*53-1003587-02*
*12 January 2015*

# Brocade Vyatta Controller

## User Guide

Supporting Brocade Vyatta Controller v1.1.1

BROCADE

**© 2015, Brocade Communications Systems, Inc. All Rights Reserved.**

ADX, Brocade, Brocade Assurance, the B-wing symbol, DCX, Fabric OS, HyperEdge, ICX, MLX, MyBrocade, OpenScript, The Effortless Network, VCS, VDX, Vplane, and Vyatta are registered trademarks, and Fabric Vision and vADX are trademarks of Brocade Communications Systems, Inc., in the United States and/or in other countries. Other brands, products, or service names mentioned may be trademarks of others.

Notice: This document is for informational purposes only and does not set forth any warranty, expressed or implied, concerning any equipment, equipment feature, or service offered or to be offered by Brocade. Brocade reserves the right to make changes to this document at any time, without notice, and assumes no responsibility for its use. This informational document describes features that may not be currently available. Contact a Brocade sales office for information on feature and product availability. Export of technical data contained in this document may require an export license from the United States government.

The authors and Brocade Communications Systems, Inc. assume no liability or responsibility to any person or entity with respect to the accuracy of this document or any loss, cost, liability, or damages arising from the information contained herein or the computer programs that accompany it.

The product described by this document may contain open source software covered by the GNU General Public License or other open source license agreements. To find out which open source software is included in Brocade products, view the licensing terms applicable to the open source software, and obtain a copy of the programming source code, please visit http://www.brocade.com/support/oscd.

# 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 and modifiers |
| | Identifies paths and Internet addresses |
| | 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. |

| Convention | Description |
|---|---|
| value | In Fibre Channel products, a fixed value provided as input to a command option is printed in plain text, for example, **--show** WWN. |
| [ ] | Syntax components displayed within square brackets are optional.<br>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.<br>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:<br>• My Cases through MyBrocade<br>• Software downloads and licensing tools<br>• Knowledge Base | Required for Sev 1-Critical and Sev 2-High issues:<br>• Continental US: 1-800-752-8061<br>• Europe, Middle East, Africa, and Asia Pacific: +800-AT FIBREE (+800 28 34 27 33)<br>• For areas unable to access toll free number: +1-408-333-6061<br>• Toll-free numbers are available in many countries. | support@brocade.com<br>Please include:<br>• Problem summary<br>• Serial number<br>• Installation details<br>• 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 Brocade Vyatta Controller version 1.1.1.

## What is new in this document

Brocade Vyatta Controller version 1.1.1 is the current release.

## Supported platforms

The Brocade Vyatta Controller is supported on the Ubuntu 14.04 operating system.

## Brocade Vyatta Controller publications

The publications for the Brocade Vyatta Controller are as follows:

- *Brocade Vyatta Controller User Guide*
- *Brocade Vyatta Controller Quick Start Guide*
- *Brocade Vyatta Controller Release Notes*
- *Brocade Vyatta Path Explorer User Guide*
- *Brocade Vyatta vRouter 5600 EMS User Guide*

# Brocade Vyatta Controller Overview



## About the controller

This section provides a brief overview of the Brocade Vyatta Controller.

The controller is the first commercial distribution based on the OpenDaylight Helium release.

The features of the controller include:

- Brocade installer
- Brocade GUI
- Multivendor support through YANG models and OpenFlow or NETCONF standards
- Third-party extensions supported through a REST API or embedded Java OSGi plug-ins
- Modular structure provisioned by Apache karaf

## Features of the 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. For more information on installing the controller GUI, refer to *Brocade Vyatta Controller Quick Start Guide*. For more information on using the controller GUI, refer to About the controller GUI on page 15.
- **NETCONF**—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 exposes the controller itself as a NETCONF-enabled device so that the controller can be managed and configured by using a NETCONF client. For more information, refer to NETCONF overview on page 45.
- **OpenFlow**—The controller supports OpenFlow. OpenFlow components consist of the controller, an OpenFlow-enabled switch, and the OpenFlow protocol, and enables you to run experimental flow protocols on your network. For more information, refer to OpenFlow overview on page 27.
- **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. For more information about the L2 switch, refer to Host Tracker overview on page 51.
- **Clustering**—The controller supports the clustering feature that facilitates high availability (HA) in your network. The controller supports the clustering feature that allows one or more controller instances to share a common data store, which means that any instance can service a request. For more information about clustering, refer to Clustering overview on page 57.

## Apps for the controller

The controller provides additional apps as add-on packages. You must install the apps after you install the controller. You can access the apps by using the controller GUI.

The Element Management System (EMS) app and the Path Explorer app are part of the controller package.

The EMS is an app that manages the Vyatta 5600 vRouter and is used to create IPsec VPN tunnels across geographically diverse data centers by using the Internet. Using the EMS, you can perform the following tasks:

- Manage all configuration communications of the Vyatta 5600 vRouter by using the NETCONF protocol and data stores that are modeled in the YANG user interface
- Build IPsec VPN tunnels using either the IPv4 or IPv6 protocol

The Path Explorer app enables automatic flow programming of devices in a network. If you provide the source, destination address, and switch information, the system calculates the shortest path between the two hosts and sets up a path between them. The system installs the required flow to the network devices to set up a path between the hosts.

# 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 tasks to enable you to get started with the controller. Refer to *Brocade Vyatta Controller Quick Start 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 Vyatta Controller GUI



## About the controller GUI

The controller provides a graphical user interface (GUI). The GUI interacts with the controller by using REST APIs and displays information such as network topology, nodes, and so on.

For information about installing the controller GUI, refer to *Brocade Vyatta Controller Quick Start Guide*.

The controller provides the following features in the GUI:

- Topology
- Nodes
- YANG UI
- Path Explorer
- Vyatta EMS

## Logging in to the controller GUI

Install the controller by using the instructions that are in *Brocade Vyatta Controller Quick Start Guide*.

To log in to the controller GUI, perform the following steps.

1. Open a supported browser and enter this login URL: http://<controller-ip>:9000. The <controller-ip> variable, is the IP address of the computer, on which the controller is installed.
2. Log in as **admin**.
3. Click **Login**.

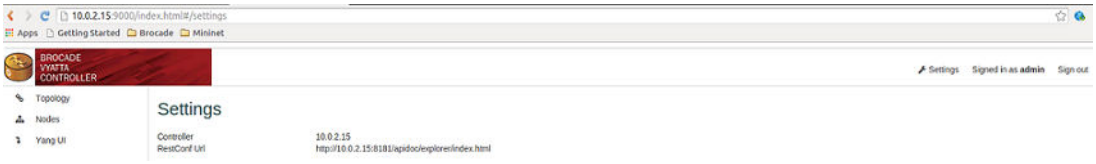
**NOTE**
The ID is **admin** and the password is also **admin**.

## Navigating the controller GUI

The controller GUI contains three parts:

- Application pane—This pane is at the left side of the window. The application pane displays the built-in and the installed add-on applications that are supported by the controller GUI.
- Content pane—This pane is at the right side of the window. Use this pane to work with the feature that you selected in the application pane.
- Tool bar—The tool bar is at the top row of the window. The tool bar presents the following options:
  - **Settings**—Provides information about the controller that is in use
  - **Signed in as**—Provides information about the signed-in user
  - **Sign Out**—Logs you off the controller GUI

**FIGURE 1** Controller GUI




# Viewing network topology

The **Topology** tab displays a graphical representation of the network topology.

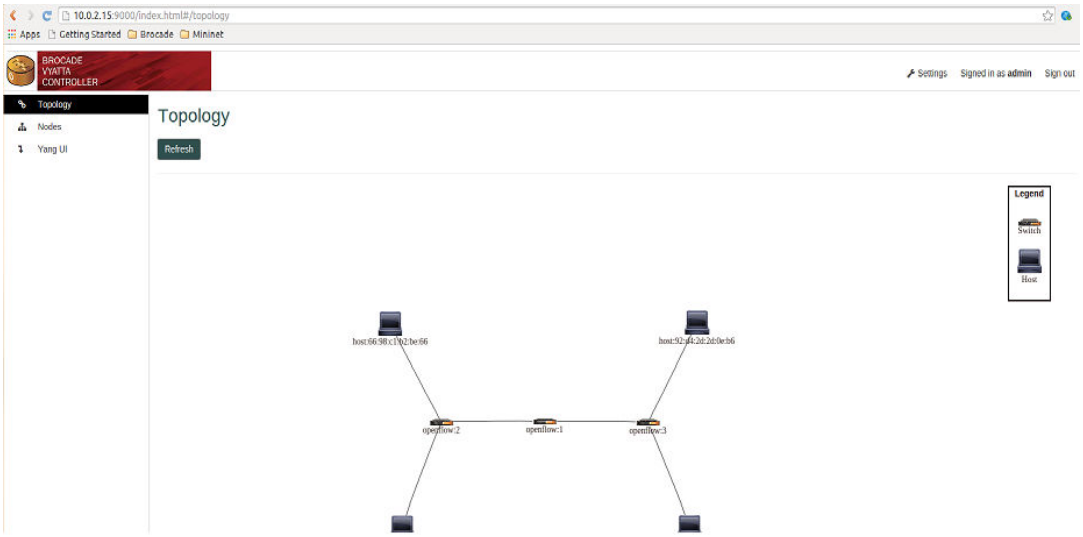
The controller determines the topology among all the network switches that are connected to the controller. The controller GUI then uses the collected topology details for graphical representation.

The controller GUI does not provide the ability to add or update information about the topology.

To view the network topology, perform the following steps.

1. Select **Topology** from the application pane.
   You see the graphical representation in the content pane. The graphical representation displays a legend about the network elements.

**FIGURE 2** Topology View



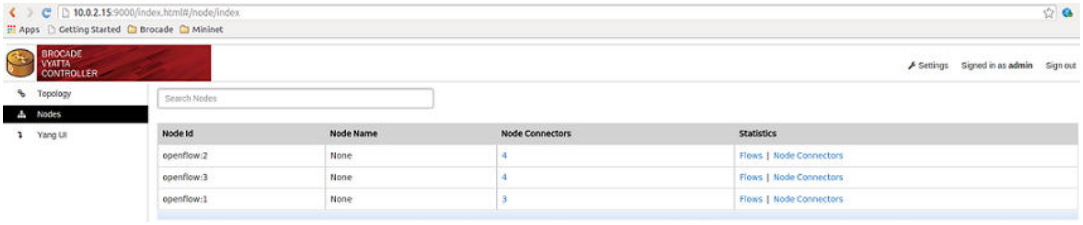

2. Hover the cursor over a host, link, or switch, to view source and destination ports.
3. Zoom in and zoom out by scrolling up or down, respectively, to view the topology for large networks.

# Viewing network statistics

The **Nodes** module in the application pane allows you to view node information and port statistics for the switches in the network.

1. Select **Nodes** from the application pane. The content pane displays a table that lists all the nodes, node connectors, and statistics.

**FIGURE 3** Nodes




2. Enter a complete node ID in the **Search Nodes** box to see only that node or enter a partial node ID in the **Search Nodes** box to search for matching nodes.
3. The number in the Node Connectors column denotes the number of ports for the switch. Click the number in the Node Connectors column to view details, such as the port ID, port name, number of ports for each switch, MAC address, and so on.
4. Click **Flows** in the statistics column to view flow-table statistics for the particular node. The table statistics include node information, such as the table ID, packet match, active flows, and so on.
5. Click **Node Connectors** to view node connector statistics for the node.

# Using the YANG UI

The YANG UI application on the controller GUI enables you to access the controller configuration and its state by using YANG modules (RESTCONF).

## Navigating the YANG UI

This section describes the high-level tasks that can be performed with the YANG UI and how to navigate it.

A YANG module defines a data structure (model) for network resources. The controller provides a technique to access the network resource or a group of network resources by using the standard REST API methods such as GET, PUT, POST, and DELETE. The resource can be data that is represented by using XML or JSON format or a remote procedure call (RPC). The following procedure uses the **opendaylight-inventory** resource as an example in the YANG UI.

To use the YANG UI, select **YANG UI** from the application pane. You can perform the following operations in the content pane.

1. Click the plus button **+** that preceeds opendaylight-inventory.

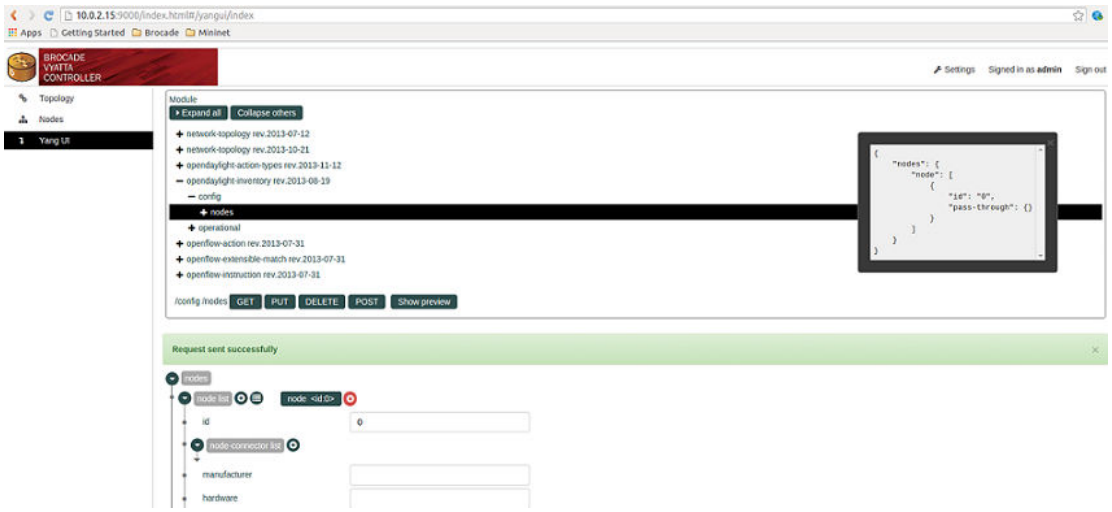
   The resulting page displays the first level in the data structure:

   - config —This is the configuration element. The controller stores the configuration of the network and devices (read and write). The user or the application provides configuration data.
   - operational —This is the operational element. The controller stores the operational state for the network and devices (read only). Operational data is the information that reflects the current state of the network.
2. Select and expand the config element.
   You see the first configurable resource, which is **nodes**.
3. Click **Nodes**. The page displays the xpath of the element at the bottom of the list. It also provides the REST API methods that apply to the element.
4. Expand **Nodes** and you see an {id} element node to display a field to input xpath values. The {id} is a configurable parameter that corresponds to input fields in the xpath. Enter appropriate values in the input fields to configure the element.
   The GET and DELETE operations do not require a request body, but the POST and PUT operations do. Every configurable resource has its own template with custom fields.
5. Click **Show Preview**, which follows the API tree, to display requests that are sent to the controller. When you enter data, the YANG UI displays a pane with text that contains the description of a request.

After you send the required method to the controller, it starts the REST request. Click **Show Preview** to see the response from the controller after you see this message: `Request sent successfully`.

Use this procedure for any configurable resource in the YANG model structure.

**FIGURE 4** YANG UI in Configuration Mode




# Configuring list elements on the YANG UI

The YANG UI displays a tree structure, which includes list elements, that can be expanded or collapsed.

The configuration tasks that you can perform by using the YANG UI follow.

1. To add a list element, click the plus button **+** after the list name. When a list element is added, an icon with the name of the list element and its key value are displayed.
2. To remove a list element, click the **X** button that follows the list element.

**NOTE**
A key value for a list is one or more inputs that you can use as identifiers for the list element. Keys in a list must be unique. If two or more elements have the same key value, the **!** warning icon is displayed near the node names.

3. To select an element, click the icon that is associated with the element. The icon is displayed only when the list contains at least one list element. The name of the list element and its neighbors are displayed as tabs in the row list.
4. To move forward or backward through the row, click the arrow button at the end of a row.

# 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 Vyatta 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 21 and cURL utility, refer to cURL utility on page 23.

## The controller YANG UI

The YANG UI application on the controller GUI enables you to access the controller configuration and the controller state by using YANG modules (RESTCONF). For more information, refer to Using the YANG UI on page 18.

## 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**.

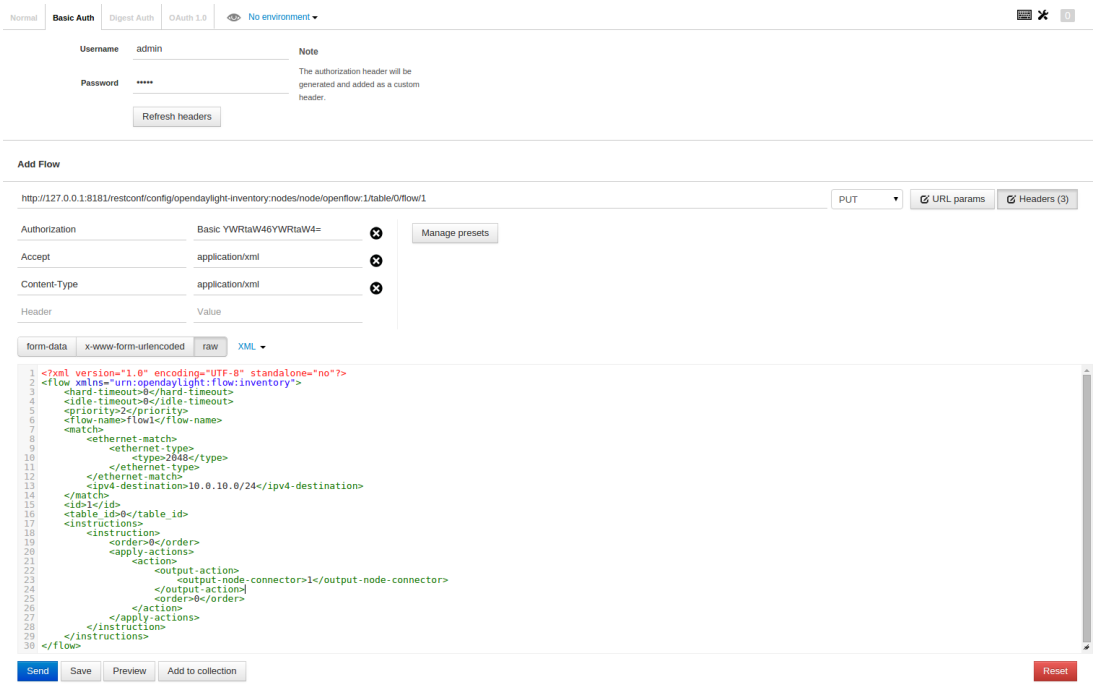
**NOTE**
You can import some of the examples into Postman by importing the following collection: https://www.getpostman.com/collections.

# 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 5** Install Flow

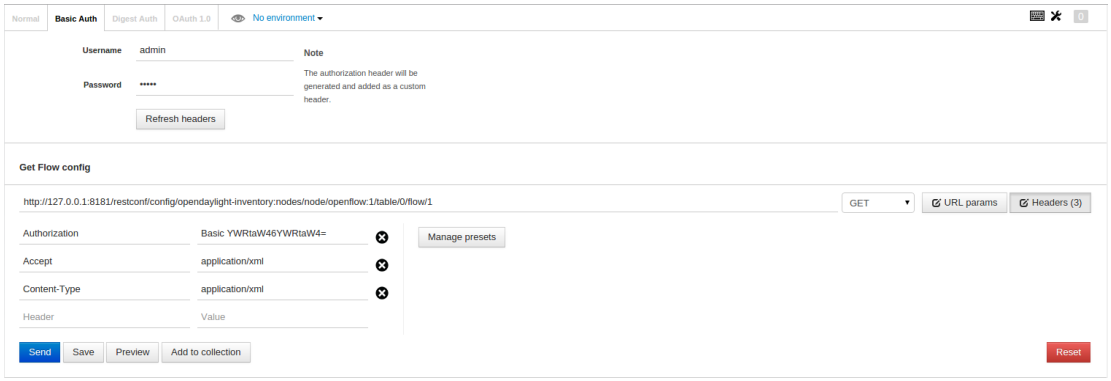



**NOTE**
Ensure that the table number and flow number in the URL and the XML body match.

## Example of retrieving the installed flow

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

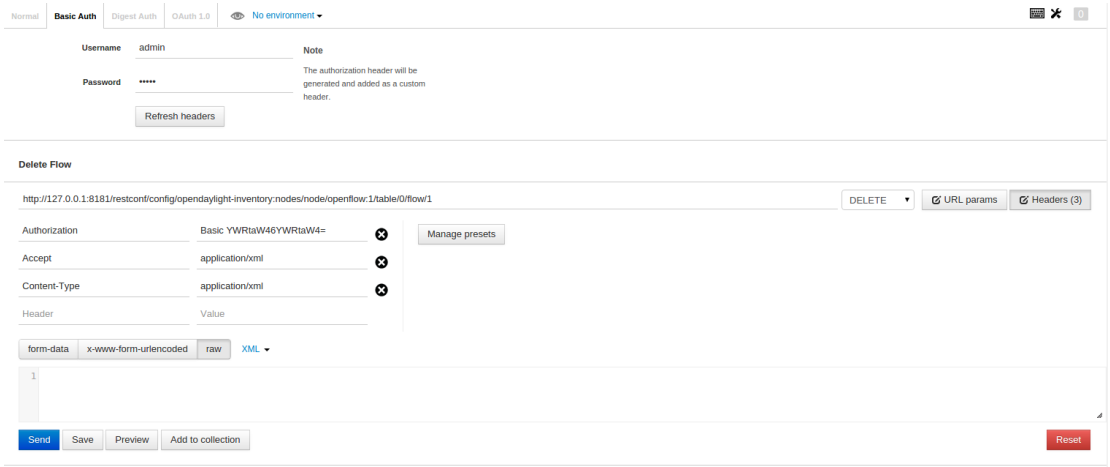
**FIGURE 6** Retrieving Flow Configuration




## 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 7** 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 Vyatta 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 Vyatta Controller Quick Start Guide* and install 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
   ```

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 51.

## 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 GUI

You can retrieve the topology data from the controller by using the GUI. For more information on how to see OpenFlow topology, refer to About the controller GUI on page 15.

## 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 21.

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 21.

## 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 37.

# 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 has been correctly programmed in the switch, issue the RESTCONF request as provided in Example of individual flow statistics on page 37.

## 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 40.

# 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 37.

## 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 40.

## 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 37.

# 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 51.

## Path Explorer—L3 flows

The Path Explorer application pushes automatic L3 flows to switches in a network. You can provide the source IP address, destination IP address, and switch information for the system to calculate the shortest path between two hosts through the provided switch and install a path between them. To access Path Explorer on the controller GUI, you must install the application separately. For more information on L3 flows by using Path Explorer, refer to *Brocade Vyatta Controller Path Explorer User Guide*.

# 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 GUI. For more information, refer to About the controller GUI on page 15.

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.

## Primary NETCONF components in the controller

The Brocade Vyatta 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/opendaylight-inventory:nodes/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>
```

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

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

  Change <controller-ip> to the name of the host or IP address on which the controller is running.
- 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>
  <username
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">user1
</username>
  <password
xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">passw
d</password>
</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 Vyatta 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.

- Active mode: This mode is the default.
- Passive mode: This mode generates no traffic flows.

**Active mode functioning**

In the Active 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, see the chapter on OpenFlow in this document.
- The controller tracks hosts attached to the network.
- Switches dispatch packets between hosts.

**NOTE**
Owing to the flooding of packets, Host Tracker in the Active 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 Passive mode.

**Passive mode functioning**

In the Passive mode, there is no communication between hosts. There are also no flows for traffic. Host Tracker installs ARP flows and LLDP flows in OpenFlow devices.

Host Tracker installs two kinds of flows in OpenFlow devices:

- LLDP flows for the controller topology discovery. For more information about topology discovery, see the chapter on OpenFlow in this document.
- 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: Active mode

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

The following installations are required:

- Controller

  For instructions on installing the controller, see *Brocade Vyatta Controller Quick Start 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>
```

# Host Tracker: switching to the Passive mode

You must install the controller to access the 54-arphandler.xml file. You must edit the 54-arphandler.xml file to enable the Passive mode of functioning of Host Tracker.

1. Enter the following command to stop the controller:
   `/opt/bvc/bin/stop`
2. Enter the following command to go to /opt/bvc/controller/etc/opendaylight/karaf:
   `cd /opt/bvc/controller/etc/opendaylight/karaf`

3. Enter the following command to open the configuration file for editing:
   `vi 54-arphandler.xml`
4. Set the **is-proactive-flood-mode** setting to `false`.
   Host Tracker is configured to function in the Passive mode.
5. Enter the following command to restart the controller:
   `/opt/bvc/bin/start`

## Verifying the functioning of Host Tracker in the Passive mode

The functioning of Host Tracker in the Passive mode differs from its functioning in the Active 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 Active mode of functioning of Host Tracker, pings succeed because of the flood flows; pings fail in the Passive mode as no traffic flows are configured.

# Clustering with the Brocade Vyatta Controller



## Clustering overview

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

The controller supports the clustering feature that allows one or more controller instances to share a common data store, which means that any instance can service a request.

Clustering offers the following advantages:

- High availability: As data is replicated, there is no loss of controller services as long as the majority of the controller instances in the cluster are available.

**NOTE**
Individual applications and plug-ins may differ in behavior. For more information, see the Known Issues section for Clustering in *Brocade Vyatta Controller Release Notes*.

- Data persistence: Application and plug-in data persists across controller restarts.
- Scaling out: The number of controller instances in a cluster can be increased to provide additional overall processing capacity.

## Setting up clustering

This section provides instructions on how to enable clustering on the controller.

You must install the controller before you set up clustering. For instructions on installing the controller, see *Brocade Vyatta Controller Quick Start Guide*.

1. Login to the controller host and enter the following command:
   `/opt/bvc/bin/client`
2. At the Karaf prompt, install the clustering feature by entering the following command:
   `feature:install odl-mdsal-clustering`
3. To exit the Karaf console, enter the following command:
   `logout`

# Setting up a three-node cluster

This section provides instructions on how to set up a three-node cluster with the controller for HA in your network.

1. Create three virtual machines (Ubuntu 14.04) to set up a three-node cluster. Each node is a controller instance in the cluster.
2. Install the controller on each node. For instructions on installing the controller, see *Brocade Vyatta Controller Quick Start Guide*.
3. On each node, enable clustering using the instructions in Setting up clustering on page 57.

**NOTE**
For the remaining steps in this section, it is assumed that the IP addresses of the three nodes are 10.0.0.1, 10.0.0.2, 10.0.0.3, respectively, and that the member IDs are member-1, member-2, and member-3.

4. Stop the controller instance on each node by entering the following command:
   `/opt/bvc/bin/stop`
5. 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.

```
Change
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 = 52428800
      }
to
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. For example, on the first node, change the following:

**NOTE**
There are two instances of seed-nodes in the akka.conf file. The configuration must be changed for both instances.

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

to

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 this 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. For example, on the second node, change the following:

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

to

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

6. Edit the module-shards.conf configuration file on each node. Add the role names of the other two nodes into 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 module-shards.conf file for each node should look like this:

```
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"
                         ]
                     }
                ]
        }
]
```

7. Start the controller instance on each node by entering the following command:

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

# Verifying the cluster setup

To verify that the nodes have been set up properly and have 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
```

where <role-name> is the role name of the node. For example, for the query to node 1, the command is the following:

```
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 should show the same value for Leader. For example:

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

# Troubleshooting



## Best practices for installing the Brocade Vyatta Controller

If you are having issues installing the controller, follow the best practices that are outlined in this section.

- The installation of the controller depends on several prerequisites. Ensure that you meet all the prerequisites and system requirements that are described in *Brocade Vyatta Controller Quick Start Guide*.
- We recommend that you have an Internet connection to run the installation procedure.
- We recommend that you have desktop access to the Ubuntu system on which you are going to install the controller. You must use the user interface to configure the controller.
- JAVA version must be 1.7. We recommend the latest patch or minor version of the Oracle JAVA JDK, which is 67 or later. Download the JAVA JDK from www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html. Java version 1.8 is not supported.

**NOTE**
Ensure that you select the Linux x64 - Compressed Binary file. The file has a tar.gz extension.

- If you are having issues understanding a specific command, enter the **man** command to bring up a detailed section on the command. For example, entering **man mv** in the terminal displays detailed information about the **mv** (move) command.
- 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), we recommend that you upgrade to the latest node.js version that is supported.

## Troubleshooting installation

If you get an error message about a missing prerequisite during installation, follow the instructions in *Brocade Vyatta Quick Start 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.

1. Start the controller.

To start the controller, refer to *Brocade Vyatta Controller Quick Start 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 Vyatta Controller Quick Start Guide*.
5. Re-enter the commands in steps 2 and 3 to check the consumption of system resources when the controller is not running.

## 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 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 bvc/logs/controller_logs`
The karaf.log file is located here.

# Glossary

| | |
|---|---|
| **Brocade Vyatta Controller Platform** | A server-and-user platform process that contains the functionality of the base controller.<br><br>For example: bvc-1.1.0.zip, bvc-dependencies-1.1.0.zip |
| **Brocade Vyatta Controller app** | An app that runs on the Brocade Vyatta Controller Platform and provides additional usage cases, features, or both.<br><br>For example: bvc-app-pathexplorer-packaging-1.1.0.zip, bvc-app-vyattaems-packaging-1.1.0.zip |
| **Brocade Vyatta Controller extension** | An extension that modifies or configures the Brocade Vyatta Controller Platform or app. Typically, an extension is bundled with the Brocade Vyatta Controller Platform, a Brocade Vyatta Controller app, or another Brocade product.<br><br>For example: bvc-ext-l2switch-noflood-packaging-1.0.0.zip |