53-1003887-01
17 August 2015

Brocade SDN Controller

User Guide

Supporting Brocade SDN Controller v 2.0.1

BROCADE

Contents

Preface

Document conventions

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

Text formatting conventions

Text formatting conventions such as boldface, italic, or Courier font may be used in the flow of the text to highlight specific words or phrases.

Format	Description
bold text	Identifies command names
	Identifies keywords and operands
	Identifies the names of user-manipulated GUI elements
	Identifies text to enter at the GUI
italic text	Identifies emphasis
	Identifies variables
	Identifies document titles
`Courier font`	Identifies CLI output
	Identifies command syntax examples

Command syntax conventions

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

Convention	Description
bold text	Identifies command names, keywords, and command options.
italic text	Identifies a variable.
value	In Fibre Channel products, a fixed value provided as input to a command option is printed in plain text, for example, **--show** WWN.

Convention	Description
[ ]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ **x** | **y** | **z** }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options. In Fibre Channel products, square brackets may be used instead for this purpose.
x | **y**	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, for example, passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, *member*[*member*...].
\	Indicates a “soft” line break in command examples. If a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

Notes, cautions, and warnings

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

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

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

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

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

Brocade resources

Visit the Brocade website to locate related documentation for your product and additional Brocade resources.

You can download additional publications supporting your product at www.brocade.com. Select the Brocade Products tab to locate your product, then click the Brocade product name or image to open the individual product page. The user manuals are available in the resources module at the bottom of the page under the Documentation category.

To get up-to-the-minute information on Brocade products and resources, go to MyBrocade. You can register at no cost to obtain a user ID and password.

Release notes are available on MyBrocade under Product Downloads.

White papers, online demonstrations, and data sheets are available through the Brocade website.

Contacting Brocade Technical Support

As a Brocade customer, you can contact Brocade Technical Support 24x7 online, by telephone, or by e-mail. Brocade OEM customers contact their OEM/Solutions provider.

Brocade customers

For product support information and the latest information on contacting the Technical Assistance Center, go to http://www.brocade.com/services-support/index.html.

If you have purchased Brocade product support directly from Brocade, use one of the following methods to contact the Brocade Technical Assistance Center 24x7.

Online	Telephone	E-mail
Preferred method of contact for non-urgent issues: • My Cases through MyBrocade • Software downloads and licensing tools • Knowledge Base	Required for Sev 1-Critical and Sev 2-High issues: • Continental US: 1-800-752-8061 • Europe, Middle East, Africa, and Asia Pacific: +800-AT FIBREE (+800 28 34 27 33) • For areas unable to access toll free number: +1-408-333-6061 • Toll-free numbers are available in many countries.	support@brocade.com Please include: • Problem summary • Serial number • Installation details • Environment description

Brocade OEM customers

If you have purchased Brocade product support from a Brocade OEM/Solution Provider, contact your OEM/Solution Provider for all of your product support needs.

- OEM/Solution Providers are trained and certified by Brocade to support Brocade® products.
- Brocade provides backline support for issues that cannot be resolved by the OEM/Solution Provider.

- Brocade Supplemental Support augments your existing OEM support contract, providing direct access to Brocade expertise. For more information, contact Brocade or your OEM.
- For questions regarding service levels and response times, contact your OEM/Solution Provider.

Document feedback

To send feedback and report errors in the documentation you can use the feedback form posted with the document or you can e-mail the documentation team.

Quality is our first concern at Brocade and we have made every effort to ensure the accuracy and completeness of this document. However, if you find an error or an omission, or you think that a topic needs further development, we want to hear from you. You can provide feedback in two ways:

- Through the online feedback form in the HTML documents posted on www.brocade.com.
- By sending your feedback to documentation@brocade.com.

Provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.

About This Document

Intended audience

This guide is intended for experienced system and network administrators. You must have a basic understanding of Linux operating systems and hypervisor environments.

Product applicability

This guide applies to the Brocade SDN Controller v2.0.1.

What is new in this document

Brocade SDN Controller 2.0.1 is the current release.

Supported platforms

Brocade SDN Controller is supported on the Ubuntu 14.04 operating system.

Supported browsers

The Brocade SDN Controller supports the following browsers:

- Mozilla Firefox
- Google Chrome

Controller publications

The following publications support the Brocade SDN Controller:

- *Brocade SDN Controller Compatibility Matrix with Apps and Extensions*
- *Brocade SDN Controller Quick Start Guide*
- *Brocade SDN Controller Software Installation Guide*
- *Brocade SDN Controller User Guide*
- *Brocade SDN Controller OpenStack Integration Guide*
- *Brocade SDN Controller Release Notes*
- *Brocade Flow Manager User Guide*
- *Brocade Flow Manager Release Notes*

Brocade SDN Controller Overview

About the controller

Brocade SDN Controller is a software-defined networking (SDN) controller that is based on the OpenDaylight project.

The controller consists of packages that form a platform on which you can install or build apps. Packages that provide a common functionality and are essential for running the controller are part of the base controller. Packages that are optional are extensions that can be installed to enable additional functionality. The controller is built on a modular design that allows apps and extensions to be updated while maintaining a stable platform, which helps to improve the delivery time of apps and extensions and reduce risk to users.

Unless mentioned otherwise in this guide, the controller refers to Brocade SDN Controller.

About the base controller

The base controller is a platform on which various apps and extensions can be installed or built to provide additional functionality. Apps and extensions sometimes depend on a specific version of the controller.

The following list presents the current components of the base controller:

- bvc-*version*.zip—Contains the base controller packages
- bvc-dependencies-*version*.zip—Contains the dependencies packages for the base controller
- bvc-core-ux-*version*.zip—Contains the user interface packages for the base controller

Features of the base controller

The base controller consists of the controller GUI, NETCONF, RESTCONF, MD-SAL, and clustering.

The following list presents the current components of the base controller:

- *Controller GUI*—The controller GUI interacts with the controller by using REST APIs and displays information such as network topology, nodes, and so on.
- *NETCONF*—The Network Configuration Protocol (NETCONF) is a network management protocol. The controller acts as both a NETCONF server and client. The NETCONF client connects to external NETCONF-enabled devices and manages them through a NETCONF connector. The NETCONF server presents the controller as a NETCONF-enabled device so that the controller can be managed and configured by using a NETCONF client.
- *RESTCONF*—RESTCONF describes how to map a YANG specification to a REST interface. RESTCONF provides a simplified interface that follows REST-like principles and is compatible with a resource-oriented device abstraction. Brocade SDN Controller implements a memory data store to store data that is related to the YANG models. Using the RESTCONF protocol, you can perform a CRUD (create, read, update or delete) operation on the data that is stored in the memory data store.
- *MD-SAL*—The Model-driven Service Abstraction Layer (MD-SAL) architecture unifies both northbound and southbound APIs and the data structures that are used in various services and components of the controller.
- *Clustering*—The controller supports the clustering feature that facilitates high availability (HA) in your network. Clustering allows one or more controller instances to share a common data store, which means that any instance can service a request. Brocade supports a three-node cluster.

About the extensions

The Brocade SDN extensions add functionality to the controller and change the way you can configure the controller or other extensions. The extensions are optional but may be required for other extensions or apps to work as expected.

The following list presents the currently available extensions:

- bvc-ext-openflow-packaging-*version*.zip—Contains the OpenFlow packages
- bvc-ext-bgp-pcep-packaging-*version*.zip—Contains the Border Gateway Protocol (BGP) and Path Computation Element Protocol (PCEP) packages
- bvc-ext-hosttracker-noflood-packaging-*version*.zip—Contains the Host Tracker No Flood Mode packages.
- bvc-ext-ovsdb-packaging-*version*.zip—Contains the Open vSwitch Database Management protocol (OVSDB) packages
- bvc-ext-l2switch-packaging-*version*.zip—Contains the Layer 2 (L2) switch packages
- bvc-ext-utilities-packaging-*version*.zip—Contains the controller utilities packages

When you run the controller `./installbsc` installation script command with the `-y` argument, all the extensions are installed by default. If you do not use the `-y` argument, you are prompted to install each extension.

Features of the extensions

The extensions that are currently shipped with the controller include OpenFlow, BGP PCEP, Layer 2 switch, Host Tracker No Flood Mode, OVSDB, and the utilities.

The following list presents the current extensions shipped with the controller.

- *OpenFlow*—The controller supports OpenFlow. OpenFlow components consist of the controller, an OpenFlow-enabled switch, and the OpenFlow protocol and enable you to run experimental flow protocols on your network.
- *BGP PCEP*—The controller supports BGP PCEP. The extension allows you to dynamically visualize links and configure tunnels in the controller through RESTCONF calls.
- *Layer 2 switch*—The Layer 2 (L2) switch performs switching operations with the controller. While the controller has a logical, centralized view of the network, the Layer 2 switch learns MAC addresses and builds a flow table to forward frames.
- *Host Tracker No Flood Mode*—Host Tracker is a component of the OpenDaylight Layer 2 (L2) switch application and is an extension that is used to disable flood mode on Host Tracker. Host Tracker tracks the locations of the hosts by using the MAC addresses of the hosts as the primary identifier in the network and then adds them to the topology. Host Tracker creates a node in the topology to represent a host and attaches it to the appropriate switches.
- *OVSDB*—The Open vSwitch Database Management Protocol (OVSDB) plug-in integration is a project for OpenDaylight that implements the Open vSwitch Database RFC 7047 management protocol, allowing the southbound configuration of vSwitches.
- *Utilities*—Utilities help you run controller operations. The current utilities for the controller include the Backup Manager script, Change Password script, Cluster Monitor, and Setup HTTPS script. You can also run a diagnostic script and send the results of running the script to Brocade support in the event of troubleshooting.

About the apps

The controller provides additional apps as add-on packages. You can access the apps by using the controller user interface.

Brocade Topology Manager is installed by default when you install the controller user interface. Most apps are not installed as part of the controller installation script. Some apps may require additional licensing rights. The following list provides some of the available apps:

- Brocade Topology Manager
- Brocade Flow Manager

For more information about the apps, refer to the specific app User Guide.

About Brocade Topology Manager

The Brocade Topology Manager app displays a network graph of the topology, hosts ports and flows information. It also displays statistics of the selected switch.

The controller determines the topology among all the network switches that are connected to the controller. The controller user interface then uses the topology details for graphical representation.

About Brocade Flow Manager

The Brocade Flow Manager app enables you to manage flows of an OpenFlow version 1.3 switch in the network.

You can perform the following operations by using the Brocade Flow Manager app:

- Manage OpenFlow switches
- Manage network paths

- View the network topology
- Verify flow statistics

You can access the Brocade Flow Manager app by using the Brocade SDN Controller user interface. To access the Flow Manager app, you must install the app separately.

Getting started with the controller

You must ensure that you perform all the prerequisite tasks before you install and configure the controller.

Here is a quick list of steps to enable you to get started with the controller. Refer to *Brocade SDN Controller Software Installation Guide* for more information.

1. Download and install the controller.
2. Verify that the controller installation is correct.
3. Log in to the controller GUI.
4. Refer to the following sections for controller features and applications.

Brocade SDN Controller Platform

Changing the default controller password

The Brocade SDN Controller is shipped with the following default profiles:

- Admin profile with the username of admin and the password of admin .
- User profile with the username of user and the password of user.

To change the default controller password, perform these steps :

1. Go to the bin directory by entering the following command: `cd /opt/bvc/bin/`
2. Run the change password script by entering the following command: `./change_password -u <username>`

   `<username>` specifies the username of the profile for which you want to change the password.
3. If you are changing the password for multiple users, enter **N** when prompted to start the controller.
4. Restart the controller (optional).

Default passwords are stored in the sqlite3 database. The sqlite database must be initialized by the controller before the change password utility is run.

NOTE
The database can be accessed only by the Linux administrator who installed the controller.

Brocade SDN Controller User Interface

Brocade SDN Controller user interface overview

The Brocade SDN Controller user interface interacts with the controller by using REST APIs and displays information such as network topology, nodes, and so on. Apps are installed separately. Apps that you have installed are listed in the app board.

Signing in to the Brocade SDN Controller user interface

This section provides information about signing in to the Brocade SDN Controller user interface.

To sign in to the controller user interface, perform the following steps.

1. Open a supported browser.
2. Enter the login URL
   - If you are using HTTP for the controller and the user interface use the following URL: `http://<controller-ip>:9001`.
   - If you are using HTTPS for the controller and the user interface, use the following url `https://<controller-ip>:8443`

   NOTE
   If you login to `https://<controller_IP>:9445` and receive an error message that prevents you from connecting to the controller, open `https://<controller_IP>:8443/apidoc/explorer/index.html` in a new tab. Enter your credentials. If you now open another tab and login `https://<controller_IP>:9445`, your login is successful.

   The <*controller-ip*> variable is the IP address of the computer in which the controller is installed. A login window is displayed.
3. Sign in as `admin`.
4. Click **Sign In**.

If you have signed in successfully , you will see the App Board page. If the sign in fails, an appropriate error message is displayed.

Navigating the Brocade SDN Controller user interface

The following table lists the features that are available from the controller user interface.

Click any of the following menu items.

TABLE 1 User Interface navigation

Menu	User Interface element	Description
Primary Toolbar		Consists of the following icons to navigate the app.
	About	Displays the user interface version, controller IP address, and copyright information
	Online Documentation	Navigates to the Brocade SDN Controller online documentation.
	Contact Support	Navigates to the Brocade support page.
	Home	Navigates to the app board listing the apps that are installed.
	Profile*username*	Displays information about the signed-in user, the controller, and the time since when the user is using the app.
	Logout*username*	Logs out the user from the user interface.
	Apps Menu	Displays a navigation bar on the right side of the pane, that lists the apps that are installed on the controller. To exit from the app menu from the side navigation bar, click anywhere away from side navigation bar.
App Board		Displays the apps.
	Start Tour	Highlights the icons on the user interface and provides a brief description of each icon. Click any of the following menu items to use the Start Tour feature: • Click **Skip** to revert to the tools menu. • Click **Back** to see the previous help. • Click **Next** to see the help for the next icon.
	Show Icon View	Displays the apps as cards. Click the image or on the title to view the app. The vertical menu icon on the app card displays the description of the app.
	Show List View	Displays the apps in a list format. Select the app name to view the app. **NOTE** If you are on the same view when you click the **Show Icon View** or the **Show List view**, you do not see any changes on user interface.
	Search or filter apps	To filter the apps, type a few characters in the search box. The apps are filtered by matching any of the following search criteria: • Name of the app • Description of the app • Tags of the app

Brocade Topology Manager

About Brocade Topology Manager

The Brocade Topology Manager app displays a network graph of the topology, hosts ports and flows information. It also displays statistics of the selected switch.

The controller determines the topology among all the network switches that are connected to the controller. The controller user interface then uses the topology details for graphical representation.

Viewing network topology

To view the network topology, perform the following steps.

1. Log in to the Brocade SDN Controller.
   The controller user interface displays all the installed apps as cards.
2. Click the **Topology Manager** app card.
   You see the graphical representation of the host and switches connected by using links.
3. Rest the cursor over the **Host** icon to display a tool tip that displays host name and IP address.
4. Rest the cursor over the **Switch** to display a tool tip that displays IP address, host name, hardware/ software version and manufacturer name.
5. Click the **Host** or the **Switch** to select the network element and also highlight the connected links.
6. Click the link to highlight the link and the connected network elements. The name of the highlighted network elements will also be displayed as a label.
7. Hold and drag a network element to a desired position to pin the network element. The network element will become fixed and you can move the rest of the topology. To unpin, right-click the network element and select **Unpin**.
8. Click the **Zoom In** and **Zoom Out** magnifier buttons to reduce or enlarge the topology view.
9. Click the **Toggle Labels** button to show or hide network elements.
10. Click the **Logout *user*** icon to log out of the Topology Manager.

Using the Switches list

The **Switches** list in the left navigation bar displays the switch name and IP address in a collapsible list. Click each switch to view additional information about the switch.

To use the Switches list, perform the following steps.

1. Click **Switches** to collapse or expand the list. The **Switches** header also displays a count of switches.
2. Type a search criteria in the **Search Switches** field to filter the switches.
   The list will display only the switches containing the text entered.
3. Click on a switch item in the list to display the Switches form.
   This form displays additional information about the switch. The corresponding network element is highlighted in the graph. For more information about viewing the details of the switch, refer to Viewing switch details on page 21

Viewing switch details

The Switches list in the left navigation bar displays the switch name and IP address in a collapsible list.

The Switches list displays only the operational flows as read by the controller from the switches. To view the flows that are listed in the Switches list, perform the following steps.

1. Select **Switches** on the left navigation bar.
2. Click one of the following icons in the title bar:
   - Click **Port** icon to show or hide ports.
   - Click **Flows** icon to show or hide flows.
   - Click **Help** to see quick help regarding the information that is displayed. Click **Help** again to hide the help.

The following information is displayed in the Switches window:

- Switches—The header is highlighted and displays the following information:
  - Name of the switch
  - IP address
  - Host name
  - Hardware or software version
  - Manufacturer name
  - Port and flow count
- Ports—The Ports column displays the following information about the port of the selected switch:
  - Port name, port number, and MAC address
  - Number of packets that are transmitted from and received by the port.
  - Number of bytes of information that are transmitted from and received by the port.
  - **Bandwidth indicator** that displays the capacity utilization of the port. It is calculated based on total bytes through the port or capacity of the port in percentage.

  - **Warning** icon that is displayed if errors occur. When errors are detected in the port, the errors are displayed in a light red background, with corresponding value for each type of error. For example, Rx and Tx errors, drops, collisions, CRC errors, frame errors, and RoR errors.
  - **Red thumbs-down** icon indicates whether the port link is down.
- Flows— The Flows section displays the operational flows that are installed on the switch in a tabular format as follows:
  - Flow header displays number of active flows, number of packets looked up, and number of packets matched.
  - Flow name column displays the flow name and table Id.

    The table is sorted by priority from high to low.
  - Match column displays all the matches that are applied for the flow. These matches includes the Ethernet type, InPort, source MAC address, destination MAC address, VLAN ID, VLAN ID priority, source IP, destination IP, IP protocol, IP DSCP, source port, and destination port. Only values that match are displayed.
  - Action column displays the actions that are applied in the flow.
  - Packet/Byte column displays the count of packets and bytes for that flow.

Viewing network hosts

The Hosts list in the side navigation bar displays the MAC address and the IP address as a collapsible list.

Click the **Host** header to collapse or expand the list. The following table lists the fields that are displayed in the Hosts tab and its descriptions.

TABLE 2 Host information

Field	Description
Name	Host name
IP address	IP address of the host
MAC address	MAC address of the host

To minimize or expand the sidebar click the **>>** icon at the top right of the sidebar.

Setting the data refresh interval

The Flow Manager app uses web sockets to retrieve inventory and flow data from the controller. If the controller sends notifications at a very frequent rate, the user interface might throttle. By default, the data refresh interval is 120 seconds (that is, the websocket groups data updates after every 120 seconds). But for large networks with hundreds of switches, the data sent by controller to be processed could be so large and the refresh interval is so low that the amount of data received might affect the performance of the user interface. In order to avoid the data throttling, the app allows and updates the data only after a certain interval of time.

The app provides a way to configure the refresh interval to get a reasonable user experience with the trade-off being, how often the data is updated.

To change the data refresh interval, perform the following steps.

1. Edit the following file: /opt/bvc/ux/server/config-bvc.json.
2. Set the **inventoryNotificationRateLimitInSecs** key to a desired value.

RESTCONF Tools

RESTCONF tools overview

RESTCONF is a protocol that provides a programmatic interface over HTTP to access data that is defined in a YANG model and stored in data stores defined in the NETCONF protocol. The Brocade SDN Controller provides the implementation of a memory data store to store data that is related to the YANG models. Using the RESTCONF protocol, you can perform a CRUD (create, read, update or delete) operation on the data that is stored in the memory data store. For more information on the RESTCONF protocol, refer to http://tools.ietf.org/html/draft-bierman-netconf-restconf-02 and https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_Document_Review:RESTCONF.

To send RESTCONF requests you can use tools, such as Chrome Postman plug-in and cURL utility.

For more information on Postman plug-in, refer to Chrome Postman plug-in on page 24 and cURL utility, refer to cURL utility on page 26.

Chrome Postman plug-in

To use Postman chrome application:

1. Download and install Postman from the following location: https://chrome.google.com/webstore/detail/postman-rest-client/
2. Open Chrome, click **Applications**, and select **Postman**.
3. Select Headers:
   - Authorization: Select the **basic** tab, use admin both as the username and password and click **Refresh Header**.
   - Accept: application/xml
   - Content-type: application/xml
4. Enter a URL by using the format at: http://<controller-ip>:8181/URI.
5. Select one of the following REST methods:
   - GET
   - PUT
   - POST
   - DELETE
6. Click the **raw** tab, select the **XML** tab, and fill body data as required.
7. Click **Send**.

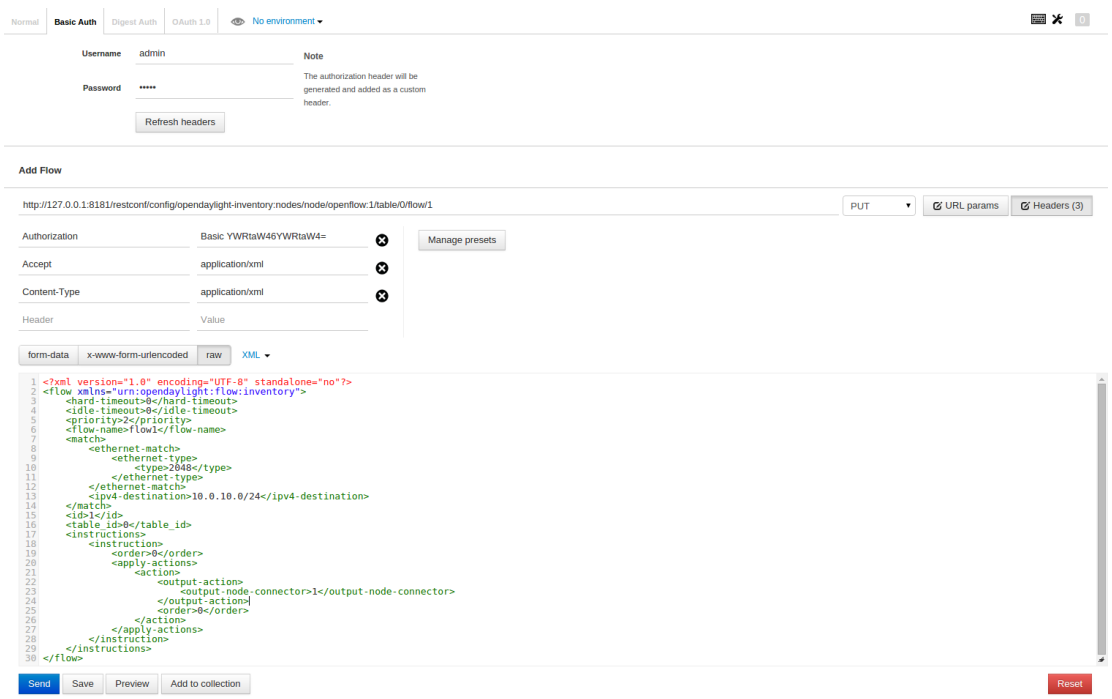
NOTE
You can import some of the examples into Postman by importing the following collection: https://www.getpostman.com/collections.

Example for installing a flow

To install a flow by using Postman, set the parameters displayed in the following example.

NOTE
Enter the username, password and click **Refresh headers**.

FIGURE 1 Install Flow

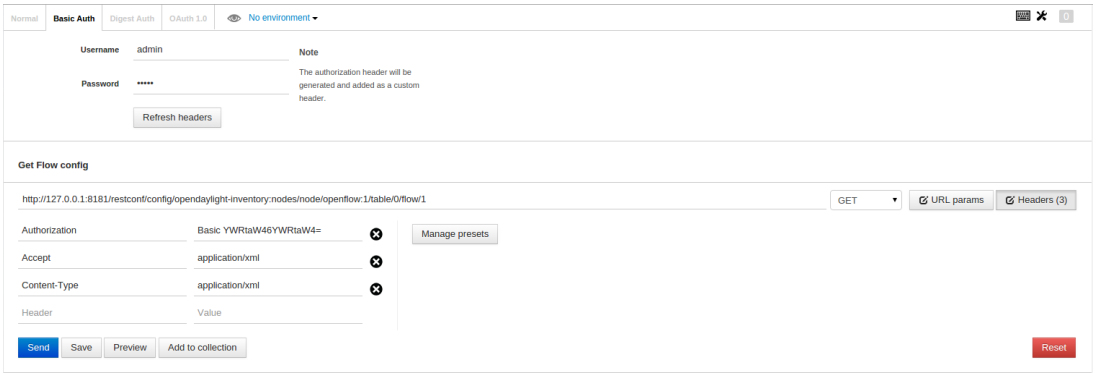


NOTE
Ensure that the table number and flow number in the URL and the XML body match.

Example of retrieving the installed flow

To retrieve the installed flow by using Postman, set the parameters that are displayed in the following example.

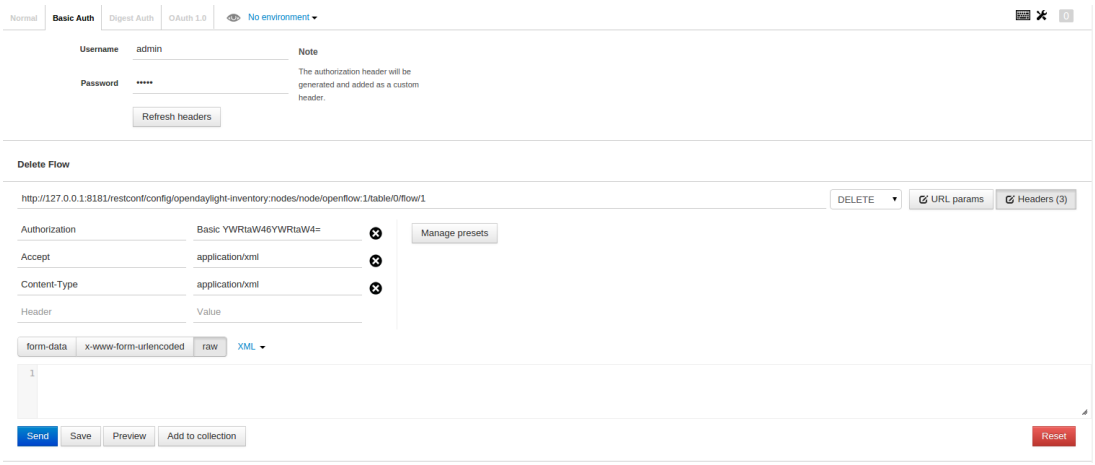
FIGURE 2 Retrieving Flow Configuration



Example of deleting the installed flow

To remove the installed flow by using Postman, set the parameters that are displayed in the following example.

FIGURE 3 Delete Flow



cURL utility

cURL is a command tool that can be used to make HTTP RESTCONF calls. The cURL tool can be used to perform RESTCONF-related CRUD operations.

A typical cURL request has the following format:

```
curl --user <username>:<password> -H <header 1> -H <header-2> -X <request-type> <url> -d '<request-body>'
```

Part of the required construct of the cURL requests follows:

- `--user <user-name>:<password>`: Specifies the username and password to use for server authentication.
- `-H Accept: <response-content-type>`: Specifies the content type that is expected in the response body for the request. For example: Accept: application/xml.
- `-H Content-type: <request-content-type>`: Specifies the content of the request body. For example: Content-type: <application/xml.
- `-X <request-type>`: Specifies the type of request you want to send to the given URI. For example: PUT, GET or DELETE.
- `-d <request-body>`: Specifies the request body (like Flow, Group, Meter, and so on). This is required for a PUT or POST request only.

Example of installing a flow

This example shows how the following cURL command sends the PUT request to the controller to install the flow.

cURL request:

- Headers:

  Username:password: admin: admin

  Accept: application/xml

  Content-type: application/xml
- Request-type: PUT
- URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Request body:

```
' <?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <priority>2</priority>
    <flow-name>flow1</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.10.0/24</ipv4-destination>
    </match>
    <id>1</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <output-action>
                        <output-node-connector>1</output-node-connector>
                    </output-action>
                    <order>0</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>'
```

cURL command:

```
curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type:
application/xml" -X PUT http://localhost:8181/restconf/config/opendaylight-
```

```
inventory:nodes/node/openflow:1/table/0/flow/1 -d '<?xml version="1.0"
encoding="UTF-8" standalone="no"?><flow
xmlns="urn:opendaylight:flow:inventory"><hard-timeout>0</hard-
timeout><idle-timeout>0</idle-timeout><priority>2</priority>flow-
name>flow1</flow-name><match><ethernet-match><ethernet-type><type>2048</
type></ethernet-type></ethernet-match><ipv4-destination>10.0.10.0/24</ipv4-
destination></match><id>1</id><table_id>0</
table_id><instructions><instruction><order>0</order><apply-
actions><action><output-action><output-node-connector>1</output-node-
connector></output-action><order>0</order></action></apply-actions></
instruction></instructions></flow>
```

cURL response: If the request fails, the error report with the reason for the failure is printed.

Example of retrieving the installed flow

The following cURL command fetches the existing flow that is stored in the controller data store.

cURL request:

Headers:

- <username>:<password>-: admin: admin

  Accept: application/xml

  Content-type: application/xml
- Request-type: GET
- URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- cURL command: `curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type: application/xml" -X GET http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1`
- cURL response: If the request is successful, the flow that is stored for the given URL is printed. Otherwise, the error with the reason for the failure is printed.

Example of deleting the installed flow

The following cURL command deletes the existing flow that is stored in the controller data store:

cURL request:

- Headers:

  <username>:<password>: admin: admin

  Accept: application/xml

  Content-type: application/xml
- Request-type: DELETE
- URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- cURL command: `curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type: application/xml" -X DELETE http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1`
- cURL response: If the request fails, the error report with the reason for the failure is printed.

OpenFlow

OpenFlow overview

OpenFlow is an open standard that enables you to run experimental flow protocols, thus, allowing easy deployment of new routing and switching functions in your network. The OpenFlow standard describes a communications protocol that allows any external entity, such as the Brocade SDN Controller, to access and configure the forwarding plane of a network device. The OpenFlow standard consists of a controller, an OpenFlow-enabled switch, and the OpenFlow communications protocol.

The switch and the controller communicate by exchanging OpenFlow protocol messages, which the controller uses to add, modify, and delete flows in the switch. By using OpenFlow, it is possible to control various aspects of the network, such as traffic forwarding, topology discovery, Quality of Service, and so on.

For more information about OpenFlow, refer to the Open Networking Foundation website at: https://www.opennetworking.org/component/content/article/42-sdn-resources/2046-technical-resources.

Deploying the controller in an OpenFlow scenario

This section describes how the controller can be used with Mininet to create simulated networks.

Follow the instructions in *Brocade SDN Controller Software Installation Guide* and install the controller.

The following steps create an OpenFlow scenario to use the controller.

1. Download Mininet. Mininet downloads are available at: http://mininet.org. The OVS version must be 2.1 or earlier.
2. Install Mininet. Instructions for installation are available at: http://mininet.org.
3. Create a virtual network consisting of two switches in the Mininet VM. On a Linux console, enter the following command:
   ```
   sudo mn --controller=remote,ip=<controller-ip>,port=6633 --topo linear,2
   ```
4. Generate traffic using Mininet by entering the following command in the Mininet shell:
   ```
   pingall
   ```

You can use the GUI, and optionally, the read-only RESTCONF.

On the controller GUI, view the icons for switches, connections, and hosts.

OpenFlow topology

The controller provides a centralized logical view of the network.

The controller uses the LLDP messages to discover the topology of the connected OpenFlow switches. The controller uses LLDP messages to discover links between the OpenFlow switches. The topology manager stores and manages the information (nodes and links) in the controller data stores. The other components, such as the LLDP speaker, topology LLDP discovery, and Host Tracker help to generate the topology database.

The following sections describe the components that are involved in the OpenFlow topology discovery.

Link Layer Discovery Protocol speaker

The responsibilities of the Link Layer Discovery Protocol (LLDP) speaker follow:

- Periodically send LLDP packets to all the node connectors of all the switches that are connected.
- Monitor status events for a node connector. If the status of a node connector for the connected switch changes from up to down, the LDDP speaker does not send packets out to that node connector. If the status changes from down to up, the LDDP speaker sends packets to that node connector.

Link Layer Discovery Protocol discovery topology

The responsibilities of topology LLDP discovery follow:

- Monitor the LLDP packets that are sent by a switch to the controller.
- Determine the details, such as the source node, source node connector, destination node, and destination node connector, from the received LLDP packets.
- Notify the topology manager of a new link-discovery event.
- Periodically check for an expired link and notify the topology manager. A link expires when it does not receive an update from the switch for the three LLDP speaker cycles.

Host Tracker

A main responsibility of Host Tracker is to determine hosts that are connected to the switch node connectors and update the topology information. For more information on the responsibilities of Host Tracker, refer to Host Tracker on page 53.

Topology manager

The responsibilities of the topology manager follow:

- Monitor the events that are published by the previously mentioned modules and store or update the controller topology details in the data store.
- Monitor events such as node add, node remove, node connector up, node connector down, link discovered. The manager also removes links and cleans up the topology data that is stored in the operational data store.

Retrieving topology details by using the controller user interface

You can retrieve the topology data from the controller by using the user interface. For more information on the user interface, refer to Brocade SDN Controller user interface..

Retrieving topology details by using RESTCONF

You can retrieve the topology data from the controller by sending a RESTCONF request. The controller fetches the topology data from the operational data store and returns it in response to the RESTCONF request. For more information on the usage of tools, refer to RESTCONF tools overview on page 24.

The details of the RESTCONF request that has to be sent to the controller are as follows:

- Headers:

  Content-type: application/xml

  Accept: application/xml

  Authentication: admin:admin
- Method: GET
- URL: http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
- Output:


```
{
    "network-topology": {
        "topology": [
            {
                "topology-id": "flow:1",
                "link": [
                    {
                        "link-id": "host:b6:da:d5:85:cf:a8:openflow:2:1/openflow:2:1",
                        "destination": {
                            "dest-node": "openflow:2",
                            "dest-tp": "openflow:2:1"
                        },
                        "source": {
                            "source-tp": "host:b6:da:d5:85:cf:a8:openflow:2:1",
                            "source-node": "host:b6:da:d5:85:cf:a8"
                        }
                    },
{
                        "link-id": "openflow:2:3",
                        "destination": {
                            "dest-node": "openflow:1",
                            "dest-tp": "openflow:1:1"
                        },
                        "source": {
                            "source-tp": "openflow:2:3",
                            "source-node": "openflow:2"
                        }
                    },

                                                .....

                                                .....

                    {
                        "link-id": "openflow:3:3",
                        "destination": {
                            "dest-node": "openflow:1",
                            "dest-tp": "openflow:1:2"
                        },
                        "source": {
                            "source-tp": "openflow:3:3",
                            "source-node": "openflow:3"
                        }
                    },
                    {
                        "link-id": "host:0a:ac:07:d9:31:5d:openflow:3:1/openflow:3:1",
```

```
        "destination": {
            "dest-node": "openflow:3",
            "dest-tp": "openflow:3:1"
        },
        "source": {
            "source-tp": "host:0a:ac:07:d9:31:5d:openflow:3:1",
            "source-node": "host:0a:ac:07:d9:31:5d"
        }
    },
```

OpenFlow programming overview

The controller provides programming interfaces that can be used to program the switch. These programming interfaces interact with an OpenFlow southbound plug-in that uses OpenFlow modification messages to program the switch.

The controller mainly provides the following programming interfaces for you to program the switch:

- Java APIs defined by the controller: You can develop your own application in the controller and use the Java APIs to program the switch. The description of this option is beyond the scope of this user guide.
- RESTCONF interface: You can use any of the RESTCONF tools to send a request to the controller to program a specific switch. When the RESTCONF request is received, the controller internally uses the Java APIs to program the switch.

For more examples of flow programming by using RESTCONF, refer to ODL OpenFlow plug-in examples at: https://wiki.opendaylight.org/view/Editing_OpenDaylight_OpenFlow_Plugin:End_to_End_Flows:Example_Flows.

For more information on an OpenFlow YANG model, refer to: https://git.opendaylight.org/gerrit/gitweb?p=controller.git;f=opendaylight/md-sal/model/model-flow-base/src/main/yang/opendaylight-flow-types.yang;a=blob.

The following sections contain examples of programming a switch by using the RESTCONF interface. For more information on RESTCONF tools, refer to RESTCONF tools overview on page 24.

Example of L2 flow programming by using RESTCONF

This example provides the details to program a flow that matches Ethernet packets with source MAC address 00:00:00:00:23:ae and destination MAC address 20:14:29:01:19:61 and sends them to port 2.

To push the flow by using RESTCONF, enter the following parameters in your RESTCONF client tool:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL: http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/flow/{flow-id}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
```

```
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>2</output-node-connector>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>0</table_id>
    <id>1</id>
    <cookie_mask>255</cookie_mask>
    <match>
        <ethernet-match>
            <ethernet-destination>
                <address>20:14:29:01:19:61</address>
            </ethernet-destination>
            <ethernet-source>
                <address>00:00:00:00:23:ae</address>
            </ethernet-source>
        </ethernet-match>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>1</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>L2-Flow</flow-name>
    <priority>2</priority>
    <barrier>false</barrier>
</flow>
```

NOTE
To use a different flow ID or table ID, ensure that the URL and the request body are synchronized.

To verify that the flow has been correctly programmed in the switch, issue the RESTCONF request as provided in Example of individual flow statistics on page 39.

When there is more than one action in the flow, such as `output-action`, `set-field`, and so on, list the actions individually under `apply-action` even when they are of the same type. For example, a change in multiple fields should enclose each change in its `set-field` action.

Example of L3 flow programming by using RESTCONF

This example provides the details for programming a flow that matches IP packets (ethertype 0x800) with the destination address within the 10.0.10.0/24 subnet and sends them to port 1.

To push the flow by using RESTCONF, enter the following parameters in your RESTCONF client tool:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL: http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/flow/{flow-id}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
```

```
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <priority>2</priority>
    <flow-name>flow1</flow-name>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
        <ipv4-destination>10.0.10.0/24</ipv4-destination>
    </match>
    <id>1</id>
    <table_id>0</table_id>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <output-action>
                        <output-node-connector>1</output-node-connector>
                    </output-action>
                    <order>0</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

NOTE
To use a different flow ID or table ID, ensure that the URL and the request body are synchronized.

To verify that the flow in the switch is programmedappropriately, issue the RESTCONF request as provided in Example of individual flow statistics on page 39.

Flows with IPv4 address or IPv6 address match-clauses must include a CIDR network mask.

Example of a group definition by using RESTCONF

To push a group definition by using RESTCONF, use the following parameters in your RESTCONF client tool:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL: http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/group/{group-id}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<group xmlns="urn:opendaylight:flow:inventory">
    <group-type>group-select</group-type>
    <buckets>
        <bucket>
         <weight>1</weight>
            <action>
                <output-action>
                    <output-node-connector>1</output-node-connector>
                </output-action>
```

```
                <order>1</order>
            </action>
            <bucket-id>1</bucket-id>
        </bucket>
        <bucket>
          <weight>1</weight>
            <action>
                <output-action>
                    <output-node-connector>2</output-node-connector>
                </output-action>
                <order>1</order>
            </action>
            <bucket-id>2</bucket-id>
        </bucket>
    </buckets>
    <barrier>false</barrier>
                <group-name>SelectGroup</group-name>
    <group-id>1</group-id>
</group>
```

NOTE
To use a different group ID, ensure that the URL and the request body are synchronized.

For example: To change the group ID to 20, update the following:

- URL to http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group/20.
- Match <group-id>20</group-id> in the request body.
- Provide the IP address of the machine on which the controller is running.
- To verify that the group has been correctly programmed in the switch, issue the RESTONF request as provided in Example of group description and group statistics on page 41.

Example of a flow that contains a group instruction

To verify the flow has been correctly programmed, issue the RESTONF request in the individual flow statistics:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL: http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/flow/{flow-d}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <group-action>
                        <group-id>1</group-id>
                    </group-action>
                    <order>1</order>
                </action>
```

```
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>0</table_id>
    <id>1</id>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
    </match>
    <flow-name>FlowWithGroupInstruction</flow-name>
    <priority>2</priority>
</flow>
```

NOTE
To use a different flow ID or table ID, ensure that the URL and the request body are synchronized.

To verify that the flow has been correctly programmed in the switch, issue the RESTONF request as provided in Example of individual flow statistics on page 39.

Example of a meter definition by using RESTCONF

To push a meter definition by using RESTCONF, use the following parameters in your RESTCONF client tool:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL: http//<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/meter/{meter-id}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<meter xmlns="urn:opendaylight:flow:inventory">
    <flags>meter-kbps</flags>
    <meter-band-headers>
        <meter-band-header>
            <band-id>0</band-id>
            <dscp-remark-rate>50</dscp-remark-rate>
            <dscp-remark-burst-size>500</dscp-remark-burst-size>
            <meter-band-types>
                <flags>ofpmbt-dscp-remark</flags>
            </meter-band-types>
            <perc_level>1</perc_level>
        </meter-band-header>
    </meter-band-headers>
    <meter-id>1</meter-id>
    <meter-name>DscpRemarkMeter</meter-name>
</meter>
```

NOTE
To use a different meter ID, ensure that the URL and the request body are synchronized.

For example: To change the meter ID to 20, update the following:

- URL to http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/20.
- Match <meter-id>20</meter-id> in the request body.
- Provide the IP address of the machine on which the controller is running.
- To verify that the meter has been correctly programmed in the switch, issue the RESTONF request as provided in Example of meter configuration and meter statistics on page 42.

Example of a flow that contains a meter instruction

To push a flow that contains a meter instruction, enter the following parameters in your RESTCONF client tool:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin:admin
- URL:

  http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/{node-id}/table/

  {table-id}/flow/{flow-id}
- Example URL: http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/

  table/0/flow/1
- Method: PUT
- Request body:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                   <output-action>
                        <output-node-connector>1</output-node-connector>
                    </output-action>
                    <order>1</order>
                </action>
            </apply-actions>
        </instruction>
        <instruction>
            <order>1</order>
            <meter>
              <meter-id>2</meter-id>
            </meter>
        </instruction>
    </instructions>
    <table_id>0</table_id>
    <id>2</id>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
    </match>
    <flow-name>FlowWithMeterInstruction</flow-name>
    <priority>2</priority>
</flow>
```

NOTE
To use a different flow ID or table ID, ensure that the URL and the request body are synchronized.

To verify that the flow has been correctly programmed in the switch, issue the RESTONF request as provided in Example of individual flow statistics on page 39.

OpenFlow path programming

Two hosts that are attached to an OpenFlow network can be connected by programming the corresponding flows in all the switches involved in the path. The flow programming can be manual, as described in the previous chapter, or automatic by using one of the path programming applications that are supported in the controller.

The following sections describes the usage of the applications that are supported in the controller.

ARP handler in Host Tracker—flood flows

The ARP handler in Host Tracker installs flood flows in the OpenFlow switches. When a packet arrives at the switch and does not match any other flow, it is sent to the controller and all active ports in the switch. For more information on the ARP Handler, refer to Host Tracker on page 53.

OpenFlow statistics

The OpenFlow-enabled switch provides a mechanism to request supported statistics. Supported statistics are as follows.

Inventory information:

- Node description—Description of the node, such as the switch manufacturer, hardware revision, software revision, serial number, and so on
- Flow table features—Features supported by flow tables of the switch
- Port description—Properties supported by each node connector of the node
- Group features—Features supported by the group table of the switch
- Meter features—Features supported by the meter table of the switch

Statistics:

- Individual flow statistics—Statistics related to individual flow installed in the flow table
- Aggregate flow statistics—Statistics related to aggregate flow for each table level
- Flow table statistics—Statistics related to the individual flow table of the switch
- Port statistics—Statistics related to all node connectors of the node
- Group description—Description of the groups installed in the switch group table
- Group statistics— Statistics related to an individual group installed in the group table
- Meter configuration— Statistics related to the configuration of the meters installed in the switch meter table
- Meter statistics—Statistics related to an individual meter installed in the switch meter table
- Queue statistics— Statistics related to the queues created on each node connector of the switch

The controller fetches both inventory and statistics from each node whenever a node connects to the controller. The controller fetches statistics periodically within a time interval of three seconds. The controller augments inventory information and statistics fetched from each connected node to its respective location in the operational data store. The controller also cleans the stale statistics at periodic intervals.

You can see some inventory information (nodes, ports, and tables) and statistics (ports) by using the controller user interface.

You can access all the collected inventory information and statistics by using RESTCONF requests. The following sections provide a brief explanation of how each set of statistics can be accessed from the controller operational data store by using a RESTCONF request.

Example of individual flow statistics

To view individual flow statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/flow/{flow-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1
- Output:

```
<flow
    xmlns="urn:opendaylight:flow:inventory">
    <id>1</id>
    <hard-timeout>600</hard-timeout>
    <table_id>0</table_id>
    <match>
        <ipv4-destination>10.0.10.0/24</ipv4-destination>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
        </ethernet-match>
    </match>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    OUTPUT PORT
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <cookie>0</cookie>
    <idle-timeout>300</idle-timeout>
    <flags></flags>
    <flow-statistics
        xmlns="urn:opendaylight:flow:statistics">
        <duration>
            <second>4</second>
            <nanosecond>953000</nanosecond>
        </duration>
        <packet-count>0</packet-count>
        <byte-count>0</byte-count>
    </flow-statistics>
    <priority>2</priority>
</flow>
```

Example of aggregate flow statistics

To view aggregate flow statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0
- Output:

```
<aggregate-flow-statistics
      xmlns="urn:opendaylight:flow:statistics">
      <flow-count>1</flow-count>
      <packet-count>0</packet-count>
         <byte-count>0</byte-count>
</aggregate-flow-statistics>
```

Example of flow table statistics

To view flow table statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/table/{table-id}/
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/table/0/
- Output:

```
 <flow-table-statistics
    xmlns="urn:opendaylight:flow:table:statistics">
    <packets-looked-up>275</packets-looked-up>
    <active-flows>0</active-flows>
    <packets-matched>0</packets-matched>
</flow-table-statistics>
```

Example of port description and port statistics

To view port description and port statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/node-connector/{node-connector-id}

- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/node-connector/openflow:1:1
- Output

```
<node-connector
    xmlns="urn:opendaylight:inventory">
    <id>openflow:1:1</id>
    <advertised-features
        xmlns="urn:opendaylight:flow:inventory">hundred-mb-hd ten-mb-hd autoeng
copper hundred-mb-fd ten-mb-fd
    </advertised-features>
    <state
        xmlns="urn:opendaylight:flow:inventory">
        <link-down>false</link-down>
        <blocked>false</blocked>
        <live>true</live>
    </state>
    <current-speed
        xmlns="urn:opendaylight:flow:inventory">102400
    </current-speed>
    <peer-features
        xmlns="urn:opendaylight:flow:inventory">
    </peer-features>
    <configuration
        xmlns="urn:opendaylight:flow:inventory">
    </configuration>
    <maximum-speed
        xmlns="urn:opendaylight:flow:inventory">102400
    </maximum-speed>
    <name
        xmlns="urn:opendaylight:flow:inventory">eth2
    </name>
    <port-number
        xmlns="urn:opendaylight:flow:inventory">1
    </port-number>
    <hardware-address
        xmlns="urn:opendaylight:flow:inventory">08:00:27:27:E9:71
    </hardware-address>
    <supported
        xmlns="urn:opendaylight:flow:inventory">hundred-mb-hd ten-mb-hd autoeng
copper hundred-mb-fd ten-mb-fd
    </supported>
    <current-feature
        xmlns="urn:opendaylight:flow:inventory">autoeng hundred-mb-fd
    </current-feature>
    <flow-capable-node-connector-statistics
        xmlns="urn:opendaylight:port:statistics">
        <receive-over-run-error>0</receive-over-run-error>
        <duration>
            <second>1627</second>
            <nanosecond>323000</nanosecond>
        </duration>
        <receive-crc-error>0</receive-crc-error>
        <receive-errors>0</receive-errors>
        <transmit-errors>0</transmit-errors>
        <bytes>
            <transmitted>19152</transmitted>
            <received>39715</received>
        </bytes>
        <collision-count>0</collision-count>
        <transmit-drops>0</transmit-drops>
        <receive-frame-error>0</receive-frame-error>
        <packets>
            <transmitted>304</transmitted>
            <received>612</received>
        </packets>
        <receive-drops>0</receive-drops>
    </flow-capable-node-connector-statistics>
</node-connector>
```

Example of group description and group statistics

To view the group description and group statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/group/{group-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/group/1/
- Output:

```
<group
    xmlns="urn:opendaylight:flow:inventory">
    <group-id>1</group-id>
    <group-type>group-all</group-type>
    <group-desc
        xmlns="urn:opendaylight:group:statistics">
        <group-id>1</group-id>
        <buckets></buckets>
        <group-type>group-all</group-type>
    </group-desc>
    <group-statistics
        xmlns="urn:opendaylight:group:statistics">
        <duration>
            <nanosecond>785000</nanosecond>
            <second>1417442545</second>
        </duration>
        <byte-count>0</byte-count>
        <group-id>1</group-id>
        <packet-count>0</packet-count>
        <ref-count>0</ref-count>
        <buckets></buckets>
    </group-statistics>
    <buckets></buckets>
</group>
```

Example of meter configuration and meter statistics

To view the meter configuration and meter statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/meter/{meter-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/meter/1
- Output :

```
<meter
    xmlns="urn:opendaylight:flow:inventory">
    <meter-id>1</meter-id>
    <meter-statistics
        xmlns="urn:opendaylight:meter:statistics">
        <meter-band-stats>
            <band-stat>
                <band-id>0</band-id>
                <packet-band-count>0</packet-band-count>
                <byte-band-count>0</byte-band-count>
            </band-stat>
        </meter-band-stats>
        <byte-in-count>0</byte-in-count>
        <flow-count>0</flow-count>
        <meter-id>1</meter-id>
```

```
        <duration>
            <nanosecond>821000</nanosecond>
            <second>26</second>
        </duration>
        <packet-in-count>0</packet-in-count>
    </meter-statistics>
    <meter-config-stats
        xmlns="urn:opendaylight:meter:statistics">
        <flags>meter-kbps</flags>
        <meter-band-headers>
            <meter-band-header>
                <band-id>0</band-id>
                <drop-rate>50</drop-rate>
                <drop-burst-size>0</drop-burst-size>
                <band-burst-size>0</band-burst-size>
                <meter-band-types>
                    <flags>ofpmbt-drop</flags>
                </meter-band-types>
                <band-rate>50</band-rate>
            </meter-band-header>
        </meter-band-headers>
        <meter-id>1</meter-id>
    </meter-config-stats>
    <meter-band-headers>
        <meter-band-header>
            <band-id>0</band-id>
            <band-burst-size>0</band-burst-size>
            <band-rate>50</band-rate>
            <drop-burst-size>0</drop-burst-size>
            <drop-rate>50</drop-rate>
            <meter-band-types>
                <flags>ofpmbt-drop</flags>
            </meter-band-types>
        </meter-band-header>
    </meter-band-headers>
    <flags>meter-kbps</flags>
</meter>
```

Example of queue statistics

To view queue statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}/node-connector/{node-connector-id}/queue/{queue-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/node-connector/openflow:1:1/queue/1
- Output

```
<queue
    xmlns="urn:opendaylight:flow:inventory">
    <queue-id>1</queue-id>
    <flow-capable-node-connector-queue-statistics
        xmlns="urn:opendaylight:queue:statistics">
        <transmitted-packets>0</transmitted-packets>
        <transmitted-bytes>0</transmitted-bytes>
        <transmission-errors>0</transmission-errors>
        <duration>
            <nanosecond>965000</nanosecond>
            <second>60</second>
        </duration>
    </flow-capable-node-connector-queue-statistics>
</queue>
```

Example of node description

To view node description, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/
- Output:

```
<node
    xmlns="urn:opendaylight:inventory">
    <id>openflow:83989236764977397776</id>
    <description
        xmlns="urn:opendaylight:flow:inventory">None
    </description>
    <switch-features
        xmlns="urn:opendaylight:flow:inventory">
        <max_buffers>0</max_buffers>
        <x:capabilities
            xmlns:x="urn:opendaylight:flow:inventory">x:flow-feature-capability-
group-stats
        </x:capabilities>
        <x:capabilities
            xmlns:x="urn:opendaylight:flow:inventory">x:flow-feature-capability-flow-
stats
        </x:capabilities>
        <x:capabilities
            xmlns:x="urn:opendaylight:flow:inventory">x:flow-feature-capability-port-
stats
        </x:capabilities>
        <x:capabilities
            xmlns:x="urn:opendaylight:flow:inventory">x:flow-feature-capability-
queue-stats
        </x:capabilities>
        <max_tables>1</max_tables>
    </switch-features>
    <manufacturer
        xmlns="urn:opendaylight:flow:inventory">Brocade Communications, Inc
    </manufacturer>
    <hardware
        xmlns="urn:opendaylight:flow:inventory">FastIron
    </hardware>
    <serial-number
        xmlns="urn:opendaylight:flow:inventory">None
    </serial-number>
    <software
        xmlns="urn:opendaylight:flow:inventory">FI 8.0.20
    </software>
</node>
```

Example of flow table features

The controller does not fetch statistics for flow table features from the switches.

Example of group features

To view group feature statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

Accept: application/xml

- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/
- Output:

```
<group-features
        xmlns="urn:opendaylight:group:statistics">
        <max-groups>255</max-groups>
        <actions>266276865</actions>
        <group-types-supported
            xmlns:x="urn:opendaylight:group:types">x:group-all
        </group-types-supported>
        <group-types-supported
            xmlns:x="urn:opendaylight:group:types">x:group-select
        </group-types-supported>
        <group-capabilities-supported
            xmlns:x="urn:opendaylight:group:types">x:select-weight
        </group-capabilities-supported>
</group-features>
```

Example of meter features

To view meter feature statistics, send the following request to the controller:

- Headers:

  Content-type: application/xml

  Accept: application/xml
- Authentication: admin: admin
- URL: http://<controller-ip>:8181/restconf/operational/opendaylight-inventory:nodes/node/{node-id}
- Example URL: http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:1/
- Output:

```
<meter-features
        xmlns="urn:opendaylight:meter:statistics">
        <max_bands>16</max_bands>
        <meter-capabilities-supported
            xmlns:x="urn:opendaylight:meter:types">x:meter-stats
        </meter-capabilities-supported>
        <meter-capabilities-supported
            xmlns:x="urn:opendaylight:meter:types">x:meter-kbps
        </meter-capabilities-supported>
        <meter-capabilities-supported
            xmlns:x="urn:opendaylight:meter:types">x:meter-burst
        </meter-capabilities-supported>
        <max_meter>256</max_meter>
        <max_color>8</max_color>
</meter-features>
```

TLS support

A secure controller network requires that both the switches and controllers are authenticated to avoid hacking. The OpenFlow channel is usually encrypted using transport layer security (TLS). Using TLS, the OpenFlow channel is an instance that has the single network connection between the switch and the controller.

The switch and the controller communicate through a TLS connection. The TLS connection is initiated by the switch when the controller is started. The controller monitors either the user-specified TCP port or the default 6633 and 6653 TCP ports. Optionally, the TLS connection is initiated by the controller to the switch, which is monitoring either the user-specified TCP port or the default 6653 TCP port.

To have a secure connection between the controller and the OpenFlow-enabled switches, private key infrastructure (PKI) management is required. The switch and the controller mutually authenticate by exchanging certificates that are signed by a site-specific private key.

Each switch must be user configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

For more information on how to configure TLS support, refer to: https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:_TLS_Support.

NETCONF

NETCONF overview

Network Configuration Protocol (NETCONF) provides mechanisms to install, update, and delete the configuration of network devices, such as routers, switches, and firewalls. It uses Extensible Markup Language (XML)-based or JavaScript Object Notation (JSON) data encoding for the configuration data and as the protocol messages. The NETCONF protocol operations are realized as remote procedure calls (RPCs).

NETCONF is a standard that is defined by the IETF. For more information, refer to the latest RFC, which is RFC6241 as of the creation of this document.

NOTE
The devices must implement NETCONF and NETCONF monitoring as per RFC specifications. Otherwise, the controller may not be able to connect successfully, download YANG modules and manage the device.

Primary NETCONF components in the controller

TheBrocade SDN Controller acts as both a NETCONF server and client.

The NETCONF client connects to external NETCONF-enabled devices and manages them through the NETCONF connector. This component is generally referred to as the NETCONF southbound interface.

The NETCONF server component exposes the controller itself as a NETCONF-enabled device so that the controller can be managed and configured by using a NETCONF client. The controller also implements NETCONF monitoring and advertises the YANG modules that it supports through the **get-schema** method. This component is generally referred to as the NETCONF northbound interface.

Managing the controller by using a NETCONF client

You can manage the controller with a NETCONF client. The controller implements NETCONF monitoring, and, thus, the client should be able to download the supported YANG modules.

To view all the YANG modules that the controller exposes, log in to the API explorer and select the **Mounted Resources** tab and click **Controller** to list all the YANG modules that are supported by the controller. You can navigate through these models in an interactive way.

Connecting to the controller from a NETCONF client

You can connect to the controller by using the following information:

- Port: 8383
- Username: admin
- Password: admin

Managing a NETCONF-enabled device by using the controller

You can manage a NETCONF-enabled device, such as a router, by using the controller through the NETCONF southbound interface.

Communication between the NETCONF-enabled device and the controller is session based. The device and controller establish a connection and session before exchanging data and close the connection and session when the exchange of data is completed.

To mount a NETCONF device, such as a Vyatta router, the controller uses the NETCONF connector module in the NETCONF subsystem. You can dynamically connect and manage a Vyatta router or any other device that implements the NETCONF server that adheres to IETF specifications.

The RESTCONF interface is used to manage the NETCONF device at run time. Using RESTCONF, you can perform operations such as the following:

- Connecting and mounting the NETCONF-enabled device on the controller
- Modifying the connection parameters
- Deleting the device from the controller
- Gathering details of the various YANG models that are provided by the device
- Performing RPC operations on the device that is provided by the YANG models

Connecting to a NETCONF-enabled device at run time

To connect and mount a new NETCONF device, send the following request to the controller.

- HTTP Method: POST
- URL: http://<controller-ip>:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules
- Headers:

  Accept: application/xml

  Content-type: application/xml
- Payload for the POST request:

```
<module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
  <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf
">prefix:sal-netconf-connector</type>
  <name>new-netconf-device</name>
  <address
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">127.0
.0.1</address>
  <port
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">830</
port>
  <username
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">admin
</username>
  <password
```

```
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">admin<
/password>
  <tcp-only
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false<
/tcp-only>
  <event-executor
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-
event-executor</type>
    <name>global-event-executor</name>
  </event-executor>
  <binding-registry
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">prefix:bi
nding-broker-osgi-registry</type>
    <name>binding-osgi-broker</name>
  </binding-registry>
  <dom-registry
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dom-
broker-osgi-registry</type>
    <name>dom-broker</name>
  </dom-registry>
  <client-dispatcher
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf">prefix:ne
tconf-client-dispatcher</type>
    <name>global-netconf-dispatcher</name>
  </client-dispatcher>
  <processing-executor
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
    <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:thread
pool</type>
    <name>global-netconf-processing-executor</name>
  </processing-executor>
</module>
```

NOTE
Copying and pasting content from a PDF file to the NETCONF client may insert carriage breaks or delete characters. Use the HTML document at http://www.brocade.com/products/all/software-defined-networking/brocade-vyatta-controller/index.page for copying and pasting XML text strings.

Modify the following parameters:

- Name, address, username, and password to match those for the device.
- Change <controller-ip> to the name of the host or IP address on which the controller is running.

The name serves as the identifier for the mounted device. A NETCONF connector is spawned immediately. Sometimes, a few moments may pass before the NETCONF device connects successfully and downloads all necessary schemas.

Connecting to a device not supporting NETCONF monitoring

The NETCONF connector depends on ietf-netconf-monitoring support when connecting to a remote NETCONF device. The ietf-netconf-monitoring support allows the NETCONF connector to list and download all the YANG schemas that are used by the device. A NETCONF connector communicates with a device by identifying the schemas used. Some of the devices use YANG models internally but do not support NETCONF monitoring. A NETCONF connector can also communicate with these devices, but you need to manually load the necessary YANG models in the controller YANG model cache for the NETCONF connector.

For more information, refer to *Connecting to a device not supporting NETCONF monitoring* section at: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Config:Examples:Netconf.

Viewing the details of the mounted NETCONF device

To view the configuration, operation, and RPC capabilities that are provided by the mounted NETCONF devices, select the API explorer on the controller by using a supported web browser.

Access the API explorer application at: http://<controller-ip>:8181/apidoc/explorer/index.html

- Modify the following parameter:

<controller-ip> to the name of the host on which the controller is running.

After you are in the GUI, click the **Mounted Resources** tab to select the mounted NETCONF device in which you are interested. The list on the page displays the various capabilities of the device. You can click each of them to get more details.

Modifying the configuration parameters of a mounted NETCONF device

After mounting and connecting to a NETCONF device, you can modify the configuration parameters at run time. For example: If the username or password is changed for a mounted NETCONF device, you can change it at run time. Here is an example of how to change the username and password of a mounted device that is named new NETCONF device.

To modify configuration parameters of a mounted NETCONF device, send the following request to the controller:

- HTTP Method: PUT
- URL: http://<controller_ip>/restconf/config/network-topology:network-topology/topology/topology-netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-sal-netconf-connector-cfg:sal-netconf-connector/<MountName>

  Change <controller-ip> to the name or IP address of the host on which the controller is running.
- Change <MountName> to the mount name of the mounted device.
- Headers:

  Accept: application/xml

  Content-type: application/xml
- Payload for the PUT request: (this needs to be a full XML)

```
module xmlns="urn:opendaylight:params:xml:ns:yang:controller:config">
<type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf
">prefix:sal-netconf-connector</type>
<name>new-netconf-device</name>
<address
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">127.0
.0.1</address>
<port
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">830</
port>
<username
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">users
1</username>
<password
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">passw
d</password>
<tcp-only
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">false
</tcp-only>
```

```
<event-executor
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:nettyevent-
executor</type>
        <name>global-event-executor</name>
</event-executor>
<binding-registry
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">prefix:bi
nding-broker-osgi-registry</type>
        <name>binding-osgi-broker</name>
</binding-registry>
<dom-registry
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dombro
ker-osgi-registry</type>
        <name>dom-broker</name>
</dom-registry>
<client-dispatcher
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf">prefix:ne
tconf-client-dispatcher</type>
        <name>global-netconf-dispatcher</name>
</client-dispatcher>
<processing-executor
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:thread
 pool</type>
        <name>global-netconf-processing-executor</name>
    </processing-executor>
</module>
```

Deleting a mounted NETCONF device

Using RESTCONF, you can delete an instance of a module. For the NETCONF connector, the modules are deleted, the NETCONF connection is dropped, and all the resources are cleaned.

The API details that are required to delete an instance of a module follow:

- HTTP Method: Delete
- URL: http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/controller-config/yang-ext:mount/config:modules/module/odl-sal-netconf-connector-cfg:sal-netconf-connector/<new-netconf-device>
- Headers:

  Accept: application/xml

  Content-Type: application/xml

Modify the following parameters:

- <controller-ip> to the name of the host on which the controller is running.
- <new-netconf-device> to the mounted NETCONF device (the last element of the URL) according to the setup requirements.

Configuration of NETCONF devices: examples

The configuration details of all modules in the controller are located at:

http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/controller-config/yang-ext:mount/config:modules/

Modify the following parameter:

- <controller-ip> to the name of the host on which the controller is running.

The configuration of a particular device, for example, a Vyatta router, is located at:

http://<controller-ip>:8181/restconf/config/opendaylight-inventory:nodes/node/<device- mounted>/yang-ext:mount/

Modify the following parameter:

- <controller-ip> to the name of the host on which the controller is running.
- <device-mounted> to the name of the mounted NETCONF device.

For more examples and details, refer to the following wiki page: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Config:Examples:Netconf.

Troubleshooting NETCONF devices

You can use NETCONF clients to connect to NETCONF devices and troubleshoot mounting and data-related issues. You can use Netopeer; a set of NETCONF tools built on the libnetconf library. The Netopeer allows operators to connect to the NETCONF-enabled devices and control them by using the NETCONF client. For more information on Netopeer, refer to: https://code.google.com/p/netopeer.

Installing Netopeer

To install Netopeer, follow these steps:

1. Install a docker on a Linux machine and copy the Netopeer image by using the following command:

```
docker run -rm -t -p 1831:830 dockeruser/netopeer
```

2. Confirm a hello message by using the following command:

```
ssh root@localhost -p 1831 -s netconf
```

The NETCONF client sends a complete XML document that contains <rpc> elements to the server.

NOTE
You may have to run these commands as the root user.

Installing Netopeer manually

To install Netopeer manually, refer to: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Config:Examples:Netconf:Manual_netopeer_installation.

Host Tracker

Host Tracker overview

Host Tracker is a component of the OpenDaylight L2Switch application. The L2Switch has the following components that are related to Host Tracker:

- Packet Handler: This component categorizes packets into flows, gathers information from the packets, and dispatches them to their destinations.
- Loop Remover: Switches that are interconnected for redundancy can form switching loops. A single frame that cycles through the loop repeatedly can waste network resources indefinitely. Loop Remover uses Link Layer Discovery Protocol (LLDP) to learn topology information, and then uses this information to avoid forwarding frames in loops.
- ARP Handler: The L2Switch uses ARP Handler to install and manage flood flows and send Address Resolution Protocol (ARP) frames to the Brocade SDN Controller. When a packet that arrives at the switch does not match other flows, or is an ARP frame, it can be sent to the controller. Alternatively, the packets can be flooded into the network.
- Address Tracker: This component allows the controller to track observations of IPv4, IPv6, and MAC addresses, and where they are observed. Address Tracker employs ARP to resolve IP addresses used by software to media access control (MAC) addresses that are used by LAN hardware. ARP tracks the IP and MAC addresses of ARP requesters and respondents.
- Host Tracker: This component tracks the locations of hosts (by using the MAC addresses of the hosts as the primary identifier) in the network and adds them to the topology. It creates a node in the topology to represent a host and attaches it to the appropriate switches.

The components are implemented in a way that creates dependencies. For example, Loop Remover depends on the implementation of Packet Handler, and Host Tracker requires the implementation of Address Tracker and, transitively, Packet Handler.

Host Tracker pushes flows for host and topology discovery, and network flooding. Host Tracker gets the locations of hosts in the network and gathers information about the traffic flowing to a host.

Functioning of Host Tracker

Host Tracker works in the following modes.

- Flood mode: This mode is the default.
- No flood mode: This mode generates no traffic flows.

Flood mode functioning

In the flood mode, Host Tracker performs the following functions:

- Tracking hosts
- Flooding packets

Host Tracker installs two kinds of flows in OpenFlow devices:

- LLDP flows for topology discovery.
- Flooding flows to forward any incoming packet from a switch to all ports, except the source port, thus, ensuring that the frame reaches its destination. The frame is also forwarded to the controller for host tracking.

With these flows, Host tracker ensures the following:

- The controller discovers the OpenFlow network topology. For more information about topology discovery, refer to *OpenFlow*.
- The controller tracks hosts attached to the network.
- Switches dispatch packets between hosts.

NOTE

While changing the host tracker feature from flood mode to no flood mode or vice versa, ensure that you disconnect the switches, clear the existing flows, and reconnect the switch. The process effectively reflects the current state of the switch.

NOTE

Owing to the flooding of packets, Host Tracker in the flood mode is best used for testing network connectivity. In the case of large networks, or networks supporting traffic load, we recommend the use of the no flood mode.

No flood mode functioning

In the no flood mode, there is no communication between hosts. There are also no flows for traffic. Host Tracker installs two kinds of flows in OpenFlow devices:

- LLDP flows for the controller topology discovery. For more information about topology discovery, refer to *OpenFlow*.
- ARP flows for the controller to track hosts and dispatch ARP packets between hosts.

With these flows, Host Tracker ensures the following:

- The controller collects information from host ports.
- The controller tracks topology information about hosts and switches.
- The controller dispatches ARP packets between hosts.

Working with Host Tracker: Flood mode

This section provides examples of how you can use the controller with Host Tracker in the Flood mode.

The following installations are required:

- Controller

  For instructions on installing the controller, see *Brocade SDN Controller Software Installation Guide*.
- Mininet

Download and install Mininet: Downloads of Mininet and instructions for installation are available at: http://mininet.org/

1. Start Mininet.
2. Enter the following command to create a virtual network that consists of two switches.
   ```
   sudo mn --controller=remote,ip=<controller-ip> --topo linear,2
   ```
3. Enter the following command to dump all flow data from all switches:
   ```
   dpctl dump-flows
   ```

   The Mininet output for switch 1 (s1) and switch 2 (s2) follows: lines1 and 2 show flood flows, and line 3 shows the LLDP flow.

   ```
   *** s1 ------------------------------------------------------------------------
   NXST_FLOW reply (xid=0x4):
   cookie=0x2b00000000000007, duration=37.279s, table=0, n_packets=2, n_bytes=140,
   idle_age=32, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
   cookie=0x2b00000000000006, duration=37.279s, table=0, n_packets=13, n_bytes=2802,
   idle_age=9, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
   cookie=0x2b00000000000005, duration=41.278s, table=0, n_packets=9, n_bytes=567,
   idle_age=0, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
   *** s2 ------------------------------------------------------------------------
   NXST_FLOW reply (xid=0x4):
   cookie=0x2b00000000000004, duration=37.288s, table=0, n_packets=2, n_bytes=140,
   idle_age=32, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
   cookie=0x2b00000000000005, duration=37.288s, table=0, n_packets=11, n_bytes=2149,
   idle_age=6, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
   cookie=0x2b00000000000007, duration=40.213s, table=0, n_packets=8, n_bytes=504,
   idle_age=0, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
   ```
4. Generate traffic by using Mininet.
   a) Enter the following command to get host 1 (h1) to ping host 2.
      ```
      h1 ping h2
      ```
   b) Enter the following command to get all hosts to ping each other.
      ```
      pingall
      ```

   The result follows:

   ```
   *** Ping: testing ping reachability
   h1 -> h2
   h2 -> h1
   *** Results: 0% dropped (2/2 received
   ```

Viewing information about hosts and topology

To see the results of the commands that were run, you can either run a REST GET request, or view the topology on the controller GUI.

1. Access the controller GUI at http://<controller-ip>:9000/ to view the graphical display of the generated topology.
2. Use one of the RESTCONF tools that is described in the chapter on RESTCONF Tools in this guide to send the following request to access topology.

   The RESTCONF request to be sent to the controller requires the following:

   - Headers:
   - Content-type: application/xml
   - Accept: application/xml
   - Authentication: admin:admin
   - Method: GET
   - URL: http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/flow:1/

   A section of the topology that was revealed for the GET request follows:

   ```
   <network-topology
       xmlns="urn:TBD:params:xml:ns:yang:network-topology">
       <topology>
           <topology-id>flow:1</topology-id>
   ```

```
            <link>
                <link-id>host:76:22:bf:ad:50:71:openflow:2:1/openflow:2:1</link-id>
                <source>
                    <source-node>host:76:22:bf:ad:50:71</source-node>
                    <source-tp>host:76:22:bf:ad:50:71:openflow:2:1</source-tp>
                </source>
                <destination>
                    <dest-tp>openflow:2:1</dest-tp>
                    <dest-node>openflow:2</dest-node>
                </destination>
            </link>
            <link>
                <link-id>openflow:1:2</link-id>
                <source>
                    <source-node>openflow:1</source-node>
                    <source-tp>openflow:1:2</source-tp>
                </source>
                <destination>
                    <dest-tp>openflow:2:2</dest-tp>
                    <dest-node>openflow:2</dest-node>
                </destination>
            </link>
            <link>
                <link-id>host:ae:5f:38:85:40:66:openflow:1:1/openflow:1:1</link-id>
                <source>
                    <source-node>host:ae:5f:38:85:40:66</source-node>
                    <source-tp>host:ae:5f:38:85:40:66:openflow:1:1</source-tp>
                </source>
                <destination>
                    <dest-tp>openflow:1:1</dest-tp>
                    <dest-node>openflow:1</dest-node>
                </destination>
            </link>
            <link>
                <link-id>openflow:2:1/host:76:22:bf:ad:50:71:openflow:2:1</link-id>
                <source>
                    <source-node>openflow:2</source-node>
                    <source-tp>openflow:2:1</source-tp>
                </source>
                <destination>
                    <dest-tp>host:76:22:bf:ad:50:71:openflow:2:1</dest-tp>
                    <dest-node>host:76:22:bf:ad:50:71</dest-node>
                </destination>
            </link>
            <link>
                <link-id>openflow:2:2</link-id>
                <source>
                    <source-node>openflow:2</source-node>
                    <source-tp>openflow:2:2</source-tp>
                </source>
                <destination>
                    <dest-tp>openflow:1:2</dest-tp>
                    <dest-node>openflow:1</dest-node>
                </destination>
            </link>
            <link>
                <link-id>openflow:1:1/host:ae:5f:38:85:40:66:openflow:1:1</link-id>
                <source>
                    <source-node>openflow:1</source-node>
                    <source-tp>openflow:1:1</source-tp>
                </source>
                <destination>
                    <dest-tp>host:ae:5f:38:85:40:66:openflow:1:1</dest-tp>
                    <dest-node>host:ae:5f:38:85:40:66</dest-node>
                </destination>
            </link>
        </topology>
    </network-topology>
```

Working with Host Tracker: No Flood Mode

Host tracker has the ability to turn off the flood mode to prevent the host tracker from injecting flows into all devices. For a list of known issues which may require the flood mode be disabled, refer to the latest *Brocade SDN Controller Release Notes*

To turn off the no flood mode, follow the installation instruction in *Host Tracker No Flood Mode Extension User Guide*.

Verifying the functioning of Host Tracker in the No flood mode

The functioning of Host Tracker in the No flood mode differs from its functioning in the Flood mode.

1. Start Mininet.
2. Enter the following command to create a virtual network that consists of two switches.
   ```
   sudo mn --controller=remote,ip=<controller-ip> --topo linear,2
   ```
3. Enter the following command to dump all flow data from all switches:
   ```
   dpctl dump-flows
   ```
   The Mininet output for switch 1 (s1) and switch 2 (s2) follows: line 1 shows the LLDP flow, and line 2 shows the ARP flow:

   *** s1 ----------

   NXST_FLOW reply (xid=0x4):

   cookie=0x2b00000000000002, duration=22.092s, table=0, n_packets=4, n_bytes=252, idle_age=2, priority=100,dl_type=0x88cc actions=CONTROLLER:65535

   cookie=0x2b00000000000001, duration=22.092s, table=0, n_packets=0, n_bytes=0, idle_age=22, priority=1,arp actions=CONTROLLER:65535

   *** s2 ----------

   NXST_FLOW reply (xid=0x4):

   cookie=0x2b00000000000001, duration=22.097s, table=0, n_packets=4, n_bytes=252, idle_age=2, priority=100,dl_type=0x88cc actions=CONTROLLER:65535

   cookie=0x2b00000000000002, duration=22.105s, table=0, n_packets=0, n_bytes=0, idle_age=22, priority=1,arp actions=CONTROLLER:65535
4. Generate traffic by using Mininet.
   a) Enter the following command to get host 1 (h1) to ping host 2 (h2).
      ```
      h1 ping h2
      ```
   b) Enter the following command to get all hosts to ping each other.
      ```
      pingall
      ```
      In the Flood mode of functioning of Host Tracker, pings succeed because of the flood flows; pings fail in the No flood mode as no traffic flows are configured.

Clustering

Clustering overview

Brocade SDN Controller supports clustering, which facilitates high availability (HA) in your network.

In a cluster, two or more controllers operate to provide HA, reliability, and scalability than can be obtained by using a single controller. When one of the controllers fails in a cluster, resources are redirected and the workload is redistributed. Brocade recommends a three-node cluster.

In a cluster, all the controllers share a common data store. The data in the data store is stored in smaller chunks called shards.

NOTE
In the current release, the only shard strategy supported is `module`. All the data of a single module is distributed in two shards; one stores the configuration data and the other stores the operational data.

Data shards contain all or a certain segment of the data in a module. For example, the inventory shard contains all the inventory data, and the topology shard contains all the topology data. If you do not specify a module in the modules.conf file and also do not specify a shard in the module-shards.conf file, then all the data is contained in a default shard. The default shard is defined in the module-shards.conf file. Each shard must have replicas that are configured for backup.

For a three-node cluster, you can configure a replica of every defined data shard to run on each cluster node.

The following list provides some of the advantages of clustering:

- Scaling: If you have multiple controllers running in a cluster, you can potentially do more work and store more data on the controllers.
- HA: If you have multiple controllers and one of them fails, the other instances still operate.
- Data persistence: You do not lose any data gathered by your controller after a manual restart or a failure.

Clustering limitations

Individual applications and plug-ins may differ in behavior when running in a cluster environment. For more information, refer to *Brocade SDN Controller Release Notes*.

In some cases, even if most controller instances are available, some functionality may not work as expected in the clustering feature.

Specifically, if the controller instance that is handling the connection between the OpenFlow plug-in and an OpenFlow switch is unavailable, attempts to program or query that switch by the controller fail even when most controller instances are available. This unavailability occurs because OpenFlow switch failover is not supported.

In addition, if REST calls are made to a specific controller instance, they fail when that instance is unavailable. Calls can be made to a different instance but do not automatically fail over.

Setting up a three-node cluster

Brocade recommends a three-node cluster in which you can configure a replica of every defined data shard to run on each cluster node.

For the following task, it is assumed that the IP addresses of the three nodes are `10.0.0.1`, `10.0.0.2`, and `10.0.0.3`. In addition, the member IDs of the three nodes are `member-1`, `member-2`, and `member-3`.

1. Create three virtual machines that can connect to each other on a network.
2. Install a controller instance on each virtual machine, one at a time.

   For instructions on installing the controller, see *Brocade SDN Controller Software Installation Guide*.

   Each controller instance is now a node.
3. Stop the controller instance on each node by entering the following command: `/opt/bvc/bin/stop`
4. Edit the akka.conf configuration file on each controller host.
   a) Open the /opt/bvc/controller/configuration/initial/akka.conf file.
   b) Replace all instances of `127.0.0.1` with the IP address of the node.
   For example, on the first node, change the IP address to `10.0.0.1`.

   Sample content from the modified file follows:

```
remote {
      log-remote-lifecycle-events = off
      netty.tcp {
        hostname = "127.0.0.1"
        port = 2550
        maximum-frame-size = 419430400
        send-buffer-size = 52428800
        receive-buffer-size = 5242880
      }
```

   The modified file content sample is as follows:

```
remote {
      log-remote-lifecycle-events = off
      netty.tcp {
        hostname = "10.0.0.1"
        port = 2550
        maximum-frame-size = 419430400
        send-buffer-size = 52428800
        receive-buffer-size = 52428800
      }
```

   c) Set the seed-nodes configuration under the cluster section so that it is a list of all the nodes in the cluster.

NOTE
The akka.conf file has two instances of seed-nodes. The configuration must be changed for both instances.

For example, on the first node, sample content from the original file follows:

```
cluster {
seed-nodes = ["akka.tcp://opendaylight-clusterdata@10.0.0.1:2550”]
```

Sample content from the modified file follows:

```
cluster {
seed-nodes = ["akka.tcp://opendaylight-clusterdata@10.0.0.1:2550",
"akka.tcp://opendaylight-clusterdata@10.0.0.2:2550",
"akka.tcp://opendaylight-clusterdata@10.0.0.3:2550"]
```

d) In the roles section, specify a role name for the instance of the node. The default name is `member-1`. The role name for each node must be unique.
For example, if the role name of the first node is left as `member-1`, make the role name of the second node `member-2`, and the role name of the third node `member-3`. On the second node, sample content from the original file follows:

```
roles = [
"member-1"
]
```

Sample content from the modified file follows:

```
roles = [
"member-2"
]
```

5. Edit the module-shards.conf configuration file on each node. Add the role names of the other two nodes to the replica list for each shard. The module-shards.conf files for all the nodes are identical. For example, if the node role names are `member-1`, `member-2`, and `member-3`, the content of the module-shards.conf file for each node follows:

```
module-shards = [
{
            name = "default"
            shards = [
                  {
                       name="default"
                       replicas = [
                       "member-1",
                       "member-2",
                       "member-3"
                       ]
                  }
              ]
},
{
              name = "topology"
              shards = [
                    {
                        name="topology"
                        replicas = [
                        "member-1",
                        "member-2",
                        "member-3"
                        ]
                    }
               ]
},
{
               name = "inventory"
               shards = [
                     {
                        name="inventory"
                        replicas = [
                        "member-1",
                        "member-2",
                        "member-3"
                        ]
                    }
                ]
},
{
               name = "toaster"
               shards = [
                     {
                         name="toaster"
                         replicas = [
                         "member-1",
```

```
                        "member-2",
                        "member-3"
                        ]
                    }
                ]
        }
]
```

6. Start the controller instance on each node by entering the following command: `/opt/bvc/bin/start`

Verifying the cluster

You can verify the cluster by querying each node with a REST command.

To verify that the nodes have been set up properly and are connected with one another, query each node by entering the following REST GET command:

```
http://<node ip>:8181/jolokia/read/
org.opendaylight.controller:Category=Shards,name=<role-name>-shard-inventory-
config,type=DistributedConfigDatastore
```

In this example, `<role-name>` is the role name of the node. For example, the command to query node 1 follows::

```
http://10.0.0.1:8181/jolokia/read/
org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-
config,type=DistributedConfigDatastore
```

The JSON output from all three nodes must show the same value for `Leader`. For example:

```
"Leader":"member-1-shard-inventory-config"
```

Recovering a failed node

In a three-node cluster, you may sometimes see a failed node.

If a node loses connectivity with the other nodes for more than five minutes, it is assumed that the node is down. You must restart the node so that it rejoins the cluster.

If a node suffers a catastrophic failure and you reinstall the controller, you must also re-edit the akka.conf configuration file and the module-shards.conf configuration file on the controller.

Backup Manager

Backup Manager is a controller utility that allows you to save a snapshot of the controller configuration and datastore. If a controller fails, you can use Backup Manager to restore both the controller configuration and the datastore to the previously saved state.

Backup Manager overview

Backup Manager uses a single script to perform both the backup and restore operations.

The tool offers you an option of backing up the configuration, data store, or both. Similarly, in the event of a failure, Backup Manager offers you the option to restore the configuration, data store, or both. If a

controller fails, you can use Backup Manager to restore the controller configuration and the data store to the previously saved state.

Backup Manager maintains a record of all the apps and extensions that are running on your controller. In the event of a restore, Backup Manager can verify the features of the new controller installation. You can skip this verification by using the `-i` argument in the command. Inserting the `-i` flag causes the restore process to ignore any failure when checking the package. The restore process continues after printing an error message. Without the `-i` flag, any missing package causes the restore process to fail.

To see all the available options for the script, run `bvc/bin/backup_mgr.py -h` at the terminal and press Enter.

Sample output follows:

```
usage: backup_mgr.py ...
optional arguments:
-h, --help show this help message and exit
-a {backup,restore}, --action {backup,restore}
Action to perform. (default=backup)
-t {all,config,data store}, --type {all,config,datastore}
Type of files to backup or restore. (default=all)
-d DIRECTORY, --directory DIRECTORY
Directory for backup. (default=<BVC_DIR>/backups)
-f FILE, --file FILE File to restore. (default=None)
-i, --ignore_version ignore version verification failure during restore
```

Backup Manager script syntax

Saves a snapshot of the controller configuration and datastore.

Syntax

bin/backup_mgr.py --action backup --type { **all** | **config** | **datastore** } **--directory** *directory_name*

bin/backup_mgr.py --action restore --type { **all** | **config** | **datastore** } **--directory** *directory_name* **--file** */file_path/backup_date.zip* **--ignore_package_check**

Parameters

backup
: Backs up the controller.

restore
: Restores the controller.

all
: Backs up or restores both the controller configuration and datastore.

config
: Backs up or restores only the controller configuration.

datastore
: Backs up or restores only the controller datastore.

directory_ name
: Name of the directory that contains the backup file. The default directory name is bvc/backups.

/file_path/backup_date.zip
: Name of the file that is used for restoring the previously saved data and configuration. The format of the file name is `bvc_backup_yyyy_mm_dd_hh_mm_ss.zip`.

--ignore_package_check
: Ignores the results of the package check.

Usage Guidelines

Running `backup_mgr.py` without any additional parameters creates a backup of both the controller configuration and datastore.

Examples

The following command backs up the controller datastore and configuration to a directory called backups.

```
user@cluster1-node1:~/test_home\> bvc/bin/backup_mgr.py


Brocade SDN Controller Backup Manager
 Starting @ : 2015-05-21 10:21:25.097128
 Creating backup file:
  /home/vyatta/test_home/bvc/backups/bvc_backup_2015_05_21_10_21_25.zip
 Backing up configuration files
 Saved 54 config files
 Backing up datastore files
 Saved 8 datastore files
 Backing up version files
 Saved 8 version files
 Backup Manager Completed @ : 2015-05-21 10:21:25.151772
```

Backing up the controller configuration and data store

You can use Backup Manager to take a backup of the controller configuration and data store.

Brocade recommends the following best practices:

- Perform a periodic backup of your controller configuration.

The script creates a backup file on the VM where the controller is running.

- Store a copy of this backup file in an alternate location away from this VM.
- Run the script for each controller, one at a time. Wait for the controller to start running before applying the script on the next controller.

Perform the following steps to take a backup of the controller configuration and data store by using Backup Manager.

1. Create a backup directory on the VM on which you are running the controller.
2. Navigate to the controller directory.
3. Run the Backup Manager script by entering the following command: `bin/backup_mgr.py --action backup --type all --directory ~/backups/`
   The script creates a backup file for both the configuration and data store information. The operation takes a couple of minutes or more to complete, depending on your configuration. Following is an example.

```
Brocade SDN Controller Backup Manager
 Starting @ : 2015-05-20 11:45:57.288239
 Stop controller ......................... [ OK ]
 Creating backup file:
  /home/vyatta/backups/bvc_backup_2015_05_20_11_46_54.zip
 Backing up configuration files
 Saved 54 config files
 Backing up datastore files
 Saved 9 datastore files
 Backing up version files
 Saved 8 version files
 Start controller ........................ [ OK ]
 Backup Manager Completed @ : 2015-05-20 11:46:55.604668
```

   If you are backing up only the configuration files, the controller is not restarted. If you are backing up the data store files, Backup Manager stops the controller, takes the backup, and then restarts the controller. This process ensures that the data store files are not overwritten while the backup is in progress.
4. Copy the backup file to an alternate location away from the VM.

Restoring the controller configuration and data store

You can use Backup Manager to restore the controller configuration and data store.

The following list provides some important information about running Backup Manager to complete a controller restore operation.

- If you are re-creating a VM for the failed controller, the new VM must have the same IP address as the old one for Backup Manager to work.
- A restore operation for the controller by using Backup Manger always involves a controller restart.
- Backup Manager allows you to restore the configuration, the data store or both.
- During the restore operation, a new backup file is automatically generated. The backup file is of the same type (all, config, or data store) as the restore operation. This new backup file can be used for recovery if the restore operation is not successful. You can specify the target directory by using the -d option.
- During the restore operation, the contents of the current data store are completely removed. This removal prevents issues in which old and new data can get mixed up and applies to the all or data store type of restore. Restoring only the configuration does not delete the data store.

Perform the following steps to restore the controller configuration and data store by using Backup Manager.

1. Install the controller on the VM, followed by the apps and extensions, if any.

For more information, refer to *Brocade SDN Controller Software Installation Guide*.

2. Navigate to the controller directory and enter this command: `bvc/bin/backup_mgr.py --action restore --type all --file backupfile_date.zip`
   The configuration and data store of the controller are restored. The operation takes a couple of minutes or more, depending on the configuration. Following is an example.

```
> bvc/bin/backup_mgr.py --action restore --type all --file  /home/vyatta/backups/
bvc_backup_2015_05_19_15_59_43.zip


 Brocade SDN Controller Backup Manager
 Starting @ : 2015-05-20 12:26:33.571524
 Verified 8 packages
 Stop controller ......................... [ OK ]
 Creating a backup for current state.
 Creating backup file:
  /home/vyatta/test_home/bvc/backups/bvc_backup_2015_05_20_12_27_14.zip
 Backing up configuration files
 Saved 54 config files
 Backing up datastore files
 Saved 11 datastore files
 Backing up version files
 Saved 8 version files
 Restoring from backup file:
  /home/vyatta/backups/bvc_backup_2015_05_19_15_59_43.zip
 Restored 62 files.
 Start controller ......................... [ OK ]
 Backup Manager Completed @ : 2015-05-20 12:27:15.829591
```

Cluster Monitor

Cluster Monitor is a controller utility that helps to monitor the cluster nodes. The script uses the akka.conf file of each node and then uses the Jolokia protocol to query the status of each node.

Cluster Monitor overview

Cluster Monitor helps to monitor the cluster nodes. Brocade recommends that you run the Cluster Monitor script as a background process so that you can continue with other work at the terminal.

To see all the available options for the script, enter the `bvc/bin/cluster_monitor.py -h` command at the terminal and press Enter.

Sample output follows:

```
vyatta@cluster1-node1:~$ test_home/bvc/bin/cluster_monitor.py -h

usage: cluster_monitor.py [-h] <options>

optional arguments:
  -h, --help            show this help message and exit
  -t TIMER, --timer TIMER
                        repeat time in seconds (default is no repeat)
  -v, --verbose         print extra messages
  -d, --debug           print debugging messages
  -l, --uselogfile      print messages to timestamped logfile
```

Cluster Monitor script syntax

Monitors each cluster node and stores its status by using a log file.

Syntax **bvc/bin/cluster_monitor.py --timer** *timer_value* [**--verbose**] [**--debug**] **--uselogfile &**

Parameters

timer_value
: Time in seconds after which the script repeats the operation.

--verbose
: Specifies that the log file is to contain a detailed report.

--debug
: Specifies that the log file is to contain debugging messages.

--uselogfile
: Specifies that the results of running the script are to be stored in a log file. The format of the file name is `cluster_monitor_yyyy_mm_dd_hh_mm_ss.zip`.

&
: Specifies that the process is to run in the background.

Usage Guidelines You can type `jobs` at the terminal to see wether the script is running in the background and then use the `kill` command to stop the script.

All the log files are stored in the /bvc/log directory. The latest log file is named `cluster_monitor_latest.log`.

If `--uselogifle` is specified, then the output is written to a log file. The log file has a time stamp in the name. Moreover, the log file is rotated if its size exceeds 1 MB. For example, if the current log file is near 1 MB, then this file is moved to <*filename*>.1, and the new log is put in the current log file. The log file with the highest index is the oldest file. Also, only 20 such files are kept. The older ones are deleted. If this option is not specified, then the output is displayed on the screen.

If `--timer` is specified, then the script repeats the status query. The script continues to run until you issue a `kill` operation or close the terminal. Without this option, the status query is performed only once and the current status is displayed. The script stops after displaying the status. You can run the script without the `--timer` option to determine the current state of the cluster.

Examples The following example shows how to run the Cluster Monitor script every 20 seconds.

```
user@cluster1-node1:~/test_home\> bvc/bin/cluster_monitor.py --timer 20 --uselogfile &
```

BGP-PCEP Extension

About BGP-PCEP extension

The BGP-PCEP extension to the Brocade SDN Controller provides Java-based implementation of Border Gateway Protocol (BGP) and Path Computation Element Protocol (PCEP).

The extension enables the controller to use more-standardized methods of communicating network path information by providing more flexibility for the control of the network.

The extension allows you to dynamically visualize links and configure tunnels in the controller through RESTCONF calls.

Currently, BGP is the main routing protocol that is used in the Internet for the exchanging of network reachability information. The extensibility of the BGP protocol makes it an ideal carrier for communicating link-state and traffic-engineering information across autonomous systems to a centralized entity such as the Brocade SDN Controller. BGP can be used to distribute Interior Gateway Protocol (IGP) information outside the normal area of IGP.

PCEP is an IETF standard for delegating path computation to an external device such as the Brocade SDN Controller. In MPLS traffic-engineering networks, the delegation offloads complex and computationally intensive workloads, such as those implemented in path-traversal algorithms, from the head-end router to a path computation element (PCE) that resides in the controller.

The ability to program tunnels is an important feature of the BGP-PCEP extension that is implemented in the controller. Users can add, update, and delete tunnels by using RESTCONF calls. This information is communicated to the path computation client (PCC) through the PCE protocol (PCEP). The PCC receives the tunnel information and sends a response to the controller. The PCC uses the tunnel information to create a new label-switched path (LSP).

Installing the BGP-PCEP extension

This section describes how to install the BGP-PCEP extension.

Install Brocade SDN Controller version 1.3.0 or a later version of the controller.

To verify the controller version, refer to the /opt/bvc/versions directory and ensure that you see the correct version in the `com.brocade.bvc-bvc-core-bvc-controller` properties file.

To install the BGP-PCEP extension, perform the following steps.

1. Download the BGP-PCEP extension zip file.
2. Enter one the following commands depending of the BGP-PCEP file name, to unzip the BGP-PCEP extension zip file to the /opt directory.

```
unzip -o bvc-ext-bgp-packaging-*.zip -d /opt
unzip -o bvc-ext-bgp-*.zip -d /opt
```

3. Enter the following command to go to the /opt/bvc directory.
   `cd /opt/bvc`
4. Enter the following command to install the BGP-PCEP extension.
   `./install`

Verifying the installation of the BGP-PCEP extension

This section describes how to verify that you have installed the BGP-PCEP extension successfully.

To verify that you have installed the BGP-PCEP extension on the controller, perform the following steps.

- Ensure that you see the following installation notification in /opt/bvc/log/install_log.

  ```
  Installing bvc-bgpcep
  ```
- Ensure that you see a module entries for bgp-rib and pcep. To view the module entries, type `http://<controller-ip>:8181/apidoc/explorer/index` on a browser that is running on the controller host.

NOTE
The <*controller-ip*> variable is the IP address of the computer on which the controller is installed.

Using the BGP-PCEP extension

This section describes how to use the BGP-PCEP extension with the controller.

You can perform the following configurations by using the BGP-PCEP extension.

- Connecting a router to the controller by using BGP on page 68
- Adding BGP IPv4 routes to the network on page 72
- Connecting a router to the controller by using PCEP on page 73
- Setting up a PCEP tunnel between routers on page 76
- Deleting a PCEP tunnel between routers on page 77

Connecting a router to the controller by using BGP

This section describes how to connect the controller to a router by using BGP. It is assumed that the router used in this example supports the BGP link-state distribution (BGP-LS) protocol.

Ensure that you have installed the following software before you connect the controller to the router:

- Ubuntu VM version 14.04
- Brocade SDN Controller version 2.0.1 or later version

  For more information about installing the controller, refer to *Brocade SDN Controller Software Installation Guide*.
- BGP-PCEP extension for the controller

  For more information about installing the BGP-PCEP extension, refer to Installing the BGP-PCEP extension on page 67.

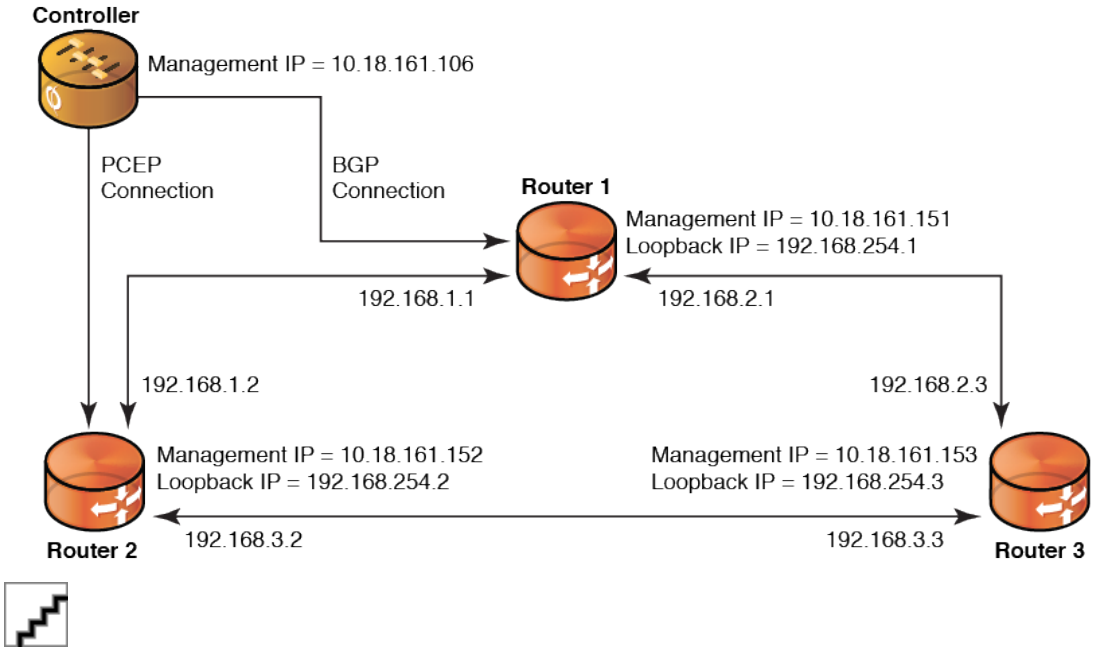
Topology for BGP Configuration

The following diagram provides the topology for the BGP configuration.

The topology setup consists of three routers and a Brocade SDN Controller with the BGP-LS and PCEP implementation. Postman REST Client can be used to configure and query the controller by using REST methods.

Router 1 is connected to the controller by using BGP. The connection is created by configuring the router and the controller xml files. You can view the network topology to verify the BGP connection. Upon successful BGP connection, the controller can receive BGP routes from the peer. Routes can also be injected into the network by using REST methods. For more information about adding routes, refer to Adding BGP IPv4 routes to the network on page 72.

FIGURE 4 BGP-PCEP use-case topology



Configuring the controller to connect with a router by using BGP

The following section describes how to configure the controller to connect with a router by using BGP. The configuration is performed at both ends, that is, at the controller and at the router.

Controller configuration

1. Enter the following command to stop the controller.
   ```
   /opt/bvc/bin/stop
   ```
2. Enter the following command to go to the opendaylight karaf directory.
   ```
   cd /opt/bvc/controller/etc/opendaylight/karaf
   ```
3. Update the following sections, in the 41-bgp-example.xml file that resides in the opendaylight karaf directory. The following steps enable you to set up the BGP connection with the peer IP address.

   a. Change the default host IP (192.0.2.1) to be the router management or loopback IP, which is 10.18.161.151 in the given example. Update the host IP in the following section of the 41-bgp-example.xml file.

   ```
   <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib
   :impl">prefix:bgp-peer</type>
   ```

```
<name>example-bgp-peer</name>
 <host>10.18.161.151</host>
 <holdtimer>180</holdtimer>
```

b. Change the following items in the section that follows.

- The local-AS value to the value of your BGP router AS configuration, which is 100 in the given example.
- The default bgp-rib-id (192.0.2.2) to the controller management IP address, which is 10.18.161.106 in the given example.

```
<type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl
">prefix:rib-impl</type>
<name>example-bgp-rib</name>
<rib-id>example-bgp-rib</rib-id>
<local-as>100</local-as>
<bgp-rib-id>10.18.161.106</bgp-rib-id>
```

c. Add the following lines for application peer configuration with the bgp-peer-id set to the controller management IP address, which is 10.18.161.106 in the given example.

```
<module>
<type
xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:bgp-
application-peer</type>
<name>example-bgp-peer-app</name>
<bgp-peer-id>10.18.162.155</bgp-peer-id>
<target-rib>
<type
xmlns:x="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">x:rib-
instance</type>
<name>example-bgp-rib</name>
</target-rib>
<application-rib-id>example-app-rib</application-rib-id>
<data-broker>
<type
xmlns:sal="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">sal:dom-
async-data-broker</type>
<name>pingpong-broker</name>
</data-broker>
</module>
```

d. If the peering router does not support link-state distribution (BGP-LS), comment the following section of the 41-bgp-example.xml file.

```
<advertized-table>
<type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:bgp:rib:impl">prefi
x:bgp-table-type</type>
<name>linkstate</name>
</advertized-table>
```

4. Enter the following command to start the controller.

```
/opt/bvc/bin/start
```

Router configuration

Configure the BGP router to advertise four-byte AS number capability; otherwise, the controller rejects the connection. If the router supports link-state distribution (BGP-LS), enable BGP link-state advertisement and IGP link-state injection to see the IGP topology in the controller.

NOTE
The router configuration might vary based on the topology configuration at your network.

To verify the configuration, refer to Verifying the BGP connection on page 70.

Verifying the BGP connection

Verify that the BGP peer is established by performing the following tasks.

1. **Verification in the router**

Use the appropriate command in the router to display details about BGP peers.

2. **Verification in the controller**
   a) Run the controller client by using `/opt/bvc/bin/client`
   b) Enter the grep command in the `/opt/bvc/controller/bin/client` folder to view the changes
   `log:display|grep established`

Verifying the BGP configuration topology

Verify that the topology was successfully received by the controller by using the following REST GET call.

Create a REST GET request by using the following information.

1. Create a REST URL.
   `http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/`
2. Use the following headers:
   - Content-Type: `application/xml`
   - Accept: `application/xml`
   - Authentication: `admin:admin`
3. Verify the number of links and nodes. Based on the topology used in this example, you should see three nodes and six links because the routers have bidirectional links.
   The sample output follows.

NOTE
The output has been truncated due to its length.

```
<topology
    xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>example-linkstate-topology</topology-id>
    <link>
        <link-id>bgpls://Ospf:0/type=link&amp;local-as=100&amp;local-
domain=3232300545&amp;local-area=0&amp;local-router=3232300547&amp;remote-
as=100&amp;remote-domain=3232300545&amp;remote-area=0&amp;remote-
router=3232300545&amp;ipv4-iface=192.168.2.3&amp;ipv4-neigh=192.168.2.1</link-id>
        <destination>
            <dest-tp>bgpls://Ospf:0/type=tp&amp;ipv4=192.168.2.1</dest-tp>
            <dest-node>bgpls://Ospf:0/
type=node&amp;as=100&amp;domain=3232300545&amp;area=0&amp;router=3232300545</dest-
node>
        </destination>
        <igp-link-attributes
            xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-topology">
            <metric>1</metric>
            <ospf-link-attributes
                xmlns="urn:TBD:params:xml:ns:yang:ospf-topology">
                <ted></ted>
            </ospf-link-attributes>
        </igp-link-attributes>
        <source>
            <source-tp>bgpls://Ospf:0/type=tp&amp;ipv4=192.168.2.3</source-tp>
            <source-node>bgpls://Ospf:0/
type=node&amp;as=100&amp;domain=3232300545&amp;area=0&amp;router=3232300547</
source-node>
        </source>
    </link>
 <topology-types></topology-types>
    <node>
        <node-id>bgpls://Ospf:0/
type=node&amp;as=100&amp;domain=3232300545&amp;area=0&amp;router=3232300546</node-
id>
        <igp-node-attributes
            xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-topology">
```

```
            <ospf-node-attributes
                xmlns="urn:TBD:params:xml:ns:yang:ospf-topology">
                <ted>
                    <te-router-id-ipv4>0.0.0.112</te-router-id-ipv4>
                </ted>
            </ospf-node-attributes>
            <router-id>0.0.0.112</router-id>
            <prefix>
                <prefix>192.168.1.0/24</prefix>
                <metric>1</metric>
            </prefix>
            <prefix>
                <prefix>192.168.254.2/32</prefix>
                <metric>1</metric>
            </prefix>
            <prefix>
                <prefix>192.168.3.0/24</prefix>
                <metric>1</metric>
            </prefix>
        </igp-node-attributes>
        <termination-point>
            <tp-id>bgpls://Ospf:0/type=tp&amp;ipv4=192.168.3.2</tp-id>
            <igp-termination-point-attributes
                xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-topology">
            </igp-termination-point-attributes>
        </termination-point>
        <termination-point>
            <tp-id>bgpls://Ospf:0/type=tp&amp;ipv4=192.168.1.2</tp-id>
            <igp-termination-point-attributes
                xmlns="urn:TBD:params:xml:ns:yang:nt:l3-unicast-igp-topology">
            </igp-termination-point-attributes>
        </termination-point>
    </node>
</topology>
```

Adding BGP IPv4 routes to the network

To add BGP IPv4 routes to the network, perform the following steps.

1. Create a REST POST request by using the following information.
   a) Create a REST URL.
   `http://<controller-ip>:8181/restconf/config/bgp-rib:application-rib/example-app-rib/tables/bgp-types:ipv4-address-family/bgp-types:unicast-subsequent-address-family/`
   b) Use the following headers:
   - Content-Type: `application/xml`
   - Accept: `application/xml`
   - Authentication:`admin: admin`

   c) Create a payload for your network configuration by using the following payload example.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <ipv4-routes xmlns="urn:opendaylight:params:xml:ns:yang:bgp-rib">
     <ipv4-route>
         <prefix>200.20.160.41/32</prefix>
         <attributes>
             <ipv4-next-hop>
                 <global>199.20.160.41</global>
             </ipv4-next-hop><as-path/>
             <multi-exit-disc>
                 <med>0</med>
             </multi-exit-disc>
             <local-pref>
                 <pref>100</pref>
             </local-pref>
             <originator-id>
                 <originator>41.41.41.41</originator>
             </originator-id> <origin>
                 <value>igp</value>
             </origin>
             <cluster-id>
                 <cluster>40.40.40.40</cluster>
```

```
                </cluster-id>
                </attributes>
              </ipv4-route>
    </ipv4-routes>
```

2. Use the commands available on the router to verify that the routes have been injected.

Connecting a router to the controller by using PCEP

This section describes how to connect a router (PCC) to the controller (PCE) by using PCEP. It is assumed that the router used in this example supports the PCEP protocol.

Ensure that you have installed the following software before you connect the controller to the router:

- Ubuntu VM version 14.04
- Brocade SDN Controller version 1.3.0 or later version

  For more information about installing the controller, refer to *Brocade SDN Controller Software Installation Guide*.
- BGP-PCEP extension for the controller

  For more information about installing the BGP-PCEP extension, refer to Installing the BGP-PCEP extension on page 67.

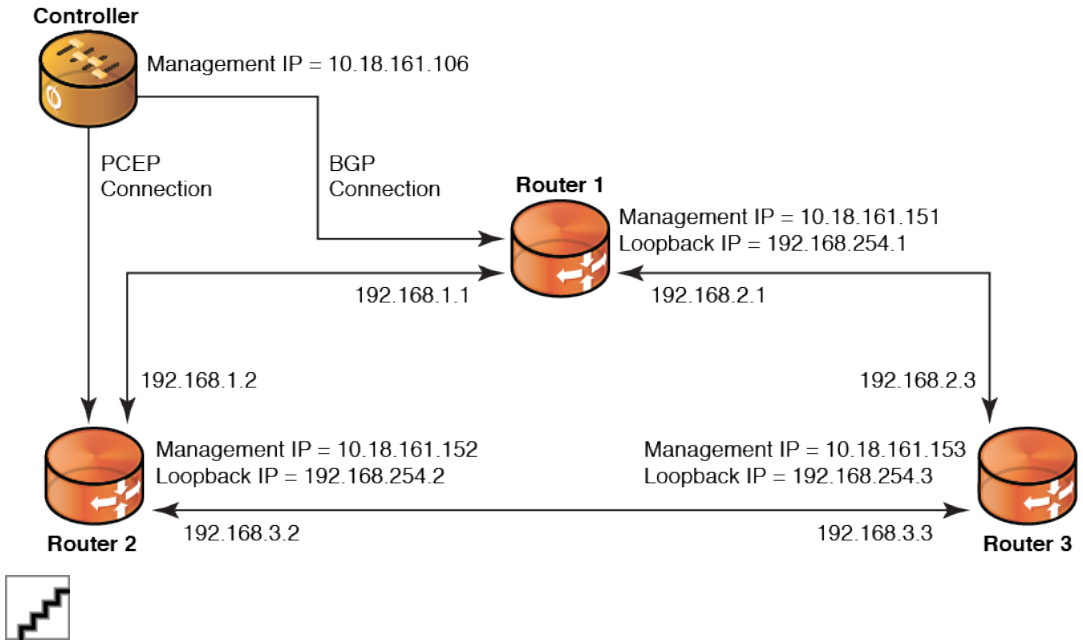
Topology for the PCEP Configuration

The following diagram provides the topology for the PCEP configuration.

The topology setup consists of three routers and a Brocade SDN Controller with the BGP-LS and PCEP implementation. Postman REST Client can be used to configure and query the controller by using REST methods.

Router 2 is connected to the controller by using PCEP. The connection is created by configuring the router and the controller xml files. You can view the network topology to verify the PCEP connection. Upon successful PCEP connection, you can create a tunnel between Router 2 and Router 3 by using REST methods. You can update the tunnel to provide an explicit path from Router 2 to Router 3 through Router 1. For more information about creating a tunnel and updating it, refer to Setting up a PCEP tunnel between routers on page 76.

FIGURE 5 BGP-PCEP use-case topology



Configuring the controller to connect with a router by using PCEP

The following section describes how to configure the controller to connect with a router by using PCEP. The configuration is performed at both ends, that is, at the controller and at the router.

NOTE
The controller configuration is required only when you are changing the PCEP version from stateful-07 to stateful-02.

Controller configuration

1. Enter the following command to stop the controller.
   `/opt/bvc/bin/stop`
2. Enter the following command to go to the opendaylight karaf directory.
   `cd /opt/bvc/controller/etc/opendaylight/karaf`
3. Edit the 32-pcep.xml file as follows.

   a. Switch commented code to ignore the stateful-7 and ietf-initiated-00 versions.

```
<!-- This block is draft-ietf-pce-stateful-pce-07 + draft-ietf-pce-inititated-
pce-00 -->
 <!--extension>
  <type>pcepspi:extension</type>
  <name>pcep-parser-ietf-stateful07</name>
 </extension>
 <extension>
  <type>pcepspi:extension</type>
  <name>pcep-parser-ietf-initiated00</name>
 </extension-->
 <!-- This block is draft-ietf-pce-stateful-pce-02 + draft-crabbe-pce-
inititated-pce-00 -->
<extension>
 <type
xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">pcepspi:
extension</type>
 <name>pcep-parser-ietf-stateful02</name>
```

```
</extension>
<extension>
 <type
xmlns:pcepspi="urn:opendaylight:params:xml:ns:yang:controller:pcep:spi">pcepspi:e
xtension</type>
 <name>pcep-parser-crabbe-initiated00</name>
</extension>
```

b. Ensure that the proposal matches your chosen draft version. Change *stateful07-proposal* to *stateful02-proposal* in the following proposal.

```
<pcep-session-proposal-factory>
    <type>pcep:pcep-session-proposal-factory</type>
    <name>stateful02-proposal</name>
</pcep-session-proposal-factory>
```

4. Edit the 32-pcep-provider.xml file to ensure that the proposal matches your chosen draft version. Change *stateful07-proposal* to *stateful02-proposal* in the following proposal.

```
<stateful-plugin>
    <type>prefix:pcep-topology-stateful</type>
    <name>stateful02</name>
</stateful-plugin>
```

5. Enter the following command to restart the controller.

```
/opt/bvc/bin/start
```

Router configuration

Configure the router to connect to the controller management IP. Enable the following protocols to setup tunnels: OSPF or ISIS, RSVP, MPLS, and Traffic Engineering.

NOTE
The router configuration might vary based on the topology configuration in your network.

To verify the configuration, refer to Verifying the PCEP connection on page 75.

Verifying the PCEP connection

Verify that the PCEP peer connection is established between the router and the controller by performing the following steps.

1. **Verification in the router**
   Run the appropriate command that is available in the router to verify the PCEP peer connection.
2. **Verification in the controller**
   Verify that the topology was successfully received by the controller by using the following REST GET call.

```
http://<controller-ip>:8181/restconf/operational/network-topology:network-topology/
topology/pcep-topology
```

The sample output follows.

```
<topology
    xmlns="urn:TBD:params:xml:ns:yang:network-topology">
    <topology-id>pcep-topology</topology-id>
    <topology-types>
        <topology-pcep
            xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
        </topology-pcep>
    </topology-types>
    <node>
        <node-id>pcc://10.18.161.152</node-id>
        <path-computation-client
            xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
            <reported-lsp>
                <name>update-tunel</name>
                <lsp

xmlns="urn:opendaylight:params:xml:ns:yang:pcep:crabbe:stateful:02">
                    <processing-rule>false</processing-rule>
                    <operational>false</operational>
                    <remove>false</remove>
```

```
                    <plsp-id>21</plsp-id>
                    <delegate>true</delegate>
                    <ignore>false</ignore>
                    <sync>false</sync>
                    <tlvs>
                        <symbolic-path-name>
                            <path-name>dXBkYXRlLXR1bmVs</path-name>
                        </symbolic-path-name>
                    </tlvs>
                </lsp>
                <path>
                    <lsp-id>0</lsp-id>
                </path>
            </reported-lsp>
            <state-sync>initial-resync</state-sync>
            <ip-address>10.18.161.152</ip-address>
            <stateful-tlv>
                <stateful

xmlns="urn:opendaylight:params:xml:ns:yang:pcep:crabbe:stateful:02">
                    <include-db-version>false</include-db-version>
                    <initiation

xmlns="urn:opendaylight:params:xml:ns:yang:pcep:crabbe:initiated:00">true
                    </initiation>
                    <lsp-update-capability>true</lsp-update-capability>
                </stateful>
            </stateful-tlv>
        </path-computation-client>
    </node>
</topology>
```

Setting up a PCEP tunnel between routers

To set up a PCEP tunnel between routers, perform the following steps.

1. Create a REST POST request by using the following information.
   a) Create a REST URL.
   `http://<controller-ip>:8181/restconf/operations/network-topology-pcep:add-lsp`
   b) Use the following headers:
      - content-Type: `application/xml`
      - Accept: `application/xml`
      - Authentication: `admin: admin`
   c) To disable PCEP segment routing:
   Edit the 33-pcep-segment-routing.xml and comment the global-pcep-dispatch configuration as follows:

```
<!-- <type
xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:pcep:impl">prefix:p
cep-dispatcher-impl</type><name>global-pcep-dispatcher</name>
```

   d) Create a payload for your network configuration by using the following payload example.

```
 <input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
<node>pcc://10.18.161.152</node>
<name>update-tunel</name>
<arguments>
 <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
  <delegate>true</delegate>
  <administrative>true</administrative>
 </lsp>
 <endpoints-obj>
  <ipv4>
   <source-ipv4-address>192.168.254.2</source-ipv4-address>
   <destination-ipv4-address>192.168.254.3</destination-ipv4-address>
  </ipv4>
 </endpoints-obj>
  <ero>
    <subobject>
     <loose>false</loose>
     <ip-prefix><ip-prefix>192.168.1.1/32</ip-prefix></ip-prefix>
```

```
      </subobject>
      <subobject>
       <loose>false</loose>
       <ip-prefix><ip-prefix>192.168.2.3/32</ip-prefix></ip-prefix>
      </subobject>
    </ero>
  </arguments>
  <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
topology-ref>
</input>
```

2. Update the tunnel by giving an explicit route object of Router 1. Create a REST POST call by using the following information.

   a) Create a REST URL.
   `http://<controller-ip>:8181/restconf/operations/network-topology-pcep:update-lsp`

   b) Use the following headers:

   - content-Type: `application/xml`
   - Accept: `application/xml`
   - Authorization: `admin: admin`

   c) Create a payload for your network configuration by using the following payload example.

```
<?xml version="1.0" encoding="UTF-8"?>
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
   <node>pcc://10.18.161.152</node>
   <name>update-tunel</name>
   <arguments>
      <lsp xmlns="urn:opendaylight:params:xml:ns:yang:pcep:ietf:stateful">
         <delegate>true</delegate>
         <administrative>true</administrative>
      </lsp>
      <ero>
         <subobject>
            <loose>false</loose>
            <ip-prefix>
               <ip-prefix>192.168.1.1/32</ip-prefix>
            </ip-prefix>
         </subobject>
         <subobject>
            <loose>false</loose>
            <ip-prefix>
               <ip-prefix>192.168.2.3/32</ip-prefix>
            </ip-prefix>
         </subobject>
      </ero>
      <operational
xmlns="urn:opendaylight:params:xml:ns:yang:pcep:crabbe:stateful:02">true</
operational>
   </arguments>
   <network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-
topology">/topo:network-topology/topo:topology[topo:topology-id="pcep-
topology"]</network-topology-ref>
</input>
```

3. Verify that the tunnel is set up and running on the router.

Deleting a PCEP tunnel between routers

To remove a PCEP tunnel between routers perform the following steps. Create a REST DELETE request by using the following information.

1. Create a REST URL.
   `http://<controller-ip>:8181/restconf/operations/network-topology-pcep:remove-lsp`
2. Use the following headers:

- content-Type: `application/xml`
- Accept: `application/xml`
- Authentication: `admin: admin`

3. Create a payload for your network configuration by using the following payload example.

```
<input xmlns="urn:opendaylight:params:xml:ns:yang:topology:pcep">
<node>pcc://10.18.161.152</node>
<name>update-tunel</name>
<network-topology-ref xmlns:topo="urn:TBD:params:xml:ns:yang:network-topology">/
topo:network-topology/topo:topology[topo:topology-id="pcep-topology"]</network-
topology-ref>
</input>
```

Host Tracker No Flood Mode Extension

About Host Tracker No Flood Mode Extension

Host Tracker is a component of the OpenDaylight L2Switch application and is an extension that is used to disable flood mode on the Host Tracker. Host Tracker tracks the locations of hosts (by using the MAC addresses of the hosts as the primary identifier) in the network and adds them to the topology. Host Tracker creates a node in the topology to represent a host and attaches it to the appropriate switches.

Installing the Host Tracker No Flood Mode extension

This section provides installation instructions for the Host Tracker No Flood Mode extension.

Host Tracker No Flood Mode is installed when you run the controller installation script. To install the Host Tracker No Flood Mode extension, perform the following steps.

Install the controller by running the `./installbsc` installation script.

For more information about the script, refer to *Brocade SDN Controller Software Installation Guide*.

- If you run the `./installbsc` installation script without any arguments, the script prompts you before installing Host Tracker No Flood Mode.
- If you run the `./installbsc` installation script command with the `-y` argument, Host Tracker No Flood Mode is installed without any prompts.

NOTE
To install Host Tracker No Flood Mode after the controller installation has completed, you must rerun the controller installation script.

Troubleshooting

Best practices for installing the controller

Follow the best practices that are outlined in this section for installing the controller. The following list provides key recommendations from Brocade.

- Ensure that you meet all the prerequisites and system requirements that are outlined in the preceding sections.

  Install OpenJDK, OpenSSH, Pyscopg, and PSmisc (for Red Hat Enterprise Linux only) before you install the controller. If you do not install these before you run the installation script, the script prompts you to install the missing prerequisite and then rerun the installation script. The controller installation script installs Zip, Unzip, and Node.js as part of the installation procedure. You do not have to install these prerequisites separately before you run the installation script.
- Install the latest patch or minor version of OpenJDK Java, version 67 or later.

  NOTE
  OpenJDK Java 7 is not supported.
- The controller user interface is tested with Node.js version 0.10.29. Only Node.js 0.10 version 29 or later is supported.

  If you have a version earlier than 29 (for example 0.10.11), upgrade to the latest Node.js version that is supported.
- You must have an Internet connection available to run the installation procedure.
- You must have desktop access to the system on which you are going to install the controller.

Troubleshooting installation

If you get an error message about a missing prerequisite during installation, follow the instructions in *Brocade SDN Controller Software Installation Guide* to install the required plug-in and then rerun the installation script.

If you see an error message similar to `OSError: [Errno 2] No such file or directory` at run time, you can ignore the message. This error does not affect the functionality of the controller.

Troubleshooting increased CPU and memory usage

If your system is slow after you install the controller, you can monitor the consumption of system resources. Linux contains many commands to check memory and CPU consumption.

To troubleshoot increased CPU and memory usage, perform the steps below:

1. Start the controller.

   To start the controller, refer to *Brocade SDN Controller Software Installation Guide*.
2. To check the memory usage, enter the following command: `top -o %MEM`
   A list of processes with memory consumption quantities is displayed.
3. To check the CPU usage, enter the following command: `top -o %CPU`
   A list of processes with CPU consumption quantities is displayed.
4. Stop the controller.

   To stop the controller, refer to *Brocade SDN Controller Software Installation Guide*.
5. Re-enter the commands in steps 2 and 3 to check the consumption of system resources when the controller is not running.

Using the results from the procedure, you can evaluate whether you need to disable specific applications or if your system requires more hardware resources.

Troubleshooting the inability to access the controller user interface from a remote system

You cannot access the user interface of the controller that is installed on a system with more than one network interface card from a remote system if the IP address of the first network interface card is not externally reachable.

If the controller is installed on a machine that is more than one network interface card, the controller uses the IP address of the first card it gets by using the ifconfig command. If this IP address cannot be reached from the external network, you are not able to access the controller web GUI from the remote machine. Although you can access the web GUI login page by using the externally reachable IP address of the controller machine, you are not able to log in successfully.

The controller web GUI uses the chosen IP address to send REST calls directly to the controller. Because that IP address cannot be reached externally, the call fails.

1. Open a Chrome browser in the remote system to derive the IP address used by the user interface to reach the controller. Follow the steps below:
   a) Open a Chrome browser window and enter the controller user interface address.
   b) Go to **Settings** > **More Tools** > **Development Tools**
      The **Developer Tools** window is displayed.
   c) Click **Network** in the **Development Tools** window.
   d) Use the admin username and admin password to log in to the controller user interface.
      The **Network** tab displays a failed REST call sent by the user interface to the controller.
   e) Locate the IP address of the controller in the details and store it in a notepad for future reference.
2. Navigate to the configuration file by entering the following command at a console of the controller host: `cd /opt/bvc/web/.`

   The configuration file is named config.json.
3. Open the config.json file and change the IP address to the externally reachable IP address that you noted in step 1.
4. Save the changes and refresh the user interface page.
   You are now able to log in to the controller user interface page successfully.

Accessing the controller support diagnostic script

The controller provides a diagnostic script to help you with troubleshooting. When run, the script creates an output.

1. Access the support diagnostic script by entering the following command: `cd /opt/bvc/bin`
2. Run the support diagnostic script by entering the following command: `./support_diagnostics`
   The `ctrlr.data.*.zip` diagnostic output file is created in the `/opt/bvc/` location.

Accessing the controller log files

If you are familiar with the Karaf console, you can view the controller log files to look for event notifications.

Go to the log directory by entering this command: `cd /opt/bvc/logs/controller_logs`
The karaf.log file is located here.

Enabling garbage collection logging

You can enable garbage collection logging (GC logging) and view the logs to analyze heap memory usage.

1. To enable basic GC logging, edit the controller/bin/karaf file as follows: In the setupDefaults routine add the -XX:+PrintGC option under the DEFAULT_JAVA_OPTS section.
2. Specify a log file by adding the -Xloggc:<file> option. For example, specifying -Xloggc:gc.log writes the GC log output to the controller/gc.log file.

NOTE
More GC logging options, such as XX:+PrintGCDetails and -XX:+PrintGCTimeStamps, are available for GC logging. For more information, consult Java developer resources that are obtainable online.

Glossary

Brocade SDN Controller platform	A server-and-user platform process that contains the functionality of the base controller. For example: SDN-Controller-*version*-Software.tar.gz
Brocade SDN Controller app	An app that runs on the Brocade SDN Controller platform and provides additional usage cases, features, or both. For example: Flow-Manager-*version*-Software.zip.zip
Brocade SDN Controller extension	An extension that modifies or configures the Brocade SDN Controller platform or app. Typically, an extension is bundled with the Brocade SDN Controller platform, a Brocade SDN Controller app, or another Brocade product. For example: bvc-ext-openflow-packaging-*version*.zip