaf03** interface swp1 to server03.
- Path 6 traverses the network from server02 interface eth2 to leaf01 interface swp51 to spine01 interface **swp4** to **leaf04** interface swp1 to server03.
- Path 7 traverses the network from server02 interface eth2 to leaf01 interface **swp52** to spine01 interface **swp3** to **leaf03** interface swp1 to server03.
- Path 8 traverses the network from server02 interface eth2 to leaf01 interface swp52 to spine01 interface **swp4** to **leaf04** interface swp1 to server03.

where **bold** text highlights the differences in the paths. If the MTU does not match across the network, or any of the paths or parts of the paths have issues, that data is called out in the summary at the top of the output and shown in red along the paths, giving you a starting point for troubleshooting.

View Historical State and Configuration

All of the check, show and trace commands can be run for the current status and for a prior point in time. For example, this is useful when you receive messages from the night before, but are not seeing any problems now. You can use the `netq check` command to look for configuration or operational issues around the time that the messages are timestamped. Then use the `netq show` commands to see information about how the devices in question were configured at that time or if there were any changes in a given timeframe. Optionally, you can use the `netq trace` command to see what the connectivity looked like between any problematic nodes at that time. This example shows problems occurred on spine01, leaf04, and server03 last night. The network administrator received notifications and wants to investigate. The diagram is followed by the commands to run to determine the cause of a BGP error on spine01. Note that the commands use the `around` option to see the results for last night and that they can be run from any switch in the network.

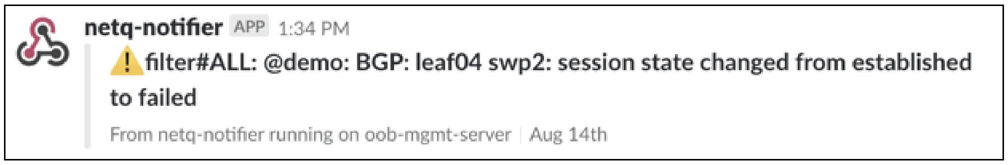


```
cumulus@switch:~$ netq spine01 check bgp around 10h
Total Sessions: 16, Failed Sessions: 2
Node     Neighbor    Peer ID    Reason              Time
-------  ----------  ---------  ------------------  -------
leaf03   swp51       spine01    Interface down      10h ago
spine01  swp3        leaf03     Hold Timer Expired  10h ago

cumulus@switch:~$ netq spine01 show bgp around 10h

Matching bgp records:
Hostname          Neighbor                     VRF             ASN
Peer ASN   PfxRx        Last Changed
----------------- ---------------------------- --------------- ----------
---------- ------------ ------------------------
Spine01           swp1(leaf01)                 default         65020
65011      2/-/-        1d:4h:55m:0s
Spine01           swp2(leaf02)                 default         65020
65012      2/-/-        1d:4h:55m:0s
Spine01           swp3(leaf03)                 default         65020
65013      2/-/-        1d:4h:55m:0s
Spine01           swp4(leaf04)                 default         65020
65014      2/-/-        1d:4h:54m:59s
```

Manage Network Events

The NetQ Notifier manages the events that occur for the devices and components, protocols and services that it receives from the NetQ Agents. The Notifier enables you to capture and filter events that occur to manage the behavior of your network. This is especially useful when an interface or routing protocol goes down and you want to get them back up and running as quickly as possible, preferably before anyone notices or complains. You can improve resolution time significantly by creating filters that focus on topics appropriate for a particular group of users. You can easily create filters around events related to BGP, LNV, and MLAG session states, interfaces, links, NTP and other services, fans, power supplies, and physical sensor measurements.

For example, for operators responsible for routing, you can create an integration with a notification application that notifies them of routing issues as they occur. This is an example of a Slack message received on a *netq-notifier* channel indicating that the BGP session on switch *leaf04* interfa ce *swp2* has gone down.


netq-notifier APP 1:34 PM
⚠️ filter#ALL: @demo: BGP: leaf04 swp2: session state changed from established to failed
From netq-notifier running on oob-mgmt-server | Aug 14th

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 switch or server. This timestamp is based on the switch or server time where the NetQ Agent is running, and is pushed in UTC format. It is important to ensure that all devices are NTP synchronized to prevent events from being displayed out of order or not displayed at all 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, 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 compensates for this 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 that have passed since the epoch time (January 1, 1970 at

00:00:00 GMT). Non-JSON output displays how far 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 example shows timestamps with different time granularity.

```
cumulus@switch:~$ netq show agents
Matching agents records:
Hostname          Status           NTP Sync Version
Sys Uptime                Agent Uptime              Reinitialize Time
Last Changed
----------------- ---------------- --------
------------------------------------ -------------------------
------------------------- --------------------------
-------------------------
act-omp03-fc101   Fresh            yes      1.4.0-cl3u10~1537996283.40268de
16d:20h:54m:29s           5d:20h:58m:47s            5d:20h:58m:47s
14.83580s
act-omp03-lc101   Fresh            yes      1.4.0-cl3u10~1537996283.40268de
16d:20h:54m:25s           4d:2h:0m:14s              4d:2h:0m:14s
21.673068s
act-omp03-lc202   Fresh            yes      1.4.0-cl3u10~1537996283.40268de
16d:20h:52m:15s           5d:20h:58m:39s            5d:20h:58m:39s
34.113479s
```

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

Additionally, if a NetQ Agent is restarted on a device, the timestamps for existing objects are not updated to reflect this new restart time. Their timestamps are preserved relative to the original start time of the Agent. A rare exception is if the device is rebooted between the time it takes the Agent being stopped and restarted; in this case, the time is once again relative to the start time of the Agent.

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
    }
}
```

Telemetry File Locations

The primary configuration file for all Cumulus NetQ tools, `netq.yml`, resides in `/etc/netq` by default.

Log files are stored in `/var/logs/cts` by default.

Refer to Investigate NetQ Issues for a complete listing of configuration files and logs for use in issue resolution.

Image and Provisioning Management Components

The NetQ Telemetry Server hosts the Image and Provisioning Management (IPM) components, including a DHCP server, ONIE installation manager, and ZTP provisioning manager. Each of the IPM components used to manage images and configuration are described here.

DHCP Server

The DHCP server uses the Dynamic Host Configuration Protocol to dynamically assign IP addresses to network devices and to provide a default path (HTTP URL) to ONIE images and ZTP scripts. You can choose to use the embedded server for all of your DHCP services or integrate with your own. For more detail about how DHCP works, refer to the RFC 2131 standard.

Network Installation Manager

The Network Install manager uses ONIE (Open Network Install Environment) to store and distribute network operating system (NOS) images. ONIE combines a boot loader and a small operating system for network switches that provides an environment for automated provisioning. ONIE utilizes the CPU complex of the switch, including the CPU SoC, DRAM, boot flash, and mass storage, and creates an environment for installation. On initial boot of a server, ONIE configures the network management interface and locates and executes the Cumulus Networks OS installation program. For more detail about the ONIE standard, refer to ONIE.

Provisioning Manager

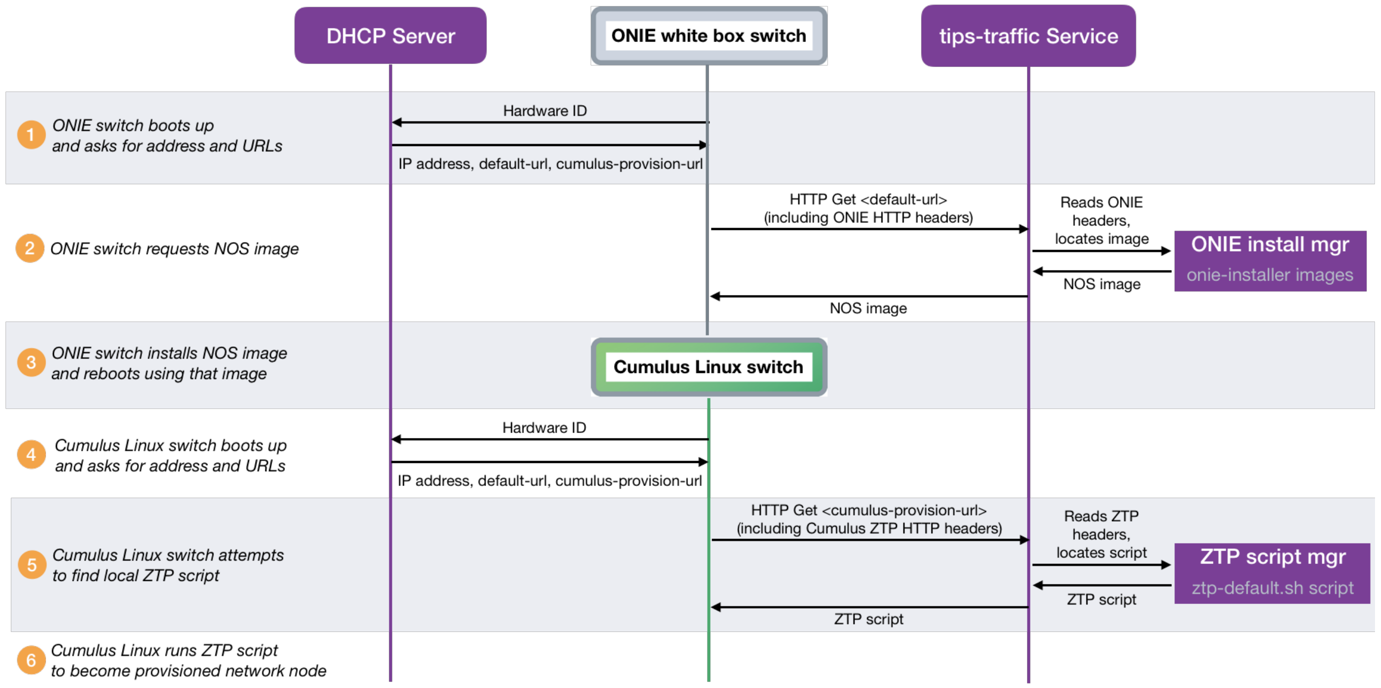
The Provisioning manager uses ZTP (Zero Touch Provisioning) to store and distribute provisioning scripts. ZTP provides a provisioning framework that allows for a one-time, user-provided script to be executed. On the first boot of a Cumulus Linux switch, IPM uses a default script, `ztp-default.sh`, provided through the DHCP server, to perform provisioning tasks, such as license installation, connectivity testing, and specifying a hostname. You can create your own ZTP script to be used instead by storing it in a designated location. For more detail about how ZTP works and tips for writing your own scripts, refer to ZTP.

Command Line Interface

IPM provides a command line interface, TIPCTL, for NOS image management and ZTP script management. While initial boot and configuration is handled automatically, IPM enables the network administrators and operators to manage network switch software over the entire lifecycle of these devices.

Image and Provisioning Management Operation

IPM installs and provisions bare metal servers to quickly transform them into Cumulus Linux switches. On the initial boot of a white box switch, IPM automatically loads the switch with the Cumulus Linux OS and provisions it with the network information required to make it a functional network node, including an IP address. This figure shows the interactions of the various IPM components with the switch hardware during an initial bring up. The *DHCP server* listens to port 67 for DHCP client messages and sends messages to client port 68. The *tips-traffic service* uses port 9300 on the Telemetry Server for requests. Objects shown in purple are components of IPM.


DHCP Server
ONIE white box switch
tips-traffic Service
1 ONIE switch boots up and asks for address and URLs
Hardware ID
IP address, default-url, cumulus-provision-url
2 ONIE switch requests NOS image
HTTP Get <default-url> (including ONIE HTTP headers)
Reads ONIE headers, locates image
ONIE install mgr
onie-installer images
NOS image
NOS image
3 ONIE switch installs NOS image and reboots using that image
Cumulus Linux switch
4 Cumulus Linux switch boots up and asks for address and URLs
Hardware ID
IP address, default-url, cumulus-provision-url
5 Cumulus Linux switch attempts to find local ZTP script
HTTP Get <cumulus-provision-url> (including Cumulus ZTP HTTP headers)
Reads ZTP headers, locates script
ZTP script mgr
ztp-default.sh script
ZTP script
ZTP script
6 Cumulus Linux runs ZTP script to become provisioned network node

After the switch is configured, subsequent configuration and upgrades are performed using the IPM CLI.

IPM File Locations

Log files are stored in `/var/logs/tips` by default.

Network Operating System (NOS) images are stored in `/var/tips/www/onie/images/` by default.

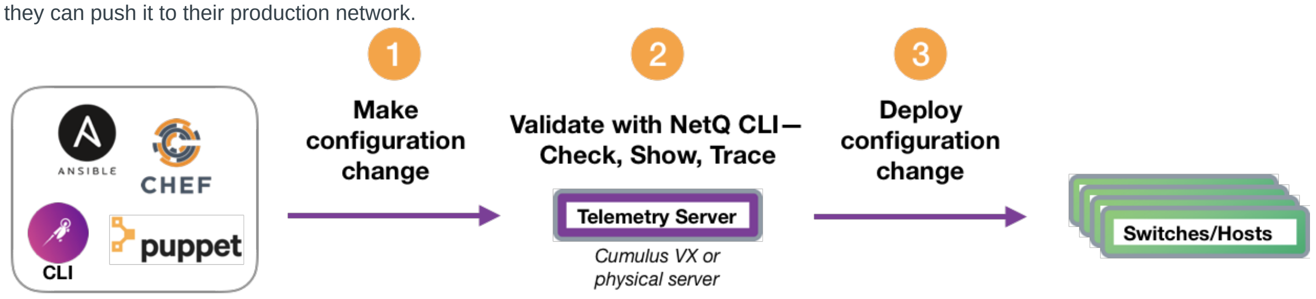
Provisioning scripts are stored in `/var/tips/www/ztp/scripts/` by default.

NetQ Virtual

You can try out NetQ in two different virtual environments. These environments enable you to try out NetQ on your own, or to test/validate updates to your network before deploying them into production. They are:

- Cumulus in the Cloud, which is a virtual data center that includes the NetQ telemetry server for monitoring your Cumulus in the Cloud instance.
- The Cumulus Networks GitHub site has a virtual NetQ demo environment. 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.

For example, a network administrator makes a configuration change that they are considering making on their production network. By running this on either of these environments, the configuration can be tested and reviewed for unforeseen impacts. If the configuration change looks good, they can push it to their production network.


ANSIBLE
CHEF
CLI
puppet
1
Make configuration change
2
Validate with NetQ CLI— Check, Show, Trace
Telemetry Server
Cumulus VX or physical server
3
Deploy configuration change
Switches/Hosts

Available Documentation

A number of user documents are available. These documents provide the information you need to proactively monitor your Linux-based network fabric using Cumulus NetQ. They assume that you have already installed Cumulus Linux and NetQ. You may start anywhere in the documentation or read it from start to finish depending on your role and familiarity with the NetQ software and Linux networking.

This Cumulus NetQ Primer is available as a PDF for offline viewing.

The following NetQ documents are available:

- Cumulus NetQ Deployment Guide
- Cumulus NetQ Telemetry User Guide
- Cumulus NetQ Image and Provisioning Management User Guide
- Cumulus NetQ Release Notes
- Cumulus NetQ Data Sheet