Engineering
Simplicity

# Junos® OS

# Puppet for Junos OS Administration Guide

Published
2021-01-08

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

## YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

## DISCLAIMER

# Table of Contents

# About the Documentation

**IN THIS SECTION**

Use this guide to automate the configuration management of devices running Junos OS with Puppet software.

## Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at https://www.juniper.net/documentation/.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at https://www.juniper.net/books.

## Documentation Conventions

Table 1 on page vi defines notice icons used in this guide.

Table 1: Notice Icons

| Icon | Meaning | Description |
|---|---|---|
| ⓘ | Informational note | Indicates important features or instructions. |
| ⚠ | Caution | Indicates a situation that might result in loss of data or hardware damage. |
| ⚡ | Warning | Alerts you to the risk of personal injury or death. |
| ✳ | Laser warning | Alerts you to the risk of personal injury from a laser. |
| 💡 | Tip | Indicates helpful information. |
| 🏅 | Best practice | Alerts you to a recommended use or implementation. |

Table 2 on page vi defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

| Convention | Description | Examples |
|---|---|---|
| **Bold text like this** | Represents text that you type. | To enter configuration mode, type the **configure** command:<br><br>user@host> **configure** |
| `Fixed-width text like this` | Represents output that appears on the terminal screen. | `user@host>` **show chassis alarms**<br><br>`No alarms currently active` |
| *Italic text like this* | • Introduces or emphasizes important new terms.<br>• Identifies guide names.<br>• Identifies RFC and Internet draft titles. | • A policy *term* is a named structure that defines match conditions and actions.<br>• *Junos OS CLI User Guide*<br>• RFC 1997, *BGP Communities Attribute* |

Table 2: Text and Syntax Conventions *(continued)*

| Convention | Description | Examples |
|---|---|---|
| *Italic text like this* | Represents variables (options for which you substitute a value) in commands or configuration statements. | Configure the machine's domain name:<br><br>[edit]<br>root@# **set system domain-name** *domain-name* |
| **Text like this** | Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components. | • To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level.<br>• The console port is labeled **CONSOLE**. |
| < > (angle brackets) | Encloses optional keywords or variables. | **stub <default-metric *metric*>;** |
| \| (pipe symbol) | Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity. | **broadcast \| multicast**<br><br>**(*string1* \| *string2* \| *string3*)** |
| # (pound sign) | Indicates a comment specified on the same line as the configuration statement to which it applies. | **rsvp { # Required for dynamic MPLS only** |
| [ ] (square brackets) | Encloses a variable for which you can substitute one or more values. | **community name members [ *community-ids* ]** |
| Indention and braces ( { } ) | Identifies a level in the configuration hierarchy. | [edit]<br>routing-options {<br>  static {<br>    route default {<br>      nexthop *address*;<br>      retain;<br>    }<br>  }<br>} |
| ; (semicolon) | Identifies a leaf statement at a configuration hierarchy level. | |
| **GUI Conventions** | | |

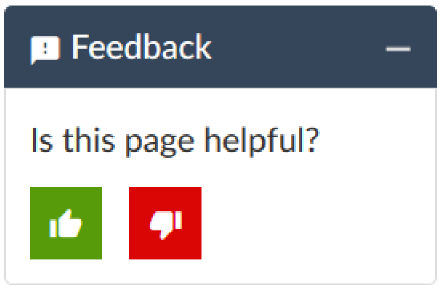
Table 2: Text and Syntax Conventions *(continued)*

| Convention | Description | Examples |
|---|---|---|
| **Bold text like this** | Represents graphical user interface (GUI) items you click or select. | • In the Logical Interfaces box, select **All Interfaces**.<br>• To cancel the configuration, click **Cancel**. |
| **>** (bold right angle bracket) | Separates levels in a hierarchy of menu selections. | In the configuration editor hierarchy, select **Protocols>Ospf**. |

## Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation. You can use either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the Juniper Networks TechLibrary site, and do one of the following:



  - Click the thumbs-up icon if the information on the page was helpful to you.
  - Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.

- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

## Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active Juniper Care or Partner Support Services support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at https://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf.
- Product warranties—For product warranty information, visit https://www.juniper.net/support/warranty/.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

## Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: https://www.juniper.net/customers/support/
- Search for known bugs: https://prsearch.juniper.net/
- Find product documentation: https://www.juniper.net/documentation/
- Find solutions and answer questions using our Knowledge Base: https://kb.juniper.net/
- Download the latest versions of software and review release notes: https://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: https://kb.juniper.net/InfoCenter/
- Join and participate in the Juniper Networks Community Forum: https://www.juniper.net/company/communities/
- Create a service request online: https://myjuniper.juniper.net

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://entitlementsearch.juniper.net/entitlementsearch/

## Creating a Service Request with JTAC

You can create a service request with JTAC on the Web or by telephone.

- Visit https://myjuniper.juniper.net.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see https://support.juniper.net/support/requesting-support/.

1

CHAPTER

# Disclaimer

# Puppet for Junos OS Disclaimer

2

CHAPTER

# Puppet for Junos OS Overview

# Understanding Puppet for Junos OS

IN THIS SECTION

## Puppet for Junos OS Overview

Puppet is configuration management software that is developed by Puppet. Puppet provides an efficient and scalable solution for managing the configurations of large numbers of devices. System administrators use Puppet to manage the configurations of physical and virtual servers and network devices. Juniper Networks provides support for using Puppet to manage certain devices running the Junos® operating system (Junos OS).

You typically deploy the Puppet software using a client-server arrangement, where the server, or Puppet master, manages one or more agent nodes. The client daemon, or Puppet agent, runs on each of the managed nodes. You create Puppet manifest files to describe your desired system configuration. The Puppet master compiles the manifests into catalogs, and the Puppet agent periodically retrieves the catalog and applies the necessary changes to the configuration.

Table 3 on page 13 describes the Puppet for Junos OS support components, and Figure 1 on page 14 illustrates the interaction of the components.

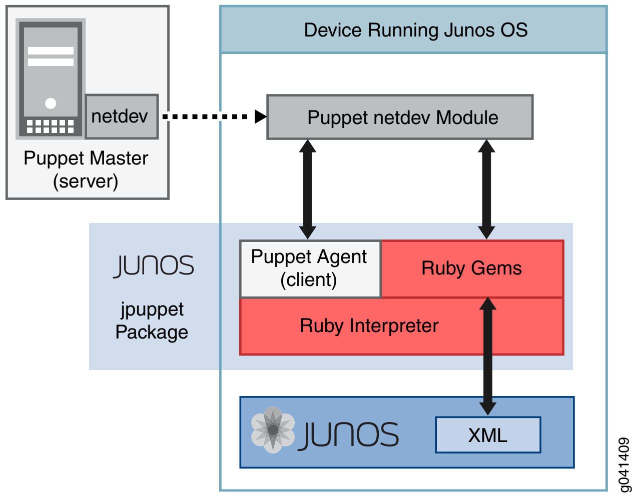
Table 3: Puppet for Junos OS Components

| Component | Description |
|---|---|
| **jpuppet** package<br><br>or<br><br>**juniper/puppet-agent** Docker container | Package or container that is installed on the agent node running Junos OS and that contains the Puppet agent, the Ruby programming language, and support libraries.<br><br>Certain devices running Junos OS have the Puppet agent integrated into the software image and do not require installing a separate package. |
| **netdevops/netdev_stdlib** Puppet module | Module that contains generic Puppet type definitions. It does not include any specific provider code. |

Table 3: Puppet for Junos OS Components *(continued)*

| Component | Description |
|---|---|
| **juniper/netdev_stdlib_junos** Puppet module | Module that contains the Junos OS-specific Puppet provider code that implements the types defined in the **netdevops/netdev_stdlib** module. You install this module on the Puppet master when managing devices running Junos OS. |
| Ruby gem for NETCONF (Junos XML API) | Gem that is installed on the Puppet master and is also bundled in the **jpuppet** package. |

Figure 1: Puppet Components for Managing Devices Running Junos OS



The **netdev_stdlib** Puppet module provides Puppet resource types for configuring:

- Physical interfaces
- Layer 2 switch ports
- VLANs
- Link aggregation groups

The Juniper Networks **netdev_stdlib_junos** module contains the Junos OS-specific Puppet provider code that implements the resource types defined in the **netdev_stdlib** module. You install the **netdev_stdlib_junos** module on the Puppet master to manage devices running Junos OS. Starting with **netdev_stdlib_junos** module version 2.0.2, the module also provides the **apply_group** defined resource type, which enables you to manage network resources that do not have type specifications in the **netdev_stdlib** module.

When using Puppet to manage devices running Junos OS, the Puppet agent makes configuration changes under exclusive lock and logs all commit operations with a Puppet catalog version for audit tracking. Puppet report logs include a Junos OS source indicator for log entries specific to Junos OS processing and tags associated with the operation or error, which enables easy report extraction.

For more information about Puppet, see the Puppet website at https://puppet.com.

## Benefits of Puppet and Puppet for Junos OS

- Provide an efficient and scalable software solution for managing the configurations of large numbers of devices
- Enable automatic enforcement of the correct state of a device
- Increase operational efficiency by automating configuration management tasks and reducing the manual configuration and management of devices
- Lower the risk and cost of service outages by reducing configuration errors
- Improve change management processes and provide transparency by logging commit operations with a Puppet catalog version for audit tracking purposes
- Enable organizations that already use Puppet to manage server resources to extend this to network devices

RELATED DOCUMENTATION

Installing Puppet for Junos OS | **23**

Puppet Manifests for Devices Running Junos OS | **37**

Puppet netdev Resources | **47**

# Puppet for Junos OS Supported Platforms

Puppet for Junos OS should only be used with the devices running the Junos OS release and **jpuppet** package specified in Table 4 on page 16. You must download the **jpuppet** package from the download folder that has the same release number as the Puppet for Junos OS release listed in the table. The version of the **netdev_stdlib_junos** module installed on the Puppet master determines which devices the Puppet master can control.

Certain devices do not require the **jpuppet** package, because the Puppet agent is either integrated into the software image or it can be run as a Docker container. Devices running Junos OS Evolved that support running the Puppet agent as a Docker container can use the Juniper Networks juniper/puppet-agent Docker container as an alternative to using the Puppet agent that is integrated with the software image.

**Table 4: Puppet for Junos OS Supported Devices and Junos OS Releases**

| Device | Junos OS Release | Puppet for Junos OS Release | jpuppet Package | Support for agent as Docker container | Compatible Versions of netdev_stdlib_junos |
|---|---|---|---|---|---|
| EX4200<br>EX4500<br>EX4550 | 12.3R2 or a later 12.3 release | 1.0 | **jpuppet-ex-1.0R1.*n*.tgz** | – | 1.0.0 |
| EX4300 (standalone and Virtual Chassis) | 14.1X53-D10 or a later 14.1X53 release | 2.0 | **jpuppet-powerpc-3.6.1_2.*n*.tgz** | – | 1.0.2<br>2.*x*.*y* |
| EX4650-48Y | 18.3R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.*x*.*y* |
| | 18.3R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.*x*.*y* |
| EX9200-15C | 20.3R1 | 4.0 | – | – | 2.1.0 or later |
| MX5<br>MX10<br>MX40 | 12.3R2 or a later 12.3 release | 1.0 | **jpuppet-mx80-1.0R1.*n*.tgz** | – | 1.0.0 |
| | 14.2R2 or a later 14.2 release<br>15.1R1 or a later 15.1 release | 2.0 | **jpuppet-powerpc-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| MX80 | 12.3R2 or a later 12.3 release | 1.0 | **jpuppet-mx80-1.0R1.*n*.tgz** | – | 1.0.0 |
| | 14.2R2 or a later 14.2 release<br>15.1R1 or a later 15.1 release | 2.0 | **jpuppet-powerpc-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 16.1R1 or later | 3.0 | **jpuppet-powerpc-3.6.1_3.*n*.tgz** | – | 2.*x*.*y* |

Table 4: Puppet for Junos OS Supported Devices and Junos OS Releases *(continued)*

| Device | Junos OS Release | Puppet for Junos OS Release | jpuppet Package | Support for agent as Docker container | Compatible Versions of netdev_stdlib_junos |
|---|---|---|---|---|---|
| MX104 | 14.2R2 or a later 14.2 release<br>15.1R1 or a later 15.1 release | 2.0 | **jpuppet-powerpc-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 16.1R1 or later | 3.0 | **jpuppet-powerpc-3.6.1_3.*n*.tgz** | – | 2.*x*.*y* |
| MX240<br>MX480<br>MX960 | 12.3R2 or a later 12.3 release | 1.0 | **jpuppet-mx-1.0R1.*n*.tgz** | – | 1.0.0 |
| | 14.2R2 or a later 14.2 release | 2.0 | **jpuppet-i386-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 16.1R1 through 18.1 | 3.0 | **jpuppet-x86-32-3.6.1_3.*n*.tgz** | – | 2.*x*.*y* |
| | 18.2R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.*x*.*y* |
| OCX1100 | 14.1X53-D20 or a later 14.1X53 release | 2.0 | – | – | 1.0.2<br>2.*x*.*y* |
| PTX10001-36MR | 20.2R1 or later | – | – | Y | 2.1.0 or later |
| PTX10003-80C<br>PTX10003-160C | 19.1R1 or later | – | – | – | 2.0.3 or later |
| | 20.1R1 or later | – | – | Y | 2.1.0 or later |
| PTX10004 | 20.3R1 or later | – | – | Y | 2.1.0 or later |
| PTX10008 | 20.1R1 or later (Junos OS Evolved only) | – | – | – | 2.1.0 or later |
| QFX3500<br>QFX3600 | 12.3X50-D20 or a later 12.3X50 release | 1.0 | **jpuppet-qfx-1.0R1.*n*.tgz** | – | 1.0.0 |

**Table 4: Puppet for Junos OS Supported Devices and Junos OS Releases *(continued)***

| Device | Junos OS Release | Puppet for Junos OS Release | jpuppet Package | Support for agent as Docker container | Compatible Versions of netdev_stdlib_junos |
|---|---|---|---|---|---|
| QFX5100 (standalone) | 13.2X51-D15 with enhanced automation | 1.0 | – | – | 1.0.0 |
| | 14.1X53-D10 with enhanced automation or a later 14.1X53 release with enhanced automation | 2.0 | – | – | 1.0.2<br>2.*x*.*y* |
| QFX5120-48T | 20.2R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.1.0 or later |
| | 20.2R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.1.0 or later |
| QFX5120-48Y | 18.3R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.*x*.*y* |
| | 18.3R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.*x*.*y* |
| QFX5120-48YM | 20.4R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.1.0 or later |
| | 20.4R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.1.0 or later |
| QFX5220-32CD | 19.1R2 or later | – | – | – | 2.0.3 or later |
| | 20.1R1 or later | – | – | Y | 2.1.0 or later |
| QFX5220-128C | 19.2R1 or later | – | – | - | 2.0.3 or later |
| | 20.1R1 or later | – | – | Y | 2.1.0 or later |

**Table 4: Puppet for Junos OS Supported Devices and Junos OS Releases *(continued)***

| Device | Junos OS Release | Puppet for Junos OS Release | jpuppet Package | Support for agent as Docker container | Compatible Versions of netdev_stdlib_junos |
|---|---|---|---|---|---|
| QFX10002 QFX10008 | 15.1X53-D30 or a later 15.1X53-D3*x* release | 2.0 | **jpuppet-i386-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 15.1X53-D30 with enhanced automation or a later 15.1X53 release with enhanced automation | 2.0 | – | – | 2.*x*.*y* |
| | 15.1X53-D60 or a later 15.1X53 release | 2.0 | **jpuppet-x86-32-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 17.1R2 through 18.1 | 3.0 | **jpuppet-x86-32-3.6.1_3.*n*.tgz** | – | 2.*x*.*y* |
| | 17.1R2 through 18.1 with enhanced automation | 3.0 | – | – | 2.*x*.*y* |
| | 18.2R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.*x*.*y* |
| | 18.2R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.*x*.*y* |

**Table 4: Puppet for Junos OS Supported Devices and Junos OS Releases *(continued)***

| Device | Junos OS Release | Puppet for Junos OS Release | jpuppet Package | Support for agent as Docker container | Compatible Versions of netdev_stdlib_junos |
|---|---|---|---|---|---|
| QFX10016 | 15.1X53-D60 or a later 15.1X53 release | 2.0 | **jpuppet-x86-32-3.6.1_2.*n*.tgz** | – | 2.*x*.*y* |
| | 15.1X53-D60 with enhanced automation or a later 15.1X53 release with enhanced automation | 2.0 | – | – | 2.*x*.*y* |
| | 17.1R2 through 18.1 | 3.0 | **jpuppet-x86-32-3.6.1_3.*n*.tgz** | – | 2.*x*.*y* |
| | 17.1R2 through 18.1 with enhanced automation | 3.0 | – | – | 2.*x*.*y* |
| | 18.2R1 or later | 4.0 | **jpuppet-x86-32-3.6.1_4.*n*.tgz** | – | 2.*x*.*y* |
| | 18.2R1 with enhanced automation or a later release with enhanced automation | 4.0 | – | – | 2.*x*.*y* |

Table 5 on page 20 describes the naming conventions for the **jpuppet** package in different Puppet for Junos OS releases. In Release 1.0 of Puppet for Junos OS, **jpuppet** packages are specific to a particular platform. In later releases, the packages are only specific to the device architecture.

**Table 5: jpuppet Package Naming Conventions**

| Puppet for Junos OS Release | Package Naming Convention |
|---|---|
| 1.0 | **jpuppet-*platform*-*m*.0R1.*n*.tgz** |
| 2.0<br>3.0<br>4.0 | **jpuppet-*architecture*-*puppet_m*.*n*.tgz** |

Where:

***architecture***—Device architecture, for example: powerpc, i386, or x86-32.

***m.n***—Puppet for Junos OS release, where *m* represents the major release number, and *n* represents the minor release number.

***platform***—Platform series, for example, **mx**.

***puppet***—Puppet version, for example, 3.6.1.

3

CHAPTER

# Installing Puppet for Junos OS

# Installing Puppet for Juniper OS

IN THIS SECTION

## Setting Up the Puppet Master

Juniper Networks provides support for using Puppet to manage certain devices running Junos OS. The Puppet master must be running Puppet open-source edition. Table 6 on page 23 outlines the version of Puppet that must be installed on the Puppet master in order to manage the different Junos OS variants and releases of Puppet for Junos OS on the client.

**Table 6: Puppet Version Required on Puppet Master**

| Junos OS Variant | Puppet for Junos OS Version | Puppet Version |
|---|---|---|
| Junos OS or Junos OS with Enhanced Automation | 1.0 | Puppet 2.7.19 or later |
| | 2.0<br>3.0<br>4.0 | Puppet 3.6.1 or later |
| Junos OS Evolved | – | Puppet 3.8.7 or later |

The Puppet master must also have the following software installed in order to use Puppet to manage devices running Junos OS:

- Juniper Networks NETCONF Ruby gem—Ruby gem that enables device management using the NETCONF protocol.
- **netdevops/netdev_stdlib** Puppet module—includes the Puppet type definitions for the netdev resources.
- **juniper/netdev_stdlib_junos** Puppet module—includes the Junos OS-specific code that implements each of the types. When you install this module on the Puppet master, it automatically installs the **netdev_stdlib** module.

To configure the Puppet master for use with devices running Junos OS:

1. Install Puppet open-source edition.

   See the Puppet website for Puppet installation instructions.

2. Install the Juniper Networks NETCONF Ruby gem using the command appropriate for your Puppet master installation.

   `root@server:~#` **gem install netconf**

   ```
   Fetching: netconf-0.2.5.gem (100%)
   Successfully installed netconf-0.2.5
   1 gem installed
   Installing ri documentation for netconf-0.2.5...
   Installing RDoc documentation for netconf-0.2.5...
   ```

3. Install or upgrade the Juniper Networks **netdev_stdlib_junos** Puppet module.

   - To install the **netdev_stdlib_junos** module, execute the following command on the Puppet master, and specify the module version required to manage your particular devices.

     `root@server:~#` **puppet module install juniper-netdev_stdlib_junos --version 2.0.6**

     ```
     Notice: Preparing to install into
     /etc/puppetlabs/code/environments/production/modules ...
     Notice: Downloading from https://forgeapi.puppet.com ...
     Notice: Installing -- do not interrupt ...
     /etc/puppetlabs/code/environments/production/modules
         juniper-netdev_stdlib_junos (v2.0.6)
           netdevops-netdev_stdlib (v1.0.0)
     ```

   - To upgrade the module when you have an older version installed, use the **upgrade** option.

     `root@server:~#` **puppet module upgrade juniper-netdev_stdlib_junos --version 2.0.6**

4. Set up the **puppet.conf** file on the Puppet master.

   For information about the configuration file, see “Setting Up the Puppet Configuration File on the Puppet Master and Puppet Agents Running Junos OS” on page 32.

**NOTE:** The Puppet agent identifies with the Puppet master using SSL. By default, the puppet master service does not sign client certificate requests. As a result, the Puppet master must approve the agent certificate the first time an agent tries to connect to the master. After the Puppet agent node is configured and running, approve the client certificate on the Puppet master by using the command appropriate for your installation, for example, by using the **puppet cert sign *host*** command or the **puppetserver ca sign --certname *host*** command.

## Configuring the Puppet Agent Node

Juniper Networks provides support for using Puppet to manage certain devices running Junos OS. The setup on the agent node depends on the device and the Junos OS variant running on the device. Certain devices require installing the Puppet agent package on the device, other devices have the Puppet agent integrated into the software image, and some devices support running the Puppet agent as a Docker container. To verify support for a specific platform and determine which setup to use for a given device and Junos OS release, see Puppet for Junos OS Supported Platforms.

Table 7 on page 25 outlines the tasks required to configure the Puppet agent node for the different types of setups. To configure the node, perform the steps in each linked task.

**Table 7: Puppet Agent Setup**

| Puppet Agent Setup | Tasks |
|---|---|
| Puppet agent must be installed using the **jpuppet** package | Perform the steps in the following tasks:<br>1. Installing the Puppet Agent Package on page 26<br>2. Configuring the Junos OS User Account on page 27<br>3. Configuring the Environment Settings on page 28 |
| Puppet agent is integrated on the device | Perform the steps in the following tasks:<br>1. Configuring the Junos OS User Account on page 27<br>2. Configuring the Environment Settings on page 28<br>3. Starting the Puppet Agent Process on page 29 |
| Puppet agent will run as a Docker container | Perform the steps in the following tasks:<br>1. Configuring the Junos OS User Account on page 27<br>2. Using the Puppet Agent Docker Container on page 30 |

**NOTE:** OCX1100 switches, QFX Series switches running Junos OS with Enhanced Automation, and devices running Junos OS Evolved have the Puppet agent integrated with the software. If the device also supports using the Puppet agent Docker container, you can elect to run the Puppet agent as a Docker container instead of using the integrated Puppet agent.

### Installing the Puppet Agent Package

To install the Puppet agent on devices running Junos OS that do not have the agent integrated into the software:

1. Determine the **jpuppet** software package required for your platform and release at Puppet for Junos OS Supported Platforms.

2. Access the download page at https://github.com/Juniper/jpuppet-download.

3. Select the release folder corresponding to the Puppet for Junos OS release to download.

4. Download to the **/var/tmp/** directory on the agent device the **jpuppet** software package that is specific to your platform or device microprocessor architecture, depending on the Puppet for Junos OS release.

   **NOTE:** Starting in Puppet for Junos OS Release 2.0, the **jpuppet** packages are specific to the microprocessor architecture. In earlier releases, the packages are specific to a particular platform. If you do not know the microprocessor architecture of your device, you can use the UNIX shell command uname -a to determine it.

   **NOTE:** We recommend that you install the **jpuppet** software package from the **/var/tmp/** directory on your device to ensure the maximum amount of disk space and RAM for the installation.

5. Configure the provider name, license type, and deployment scope associated with the application.

```
[edit]
user@host# set system extensions providers juniper license-type juniper deployment-scope commercial
user@host# commit and-quit
```

6. Install the software package using the **request system software add** operational mode command, and include the **no-validate** option.

   ```
   user@host> request system software add /var/tmp/jpuppet-package-name no-validate
   ```

7. Verify that the installation is successful by issuing the **show version** command.

   The list of installed software should include the **jpuppet** package. For example:

   `admin@jd>` **show version**

   ```
   Hostname: jd
   Model: mx80-48t
   Junos: 16.1R1.7
   JUNOS Base OS boot [16.1R1.7]
   JUNOS Base OS Software Suite [16.1R1.7]
   JUNOS Crypto Software Suite [16.1R1.7]
   JUNOS Packet Forwarding Engine Support (MX80) [16.1R1.7]
   JUNOS Web Management [16.1R1.7]
   JUNOS Online Documentation [16.1R1.7]
   JUNOS Services Application Level Gateways [16.1R1.7]
   JUNOS Services Jflow Container package [16.1R1.7]
   JUNOS Services Stateful Firewall [16.1R1.7]
   JUNOS Services NAT [16.1R1.7]
   JUNOS Services RPM [16.1R1.7]
   JUNOS Macsec Software Suite [16.1R1.7]
   JUNOS Services Crypto [16.1R1.7]
   JUNOS Services IPSec [16.1R1.7]
   JUNOS py-base-powerpc [16.1R1.7]
   JUNOS py-extensions-powerpc [16.1R1.7]
   JUNOS Kernel Software Suite [16.1R1.7]
   JUNOS Routing Software Suite [16.1R1.7]
   JET app jpuppet [3.6.1_3.0]
   ```

   **NOTE:** The package name might vary depending on the Puppet for Junos OS release.

## Configuring the Junos OS User Account

You must configure a user account to run the Puppet agent. The user must have configure, control, and view permissions. You can configure any username and authentication method for the account.

To configure a Junos OS user account to run the Puppet agent:

1. Configure the account username, login class, authentication method, and shell.

   ```
   [edit]
   user@host# set system login user puppet class class
   user@host# set system login user puppet authentication authentication-options
   user@host# set system login user puppet shell csh
   ```

2. Commit the configuration.

   ```
   [edit]
   user@host# commit and-quit
   ```

## Configuring the Environment Settings

Set up the directory structure and environment settings on any agent nodes on which you installed the Puppet agent package or that use the Puppet agent that is integrated with the software image.

To configure the necessary directory structure and environment settings to run the Puppet agent:

1. Log in to the agent node using the Puppet account username and password.

2. If you are not already in the UNIX-level shell, enter the shell.

   ```
   user@host> start shell
   ```

3. Create a **$HOME/.cshrc** file, and include the content corresponding to the variant of Junos OS and the release of Puppet for Junos OS installed on the device, which is outlined in Table 8 on page 28.

   **Table 8: Content in Puppet Agent .cshrc File**

   | Junos OS Variant | Puppet for Junos OS Release | **.cshrc** content |
   |---|---|---|
   | Junos OS or Junos OS with Enhanced Automation | 1.0 or 2.0 | **setenv PATH ${PATH}:/opt/sdk/juniper/bin** |
   | | 3.0 or 4.0 | **setenv PATH ${PATH}:/opt/jet/juniper/bin** |
   | Junos OS Evolved | – | **setenv PATH ${PATH}:/usr/bin** |

4. Exit the device and log back in using the Puppet account username and password.

5. If you are not already in the UNIX-level shell, enter the shell.

```
user@host> start shell
```

6. Verify that the **jpuppet** code is installed and that the PATH variable is correct by running Facter, which should display device-specific information. For example:

   % **facter**

```
architecture => mx80-48t
domain => example.com
facterversion => 2.0.1
fqdn => jd.example.com
hardwareisa => powerpc
hardwaremodel => mx80-48t
hostname => jd
id => puppet
ipaddress => 198.51.100.1
kernel => JUNOS
<...more...>
```

7. Create the following **$HOME/.puppet** directory structure:

```
% mkdir -p $HOME/.puppet/var/run
% mkdir -p $HOME/.puppet/var/log
```

8. Place your **puppet.conf** file in the **$HOME/.puppet** directory.

   For information about the configuration file, see "Setting Up the Puppet Configuration File on the Puppet Master and Puppet Agents Running Junos OS" on page 32.

## Starting the Puppet Agent Process

Devices that have the Puppet agent integrated into the software require that you start the Puppet agent process on the device. Start the Puppet agent process after configuring the Junos OS user account and environment settings.

To start the Puppet agent process:

1. Enter the shell.

```
user@host> start shell
```

2. Start the Puppet agent process by executing the **puppet agent** command, and include any desired options.

   - For example, on devices running Junos OS or Junos OS with Enhanced Automation:

     ```
     % puppet agent --server servername --waitforcert 60 --test
     ```

   - On devices running Junos OS Evolved, switch to the default VRF for management traffic, vrf0, and then start the agent.

     ```
     [vrf:none] user@host:~# switchvrf $$ vrf0
     [vrf:vrf0] user@host:~# puppet agent --test
     ```

     **NOTE:** You can choose to define the server settings in your Puppet configuration file instead of specifying the settings as command options.

## Using the Puppet Agent Docker Container

Certain devices running Junos OS Evolved support running the Puppet agent as a Docker container. Docker is a software container platform that is used to package and run an application and its dependencies in an isolated container. Juniper Networks provides a Docker image for the Puppet agent on Docker Hub.

When you run the Puppet agent using the Docker container, the container:

- Shares the hostname and network namespace of the host
- Uses the host network to communicate with the Puppet server
- Authenticates to the host using key-based SSH authentication

To use the Puppet agent Docker container on supported devices:

1. Log in as the root user.

2. Switch to the default VRF for management traffic, vrf0.

   ```
   [vrf:none] root@host:~# switchvrf $$ vrf0
   ```

3. Start the Docker service, and bind it to the default VRF for management traffic, vrf0.

   ```
   [vrf:none] root@host:~# systemctl start docker@vrf0
   ```

4. Set the **DOCKER_HOST** environment variable.

```
[vrf:none] root@host:~# export DOCKER_HOST=unix:///run/docker-vrf0.sock
```

5. Start the Puppet agent Docker container as follows, and set the **NETCONF_USER** to the Junos OS user account that was set up to run the agent.

```
[vrf:none] root@host:~# docker run -d -e PATH="/usr/local/bundle/bin:$PATH" -e
NETCONF_USER=puppet --network=host --name=puppet-agent juniper/puppet-agent:latest
```

6. Generate the SSH key pair that will be used to authenticate the container to the host.

```
[vrf:none] root@host:~# docker exec -it puppet-agent ssh-keygen -t rsa -N "" -f /root/.ssh/id_rsa
```

```
Generating public/private rsa key pair.
Created directory '/root/.ssh'.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
aa:69:77:b0:47:b0:c4:8f:90:39:f7:0d:04:61:ca:d1 root@host
The key's randomart image is:
+---[RSA 2048]----+
...
```

7. Copy the public key to the host, and add it to the root user's **authorized_keys** file.

```
[vrf:none] root@host:~# docker cp puppet-agent:/root/.ssh/id_rsa.pub .
[vrf:none] root@host:~# cat id_rsa.pub >> .ssh/authorized_keys
```

8. Verify the connection from the container to the host.

```
[vrf:none] root@host:~# docker exec -it puppet-agent ssh puppet@localhost
```

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is 3c:3c:ed:5c:ce:ee:34:09:79:22:d3:cd:af:d0:68:4a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
--- JUNOS 20.1-20200115.0-EVO Linux (none) 4.8.28-WR2.2.1_standard #1 SMP PREEMPT
 Thu Jun 13 00:19:16 PDT 2019 x86_64 x86_64 x86_64 GNU/Linux
[vrf:none] puppet@host:~#
```

9. Place your **puppet.conf** file in the container's **/etc/puppet** directory.

```
[vrf:none] root@host:~# docker cp /var/tmp/puppet.conf puppet-agent:/etc/puppet
```

**NOTE:** For information about the configuration file, see "Setting Up the Puppet Configuration File on the Puppet Master and Puppet Agents Running Junos OS" on page 32.

10. Start the Puppet agent.

```
[vrf:none] root@host:~# docker exec -it puppet-agent puppet agent -t
```

11. On the Puppet master, accept the agent's keys using the command appropriate for your installation.

```
root@server:~# puppet cert sign host.example.com
```

## Setting Up the Puppet Configuration File on the Puppet Master and Puppet Agents Running Junos OS

The Puppet configuration file, **puppet.conf**, defines the settings for the Puppet master and agent nodes. It is an INI-formatted file with code blocks that contain indented setting = value statements. The main code blocks are:

- **[master]**—settings for the Puppet master.
- **[agent]**—settings for the agent node.
- **[main]**—global settings that are used by all commands and services. The settings in the **[master]** and **[agent]** blocks override those in **[main]**.

On the Puppet master, the configuration file resides at **$confdir/puppet.conf**. On agent nodes running Junos OS, the location depends on your setup. Table 9 on page 32 outlines the location where the Puppet configuration file should reside for a given setup on devices running Junos OS.

**Table 9: Puppet Configuration File Location**

| Puppet agent setup | **puppet.conf** location |
|---|---|
| Puppet agent is installed using the **jpuppet** package | **$HOME/.puppet** directory for the Junos OS user account set up to run the Puppet agent |
| Puppet agent is integrated on the device | **$HOME/.puppet** directory for the Junos OS user account set up to run the Puppet agent |
| Puppet agent is running as a Docker container | **/etc/puppet** directory within the container |

Creating environment-specific Puppet configuration files is beyond the scope of this document. However, when using Puppet to manage devices running Junos OS, the Puppet master and agent node **puppet.conf** files must contain the following statement within the **[main]** configuration block:

```
[main]
    pluginsync = true
```

In addition, client devices running Junos OS Evolved must include the **certname** statement in the **puppet.conf** file and specify the node’s certificate name. The Puppet master uses the certificate name, which can be a hostname, an IP address, or any user-defined name in lowercase characters, to identify the client.

```
[main]
    certname = puppet-client
    pluginsync = true
```

The following example shows a sample **puppet.conf** file for an agent node running Junos OS:

```
[main]
    libdir = $vardir/lib
    logdir = $vardir/log/puppet
    rundir = $vardir/run/puppet
    ssldir = $vardir/ssl
    moduledir = $libdir
    factpath = $libdir/facter
    pluginsync = true

[agent]
    server = puppetmaster.example.com
    classfile = $vardir/classes.txt
    localconfig = $vardir/localconfig
```

The following example shows a sample **puppet.conf** file for an agent node running Junos OS Evolved:

```
[main]
    libdir = $vardir/lib
    logdir = $vardir/log/puppet
    rundir = $vardir/run/puppet
    ssldir = $vardir/ssl
    moduledir = $libdir
    factpath = $libdir/facter
    certname = agent01.example.com
    pluginsync = true
```

```
[agent]
    server = puppetmaster.example.com
    classfile = $vardir/classes.txt
    localconfig = $vardir/localconfig
```

For more information about Puppet configuration files, see the Puppet website at https://puppet.com/.

## Configuring the Puppet for Junos OS Addressable Memory

On devices running Junos OS, the amount of memory available to Puppet is 64 MB by default. You can expand the usable memory to the system maximum values as defined in Table 10 on page 34.

**Table 10: Puppet Agent Execution Environment Memory Limits**

| Device | Upper Memory Limit |
|---|---|
| EX4200, EX4500, EX4550 | 128 MB |
| EX4300 | 64 MB |
| MX5, MX10, MX40, MX80 | 64 MB |
| MX104 | 64 MB |
| MX240, MX480, MX960 | 2048 MB |
| OCX1100 | 64 MB |
| QFX3500, QFX3600 | 1024 MB |
| QFX5100 | 64 MB |
| QFX10002, QFX10008, QFX10016 | 1024 MB |

To expand the amount of memory available to the Puppet agent execution environment, including the Puppet agent and Facter processes:

1. Log in to the Puppet agent using the Puppet user account username and password.

2. In the Puppet user **$HOME/.cshrc** file, add the **limit data *memory*** command to the file. For example:

```
limit data 128M
```

**Release History Table**

| Release | Description |
|---|---|
| 2.0 | Starting in Puppet for Junos OS Release 2.0, the **jpuppet** packages are specific to the microprocessor architecture. In earlier releases, the packages are specific to a particular platform. |

RELATED DOCUMENTATION

Understanding Puppet for Junos OS | **13**

Puppet for Junos OS Supported Platforms | **15**

Monitoring and Troubleshooting Puppet for Junos OS | **80**

4

CHAPTER

# Managing Devices Running Junos OS

# Puppet Manifests for Devices Running Junos OS

IN THIS SECTION

## Creating Puppet Manifests Using the netdev Resources

Puppet manifests are files written in the Puppet language that describe your desired system configuration. The Puppet master compiles the manifests into catalogs. The agent nodes periodically download the catalogs and make the required changes so that the resulting system configuration matches the desired configuration.

Puppet manifest files are identified by the **.pp** extension. In the manifest, you use the Puppet language to describe the resources to manage on each agent node.

The **netdev_stdlib** module defines resource types that model properties for various network resources. The module includes resource definitions for the network device, physical interfaces, Layer 2 switching services, VLANs, and link aggregation groups (LAGs). For a list of available resource types, see “Puppet netdev Resources” on page 47.

The Juniper Networks **netdev_stdlib_junos** module, which you install on the Puppet master when managing devices running Junos OS, contains the Junos OS-specific Puppet provider code that implements the resource types defined in the **netdev_stdlib** module. Starting in **netdev_stdlib_junos** module version 2.0.2, the module also provides the **apply_group** defined resource type, which enables you to manage network resources that do not have type specifications in the **netdev_stdlib** module. For more information, see “Puppet for Junos OS apply_group Defined Resource Type” on page 62.

The following sample Puppet manifest is for a switch with the hostname jd.example.com. The manifest defines three VLANs, Pink, Green, and Red, with VLAN IDs 105, 101, and 103, respectively. The manifest defines that the ge-0/0/20 trunk interface accept tagged packets for both Pink and Green VLANs. By default, the ge-0/0/19 interface will be configured as an access port, which accepts untagged packets. The Red VLAN is the native VLAN for both ge-0/0/19 and ge-0/0/20.

```
node "jd.example.com" {
   netdev_device { $hostname: }
```

```
    netdev_vlan { "Pink":
       vlan_id => 105,
       description => "This is a pink vlan",
    }

    netdev_vlan { "Green":
       vlan_id => 101,
    }

    netdev_vlan { "Red":
       vlan_id => 103,
       description => "This is the native vlan",
    }

    netdev_l2_interface { 'ge-0/0/19':
       untagged_vlan => Red,
    }

    netdev_l2_interface { 'ge-0/0/20':
       description => "connected to R1-central",
       untagged_vlan => Red,
       tagged_vlans => [ Green, Pink ],
    }
 }
```

## Example: Creating Puppet Manifests for Devices Running Junos OS

**IN THIS SECTION**

This example shows how to create a sample Puppet manifest to manage VLANs and Layer 2 interfaces on a Puppet agent node running Junos OS. The manifest takes advantage of class definitions and variables in the Puppet language to create a more flexible and scalable manifest file.

### Requirements

- EX Series switch running Junos OS Release 12.3R2 or later 12.3 release with the **jpuppet** software package installed and a Junos OS user account for Puppet.
- Puppet master with the Juniper Networks NETCONF Ruby gem and **juniper/netdev_stdlib_junos** Puppet module installed.

### Overview

In this example, you create a Puppet manifest to manage VLANs and Layer 2 interfaces on switches running Junos OS that are in the “database” pod. The **netdev_stdlib** module defines the **netdev_device**, **netdev_vlan**, and **netdev_l2_interface** resource types that are used in this example to model the connection properties, VLANs, and Layer 2 interfaces on devices running Junos OS.

The Puppet class definition, **database_switch**, contains the settings for switches that are members of the “database” pod. Within the class definition, you must define a **netdev_device** resource that models the connection properties of the target switch. The **netdev_device** argument is the **$hostname** variable, which is provided by Facter. Within the class definition, you also create the **netdev_vlan** and **netdev_l2_interface** resources for the switches.

To create the necessary resources, this example uses the Puppet function **create_resources**, which converts a hash into a set of resources of the specified type. The function has two mandatory arguments, the resource type and a hash table that describes the resource titles and parameters. An optional third argument contains a hash table of default parameters that are applied to each new resource. If you specify the same parameter in both hash arguments, the parameter value in the mandatory argument overrides the default value in the optional argument.

In this example, you construct the variables **$vlans**, **$db_ports**, and **$db_port_settings**, which contain hashes that describe the VLAN and Layer 2 interface resources on the agent node. The hash values must be attributes that are defined in the netdev module for that resource type. You use the hashes as arguments to the **create_resources** Puppet function to create the resources that are added to the catalog.

The **$vlans** variable is a hash defining five VLAN resources spanning VLAN IDs in the range 100 through 104. Each hash entry defines the resource title (VLAN name) as the hash key and the resource attributes (vlan_id and description) as the hash values. For example:

```
$vlans = {
   'Blue'    => { vlan_id => 100, description => "This is a Blue VLAN, just updated" 
 },
   ...
}
```

The **$db_ports** variable is a hash defining which switch interfaces will be managed, and the **$db_port_settings** variable defines the default settings for these ports. The default settings configure the interface as a trunk interface that accepts tagged packets from the Blue, Green, and Yellow VLANs with the Red VLAN as the native VLAN.

```
$db_ports = {
   "ge-0/0/0"  => { description => "${db_port_desc} ge0" },
   ...
}
```

```
$db_port_settings = {
   untagged_vlan => Red,
   tagged_vlans => [Blue, Green, Yellow]
}
```

After you construct the hashes that define the resources, you use the **create_resources** function to create the resources. You create the VLAN resources by using the **create_resources** Puppet function with the **netdev_vlan** resource type and the **$vlan** hash as arguments. You create the Layer 2 interface resources by using the **create_resources** Puppet function with the **netdev_l2_interface** resource type and the **$db_ports** hash as arguments. Additionally, include the **$db_port_settings** hash as the optional third argument containing the default settings for those ports.

## Configuration

### Step-by-Step Procedure

To create a sample Puppet manifest to manage VLANs and Layer 2 interfaces on a Puppet agent node running Junos OS:

1. Create a file named **database_switch.pp**.

2. Define the VLANs that the Puppet agent will create on the agent nodes running Junos OS.

```
### Define a list of VLANS to create
$vlans = {
   'Blue'    => { vlan_id => 100, description => "This is a Blue VLAN, just
updated" },
   'Green'   => { vlan_id => 101, description => "This is a Green VLAN" },
   'Purple'  => { vlan_id => 102, description => "This is a Purple VLAN" },
   'Red'     => { vlan_id => 103, description => "This is a Red VLAN" },
   'Yellow'  => { vlan_id => 104, description => "This is a Yellow VLAN" }
}
```

3. Create the code block for the **database_switch** class, which will contain the settings for switches in the “database” pod.

```
### Define a class for all Switches in the 'database' pod
### Start class definition
class database_switch {

}
### End class definition
```

4. Within the **database_switch** class definition, define the **netdev_device** resource for the switch.

```
   netdev_device { $hostname: }
```

5. Within the **database_switch** class definition, create the VLAN resources by using the **create_resources** Puppet function with the **netdev_vlan** resource type and the **$vlans** hash as arguments.

```
# Create all the VLANs on the switch
   create_resources( netdev_vlan, $vlans )
```

6. Within the **database_switch** class definition, define the Layer 2 interfaces and port settings on the member switches.

```
# Set up ports to use a selected list of VLANS
   $db_port_desc = "This is for database"

   $db_ports = {
      "ge-0/0/0"  => { description => "${db_port_desc} ge0" },
      "ge-0/0/1"  => { description => "${db_port_desc} ge1" },
      "ge-0/0/2"  => { description => 'this is ge2' },
      "ge-0/0/10" => { description => 'this is ge10' },
      "ge-0/0/11" => { description => 'this is ge11' },
      "ge-0/0/12" => { description => 'this is ge12' }
   }

   $db_port_settings = {
      untagged_vlan => Red,
      tagged_vlans => [Blue, Green, Yellow]
   }
```

7. Within the **database_switch** class definition, create the Layer 2 interface resources by using the **create_resources** Puppet function with the **netdev_l2_interface** resource type, the **$db_ports** hash, and the **$db_port_settings** hash as arguments.

```
     create_resources( netdev_l2_interface, $db_ports, $db_port_settings )
```

8. Use the class definition for that node.

```
  ### Use the class definition for this node
  node "jd.example.com" {
     include database_switch
  }
```

***Results***

On the Puppet master, review the completed **database_switch.pp** manifest file. If the file does not display the intended code, repeat the instructions in this example to correct the manifest.

```
### Define a list of VLANS to create
$vlans = {
   'Blue'    => { vlan_id => 100, description => "This is a Blue VLAN, just updated"
 },
   'Green'   => { vlan_id => 101, description => "This is a Green VLAN" },
   'Purple'  => { vlan_id => 102, description => "This is a Purple VLAN" },
   'Red'     => { vlan_id => 103, description => "This is a Red VLAN" },
   'Yellow'  => { vlan_id => 104, description => "This is a Yellow VLAN" }
}

### Define a class for all Switches in the 'database' POD
### Start class definition
class database_switch {

   netdev_device { $hostname: }

# Create all the VLANs on the switch
   create_resources( netdev_vlan, $vlans )
# Set up ports to use a selected list of VLANS
   $db_port_desc = "This is for database"

   $db_ports = {
      "ge-0/0/0"  => { description => "${db_port_desc} ge0" },
      "ge-0/0/1"  => { description => "${db_port_desc} ge1" },
      "ge-0/0/2"  => { description => 'this is ge2' },
      "ge-0/0/10" => { description => 'this is ge10' },
      "ge-0/0/11" => { description => 'this is ge11' },
      "ge-0/0/12" => { description => 'this is ge12' }
   }

   $db_port_settings = {
      untagged_vlan => Red,
      tagged_vlans => [Blue, Green, Yellow]
   }

   create_resources( netdev_l2_interface, $db_ports, $db_port_settings )
}
### End class definition

### Use the class definition for this node
node "jd.example.com" {
   include database_switch
}
```

After the Puppet agent applies the configuration changes, the resulting configuration updates are:

```
[edit]
interfaces {
   ge-0/0/0 {
      unit 0 {
          description "This is for database ge0";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
      }
   }
   ge-0/0/1 {
      unit 0 {
          description "This is for database ge1";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
      }
   }
   ge-0/0/2 {
      unit 0 {
          description "this is ge2";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
      }
   }
   ge-0/0/10 {
      unit 0 {
          description "this is ge10";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
      }
   }
   ge-0/0/11 {
      unit 0 {
          description "this is ge11";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
```

```
      }
   }
   ge-0/0/12 {
      unit 0 {
          description "this is ge12";
          family ethernet-switching {
              port-mode trunk;
              vlan {
                  members [ Blue Green Yellow ];
              }
              native-vlan-id Red;
          }
      }
   }
}

vlans {
   Blue {
      description "This is a Blue vlan, just updated";
      vlan-id 100;
   }
   Green {
      description "This is a Green vlan";
      vlan-id 101;
   }
   Purple {
      description "This is a Purple vlan";
      vlan-id 102;
   }
   Red {
      description "This is a Red vlan";
      vlan-id 103;
   }
   Yellow {
      description "This is a Yellow vlan";
      vlan-id 104;
   }
}
```

## Verification

### *Verifying the Puppet Manifest*

#### Purpose

After the Puppet agent applies the configuration changes, verify that the Puppet agent node has the correct configuration.

**Action**

View the configuration or configuration differences, and verify that the Puppet agent made the correct changes. To view the full configuration, use the **show configuration** operational mode command. To view the configuration differences, use the **show configuration | compare rollback *rollback-number*** operational mode command.

**Meaning**

If the changes to the configuration include the updates defined in the manifest, then the manifest was created and applied correctly.

## Troubleshooting

***Troubleshooting Configuration Issues***

**Problem**

The configuration on the agent node does not reflect the changes requested in the manifest.

If you do not see any updates to the configuration, the switch might not be included in the managed agent nodes, or the Puppet agent might not have downloaded the latest catalog and performed the configuration update. If you do see updates to the configuration, but they are incorrect, the Puppet manifest might contain incorrect information.

**Solution**

Make sure that the Puppet master is properly configured to create the catalog for that node. If the Puppet master is properly configured, review the Puppet manifest file to ensure that it contains the correct configuration changes, and if necessary, correct the manifest.

If you have reporting enabled, also review the log files on the Puppet master to verify that the agent node downloaded the latest catalog and committed the configuration changes. If the Puppet agent could not obtain a lock on the configuration database, could not upload the configuration changes due to a syntax error, or could not commit the configuration on the device, the configuration remains unchanged.

RELATED DOCUMENTATION

Puppet netdev Resources | 47

Puppet for Junos OS apply_group Defined Resource Type | 62

Monitoring and Troubleshooting Puppet for Junos OS | 80

# Puppet netdev Resources

IN THIS SECTION

## Understanding the netdev_stdlib Puppet Resource Types

On the Puppet master, two Puppet modules are required to manage devices running Junos OS. The first module, **netdevops/netdev_stdlib**, includes the Puppet type definitions for the netdev resources. The netdev resources model the properties for various network resources and control specific Ethernet switch configuration such as VLANs. Table 11 on page 47 describes the resource types defined by the **netdev_stdlib** module. In the Puppet manifest, you use the netdev resource types in resource declarations to specify the desired configurations of the agent nodes running Junos OS.

NOTE: The **netdev_stdlib** resource definitions represent a superset of configuration parameters for that resource. The manifest file should only configure those parameters that are supported on a given platform or that are relevant to the given interface type.

**Table 11: Resource Types Defined in the netdev_stdlib Module**

| Type Name | Description |
|---|---|
| netdev_device | Models the properties of the network device. |
| netdev_interface | Models the properties for a physical interface.<br>The properties for a physical interface are managed separately from the services on the interface. |

Table 11: Resource Types Defined in the netdev_stdlib Module *(continued)*

| Type Name | Description |
|---|---|
| netdev_l2_interface | Models the properties for Layer 2 switching services on an interface.<br>The services for a Layer 2 interface are managed separately from the physical interface. |
| netdev_lag | Models the properties for a link aggregation group (LAG).<br>The properties for a LAG are managed separately from the physical member links and services on the interface. |
| netdev_vlan | Models the properties for a VLAN resource. |

The second Puppet module, **juniper/netdev_stdlib_junos**, includes the Junos OS-specific code that implements each of the types defined by **netdev_stdlib**. When you install the **netdev_stdlib_junos** module on the Puppet master, it automatically installs the **netdev_stdlib** module.

In a Puppet manifest, you must specify one and only one **netdev_device** for a given node. The netdev provider code automatically creates dependencies between the **netdev_device** resource and the other netdev resources. If the **netdev_device** cannot be created, then the Puppet agent does not process the other resources.

To create the **netdev_device** resource, the Puppet agent must open a NETCONF session with the device running Junos OS and establish an exclusive lock on the configuration database. Since the Puppet agent is running on the device, opening a connection should not fail. However, obtaining an exclusive lock could fail if another administrator is managing the device and already has a lock on the configuration database.

The **netdev_interface** resource type models the properties for a physical interface, whereas **netdev_l2_interface** models the properties for Layer 2 switching services on an interface. You only need to define the **netdev_interface** resource to change physical interface properties such as speed, MTU, or duplex mode. You do not need to define a **netdev_interface** resource as a prerequisite for defining a **netdev_l2_interface** resource.

The **netdev_vlan** resource type models the properties for a VLAN resource. A **netdev_l2_interface** resource can reference VLANs created using **netdev_vlan** resources, or it can reference VLANs already existing in the device configuration. Thus, you do not need to define a **netdev_vlan** resource in order to use VLANs in the **netdev_l2_interface** definition.

**NOTE:** Only the **netdev_device** and **netdev_interface** resources are supported on OCX1100 switches.

**NOTE:** To manage resources that do not have type specifications in the **netdev_stdlib** module, you can use the **apply_group** defined resource type provided as part of the **netdev_stdlib_junos** module.

## netdev_device

### Syntax

```
netdev_device { "name":  }
```

### Release Information

Resource support starting in **netdev_stdlib_junos** module version 1.0.0.

### Description

Puppet resource type that models the management connection to the agent node running Junos OS. In a Puppet manifest, you must specify one and only one **netdev_device** for a given node.

### Attributes

***name***—Name identifying the agent node. This can be a user-defined identifier and does not need to have any relationship to the actual node name.

### Usage Examples

The following Puppet manifest code creates a **netdev_device** resource. In this example, the **netdev_device** name is the value of the **$hostname** variable, which is provided by Facter.

```
node "jd.example.com" {

    netdev_device { $hostname: }
    <…additional resources…>

}
```

## netdev_interface

### Syntax

```
netdev_interface { "name":
    ensure => (present | absent),
    active => (true | false),
    admin => (up | down),
    description => "interface-description",
    speed => speed,
    duplex => (auto | full | half),
    mtu => mtu
}
```

**Release Information**

Resource support starting in **netdev_stdlib_junos** module version 1.0.0.

**Description**

Puppet resource type that enables you to model the properties and manage the configuration of a physical interface.

NOTE: The **netdev_stdlib** resource definitions represent a superset of configuration parameters for that resource. The manifest file should only configure those parameters that are supported on a given platform or that are relevant to the given interface type.

**Attributes**

***name***—Junos OS interface name, for example, ge-0/0/0.

**active** —(Optional) Specify whether to activate or deactivate the corresponding configuration. A value of **true** activates the configuration. A value of **false** deactivates the configuration without removing it.

**Default: true**

NOTE: If the resource declaration includes the **active** attribute and also **ensure => absent**, the client deletes the corresponding configuration and ignores the **active** attribute.

**admin**—(Optional) Configure the interface as administratively enabled or disabled. A value of **up** configures the interface as administratively enabled, and a value of **down** administratively disables the interface.

**Default: up**

**description**—(Optional) Interface description.

**Default:** “Puppet created interface: <name>”

**duplex**—(Optional) Interface duplex mode. Acceptable values are **auto**, **full**, and **half**.

**Default: auto**

**NOTE:** EX4300 switches support full duplex only. If you include the duplex attribute in your manifest file and set it to anything other than **full**, the Puppet agent displays an error message when it runs and ignores the duplex attribute setting.

**ensure**—(Optional) Specify whether to create or delete the configuration. A value of **present** creates the configuration. A value of **absent** deletes the configuration.

**Default: present**

**mtu**—(Optional) Maximum transmission unit (MTU) of the interface.

**speed**—(Optional) Interface speed. Acceptable values are **auto**, **10m**, **100m**, **1g**, and **10g**.

**Default: auto**

**NOTE:** Setting the speed attribute to the default value of **auto** causes the device to use the existing configuration for the **speed** statement and does not explicitly configure anything for the interface speed.

## Usage Examples

The following Puppet manifest code configures the description, speed, and duplex mode for interface ge-0/0/0:

```
node "jd.example.com" {

    netdev_device { $hostname: }

    netdev_interface { "ge-0/0/0":
        description => "connected to old hub",
        speed => 100m,
        duplex => full
   }
}
```

On a switch running Junos OS, the resulting configuration is:

`root@jd.example.com>` **show configuration interfaces ge-0/0/0**

```
description "connected to old hub";
ether-options {
    link-mode full-duplex;
    speed {
        100m;
    }
}
```

On an MX Series router running Junos OS, the resulting configuration is:

```
root@jd.example.com> show configuration interfaces ge-0/0/0
```

```
description "Connected to old hub";
speed 100m;
link-mode full-duplex;
```

If the Puppet manifest sets the **speed** attribute to **auto**, the device uses the existing configuration for the **speed** statement and does not explicitly configure anything for the interface speed. The following Puppet manifest code configures the **mtu** statement for the ge-0/0/0 interface and instructs the device to use the existing configuration for the **speed** statement:

```
node "jd.example.com" {

    netdev_device { $hostname: }

    netdev_interface { "ge-0/0/0":
        speed => auto,
        mtu => 1514
   }
}
```

The resulting configuration uses the existing configuration for the **speed** statement, which in this case is 100m.

```
root@jd.example.com> show configuration interfaces ge-0/0/0
```

```
speed 100m;
mtu 1514;
```

## netdev_l2_interface

### Syntax

```
netdev_l2_interface { "name":
    ensure => (present | absent),
    active => (true | false),
    description => "interface-description",
    tagged_vlans => (vlan  |  [vlan1, vlan2, vlan3, ...]),
    untagged_vlan => vlan,
    vlan_tagging => (enable | disable)
}
```

### Release Information

Resource support starting in **netdev_stdlib_junos** module version 1.0.0.

### Description

Puppet resource type that enables you to model the properties and manage the configuration of Layer 2 switching services on an interface. You do not need to define a **netdev_interface** resource as a prerequisite for defining a **netdev_l2_interface** resource.

NOTE: The **netdev_l2_interface** resource is not supported on OCX1100 switches.

A **netdev_l2_interface** resource can reference VLANs created using **netdev_vlan** resources, or it can reference VLANs that already exist in the device configuration. Thus, you do not need to define a **netdev_vlan** resource in order to use VLANs in the **netdev_l2_interface** definition.

### Attributes

***name***—Junos OS interface name, excluding any logical unit number, for example, ge-0/0/0.

**active** —(Optional) Specify whether to activate or deactivate the corresponding configuration. A value of **true** activates the configuration. A value of **false** deactivates the configuration without removing it.

**Default: true**

NOTE: If the resource declaration includes the **active** attribute and also **ensure => absent**, the client deletes the corresponding configuration and ignores the **active** attribute.

**description**—(Optional) Interface description.

**Default:** “Puppet created netdev_l2_interface: <name>”

**ensure**—(Optional) Specify whether to create or delete the configuration. A value of **present** creates the configuration. A value of **absent** deletes the configuration.

**Default: present**

**tagged_vlans**—(Optional) Configure one or more VLANs that can carry traffic on a trunk interface. The value can be a single VLAN name or an array of VLAN names. If you set this attribute, the code automatically configures the port as a trunk port.

**untagged_vlan**—(Optional) Configure the specified VLAN as the native VLAN on an interface. The value is the name of the VLAN for untagged packets.

**vlan_tagging**—(Optional) Configure the mode for the given port as access or trunk.

A value of **enable** configures the port in trunk mode, in which tagged packets are processed. A value of **disable** configures the port in access mode, in which tagged packets are discarded.

If you do not specify a value for this attribute, but you do set the **tagged_vlans** attribute, the code automatically configures the port as a trunk port. When you configure an MX Series router, you must define the **tagged_vlans** attribute for a trunk port configuration or define the **untagged_vlan** attribute for an access port configuration.

**Default: disable**

## Usage Examples

The following Puppet manifest code configures ge-0/0/0 as a trunk port accepting tagged frames from the **Pink** and **Green** VLANs. The code configures the **Red** VLAN as the native VLAN for that interface.

```
node "jd.example.com" {

    <…config omitted…>

    netdev_l2_interface { "ge-0/0/0":
        tagged_vlans => [ Green, Pink ],
        untagged_vlan => Red
    }

}
```

On a switch running Junos OS, the resulting configuration is:

`root@jd.example.com>` **show configuration interfaces ge-0/0/0**

```
unit 0 {
    description "Puppet created netdev_l2_interface: ge-0/0/0";
    family ethernet-switching {
        port-mode trunk;
        vlan {
            members [ Green Pink ];
        }
        native-vlan-id Red;
    }
}
```

On an MX Series router, the resulting configuration uses the corresponding VLAN IDs instead of VLAN names, as shown in the following output:

`root@jd.example.com>` **show configuration interfaces ge-0/0/0**

```
flexible-vlan-tagging;
native-vlan-id 103;
encapsulation flexible-ethernet-services;
unit 0 {
    description "Puppet created netdev_l2_interface: ge-0/0/0";
    family bridge {
        interface-mode trunk;
        vlan-id-list [ 101 103 105 ];
    }
}
```

## netdev_lag

### Syntax

```
netdev_lag { "name":
    ensure => (present | absent),
    active => (true | false),
    links => ('interface-name'  |  ['interface-name1', 'interface-name2' ...]),
    lacp => (active | disabled | passive),
    minimum_links => minimum
}
```

### Release Information

Resource support starting in **netdev_stdlib_junos** module version 1.0.0.

### Description

Puppet resource type that enables you to model the properties and manage the configuration of link aggregation groups (LAGs). In Junos OS, LAG ports are referred to as aggregated Ethernet bundles or ae ports.

NOTE: The **netdev_lag** resource is not supported on OCX1100 switches.

The **links** attribute causes physical interfaces to be added or removed from the LAG. To successfully assign the physical interfaces in the **links** attribute list to a LAG, you must ensure that there are no existing logical units configured on those physical interfaces. To enforce this prerequisite, you can use the **netdev_l2_interface** resource with **ensure=>absent** to remove any existing logical units.

NOTE: Junos OS requires at least one unit configured under the LAG (ae) port for the links to display as part of the **show** command. Therefore, you need to define Layer 2 services using the **netdev_l2_interface** resource type.

**Attributes**

***name***—Junos OS LAG name, excluding any logical unit number, for example, ae0.

**active**—(Optional) Specify whether to activate or deactivate the corresponding configuration. A value of **true** activates the configuration. A value of **false** deactivates the configuration without removing it.

**Default: true**

NOTE: If the resource declaration includes the **active** attribute and also **ensure => absent**, the client deletes the corresponding configuration and ignores the **active** attribute.

**ensure**—(Optional) Specify whether to create or delete the configuration. A value of **present** creates the configuration. A value of **absent** deletes the configuration.

**Default: present**

**lacp**—(Optional) Link Aggregation Control Protocol (LACP) mode.

- **disabled**—LACP is not used.
- **active**—LACP active mode.
- **passive**—LACP passive mode.

**Default: disabled**

**links**—Configure one or more physical interfaces as members of the LAG bundle. The value can be a single interface or an array of interfaces.

**minimum_links**—(Optional) Integer that defines the minimum number of physical links that must be in the **up** state to declare the LAG port in the **up** state.

### Usage Examples

The following Puppet manifest code configures a LAG bundle ae0 consisting of three interfaces, ge-0/0/15, ge-0/0/20, and ge-0/0/21, which accept tagged frames from the **Blue** and **Green** VLANs. The code configures the **Red** VLAN as the native VLAN.

```
node "jd.example.com" {

    <…config omitted…>

    netdev_lag { "ae0":
        links => [ 'ge-0/0/15', 'ge-0/0/20', 'ge-0/0/21' ]
    }

    netdev_l2_interface { "ae0":
        tagged_vlans => [ Blue, Green ],
        untagged_vlan => Red
    }

}
```

On a switch running Junos OS, the resulting configuration is:

`root@jd.example.com>` **show configuration interfaces**

```
ge-0/0/15 {
    ether-options {
        802.3ad ae0;
    }
}
ge-0/0/20 {
    ether-options {
        802.3ad ae0;
    }
}
ge-0/0/21 {
    ether-options {
        802.3ad ae0;
    }
}
ae0 {
    unit 0 {
        description "Puppet created netdev_l2_interface: ae0";
        family ethernet-switching {
            port-mode trunk;
            vlan {
                members [ Blue Green ];
            }
            native-vlan-id Red;
        }
    }
}
```

On an MX Series router running Junos OS, the resulting configuration is:

`root@jd.example.com>` **show configuration interfaces**

```
ge-0/0/15 {
    gigether-options {
        802.3ad ae0;
    }
}
ge-0/0/20 {
    gigether-options {
        802.3ad ae0;
    }
}
ge-0/0/21 {
    gigether-options {
        802.3ad ae0;
    }
}

ae0 {
    apply-macro "netdev_lag[:links]" {
        ge-0/0/15;
        ge-0/0/20;
        ge-0/0/21;
    }
    flexible-vlan-tagging;
    native-vlan-id 103;
    encapsulation flexible-ethernet-services;
    unit 0 {
        description "Puppet created netdev_l2_interface: ae0";
        family bridge {
            interface-mode trunk;
            vlan-id-list [ 103 520 101 ];
        }
    }
}
```

**NOTE:** Puppet for Junos OS uses an **apply-macro** statement in LAG configurations to identify the list of LAG members.

## netdev_vlan

### Syntax

```
netdev_vlan { "name":
    ensure => (present | absent),
    active => (true | false),
    vlan_id => id,
    description => "vlan-description"
}
```

### Release Information

Resource support starting in **netdev_stdlib_junos** module version 1.0.0.

### Description

Puppet resource type that enables you to model the properties and manage the configuration of VLANs on agent nodes running Junos OS.

**NOTE:** The **netdev_vlan** resource is not supported on OCX1100 switches.

### Attributes

***name***—Name of the VLAN, which must be a VLAN name that is valid on the agent node.

**active**—(Optional) Specify whether to activate or deactivate the corresponding configuration. A value of **true** activates the configuration. A value of **false** deactivates the configuration without removing it.

**Default: true**

NOTE: If the resource declaration includes the **active** attribute and also **ensure => absent**, the client deletes the corresponding configuration and ignores the **active** attribute.

**description**—(Optional) VLAN description.

**Default:** "Puppet created VLAN: <name>: <vlan-id>"

**ensure**—(Optional) Specify whether to create or delete the configuration. A value of **present** creates the configuration. A value of **absent** deletes the configuration.

**Default: present**

**vlan_id**—VLAN tag identifier. Valid VLAN IDs range from 1 through 4094.

### Usage Examples

The following Puppet manifest code defines a VLAN named **Green** with a VLAN ID of 500:

```
node "jd.example.com" {

    netdev_device { $hostname: }

    netdev_vlan { "Green":
        vlan_id => 500
    }

}
```

On a switch running Junos OS, the resulting configuration is:

```
vlans {
    Green {
        description "Puppet created VLAN: Green: 500";
        vlan-id 500;
    }
}
```

On an MX Series router, the resulting configuration is:

```
bridge-domains {
    Green {
        description "Puppet created VLAN: Green: 500";
        domain-type bridge;
        vlan-id 500;
   }
}
```

The following Puppet manifest code deactivates the **Green** VLAN, which has a VLAN ID of 500:

```
node "jd.example.com" {

    netdev_device { $hostname: }

    netdev_vlan { "Green":
        active => false,
        vlan_id => 500
    }

}
```

On a switch running Junos OS, the resulting configuration is:

`root@jd.example.com>` **show configuration vlans**

```
inactive: Green {
    description "Puppet created VLAN: Green: 500";
    vlan-id 500;
}
```

On an MX Series router, the resulting configuration is:

`root@jd.example.com>` **show configuration bridge-domains**

```
inactive: Green {
    description "Puppet created VLAN: Green: 500";
    domain-type bridge;
    vlan-id 500;
}
```

RELATED DOCUMENTATION

# Puppet for Junos OS apply_group Defined Resource Type

IN THIS SECTION

## Understanding the Puppet for Junos OS apply_group Defined Resource Type

Puppet for Junos OS enables you to use Puppet to manage certain devices running the Junos® operating system (Junos OS). The Puppet **netdev_stdlib** module is a vendor-neutral network abstraction framework that defines Puppet type specifications for certain resources used on network devices. The **netdev_stdlib_junos** module contains the provider implementations that enable you to configure these resources, which include physical interfaces, VLANs, link aggregation groups (LAGs), and Layer 2 switching services, on devices running Junos OS.

Puppet enables you to more easily manage resources on your network devices, but you are generally limited to configuring resource types that are already defined and implemented. Starting with **netdev_stdlib_junos** module version 2.0.2, the module provides the **apply_group** defined resource type, which enables you to manage resources that do not have separate type specifications. **apply_group** enables you to create generic resources as groups under the **[edit groups]** hierarchy level and apply those groups to your configuration.

The **apply_group** defined resource type references a custom Embedded Ruby (ERB) template that generates Junos OS configuration data. ERB is a templating engine for Ruby that enables you to create templates consisting of plain text documents with embedded Ruby code. ERB is part of the Ruby standard library, and ERB templates are supported in Puppet.

Because ERB templates are plain text documents, the template can include Junos OS configuration data in any of the supported formats including formatted ASCII text, Junos XML elements, or Junos OS **set** commands. The ability to add Ruby code to the template provides flexibility through the use of variable substitution and flow control. You can customize the input provided to the template for different Puppet client nodes by defining the relevant variables for that node in the manifest. When the ERB template is rendered, the plain text is copied directly to the output, and the embedded Ruby tags are processed. The client node applies the resulting configuration changes at the **[edit groups]** hierarchy level under the group name that matches the title for that **apply_group** resource.

An **apply_group** resource enables you to create and delete configuration groups as well as activate or deactivate a group. When you create or activate a configuration group, the client node also configures the group name in the **apply-groups** statement at the top of the configuration hierarchy so that the configuration inherits the statements in the corresponding group. When you delete or deactivate a configuration group, the client node removes the group name from the **apply-groups** statement if configured.

## Creating Embedded Ruby Templates to Use with the Puppet for Junos OS apply_group Resource

Puppet for Junos OS enables you to use Puppet to manage certain devices running Junos OS. Starting with **netdev_stdlib_junos** module version 2.0.2, you can use the **apply_group** defined resource type to manage resources in the Junos OS configuration that do not have type specifications in the **netdev_stdlib** module. An **apply_group** resource references a custom Embedded Ruby (ERB) template that generates the Junos OS configuration data for a specific resource using the supplied inputs.

The ERB templating system for Ruby enables you to generate output from a template consisting of plain text with embedded Ruby code. You can create generic ERB templates that generate the desired Junos OS configuration data for any resource. Because ERB templates are plain text documents, the template can include Junos OS configuration data in any of the supported formats including formatted ASCII text, Junos XML elements, or Junos OS **set** commands. When the ERB template is rendered, the plain text is copied directly to the output, and the code in embedded Ruby tags is executed as Ruby code. ERB templates can also reference any node-specific Puppet variables that you define in the manifest.

ERB templates that are referenced by the **apply_group** resource must be placed in the **netdev_stdlib_junos/templates** directory on the Puppet master. The template filename must use the following format where the base filename can be any user-defined string, and the configuration format must reflect the format of the configuration data in the template, which can be "set", "text", or "xml". If the filename does not specify a format, the Puppet client uses XML as the default.

```
filename.configuration-format.erb
```

After you create and stage your ERB templates, you can use them to generate configuration data for resources on client nodes running Junos OS. To use a template, the Puppet manifest must include the **apply_group** resource, and the **template_path** attribute must reference the template. Any variables required by the template must be declared in the manifest.

ERB templates can contain Ruby tags, which are delimited by <% and %>. Table 12 on page 64 summarizes the different ERB tag types, their syntax, and their impact on the rendered output. ERB code tags are generally used for flow control. The ERB processor executes code in a code tag but does not insert any values into the output. ERB tags that contain an equals sign (=) are expressions. The ERB processor evaluates the expression and places the resulting value in the output. ERB tags that contain the hash (#) symbol are comments that do not affect the rendered output.

**Table 12: Embedded Ruby Tag Types**

| Tag Type | Syntax | Behavior |
|---|---|---|
| Code | **<% *code* %>** | Executes the code, but does not insert a value into the output. |
| Comment | **<%# *comment* %>** | Ignores any code following # and does not insert any text into the output. |
| Expression | **<%= *expression* %>** | Generates a value from the expression and inserts the value into the output. |
| Literal | **<%% %%>** | Inserts a literal **<% %>** into the output. |

You can use Ruby tags in your templates to manipulate data, perform variable substitutions, iterate over indexed collections like arrays and hashes, and create conditional constructs. Some of the more common constructs are presented here. For detailed information about using ERB templates in Puppet, see https://puppet.com/docs/puppet/latest/lang_template_erb.html.

An ERB template can iterate over collections, such as arrays or hashes, by using the **<% @*variable*.each ... %>** syntax. The following template iterates over each service in an array and generates Junos OS configuration data that configures each service at the **[edit system services]** hierarchy level under the specified configuration group.

```
<% @services.each do | service | %>
set system services <%= service[0] %> <%= service[1] %>
<% end %>
```

Consider the following array declaration in the manifest, which defines several services:

```
$services = [ [ 'ftp' ], [ 'ssh' ], [ 'telnet' ], [ 'netconf', 'ssh' ] ]
```

When the template is evaluated with the given array, the template generates the following configuration data as **set** commands:

```
set system services ftp
set system services ssh
set system services telnet
set system services netconf ssh
```

You can also iterate over items in a hash, which is indexed using a key rather than a number. The following ERB template iterates over log files in a hash and generates configuration data that configures the files along with their facility and severity details at the **[edit system syslog]** hierarchy level under the specified configuration group. Each log file is mapped to an array of hashes that store the facility and severity details.

```
system {
    syslog {
    <% @syslog_names.each do | name, details | %>
        file <%= name %> {
            <% details.each do | detail | %>
                  <%= detail['facility'] %> <%= detail['level'] %>;
            <% end %>
        }
        <% end %>
    }
}
```

Consider the following hash declaration in the manifest, which defines two log files and specifies the facility and severity of the messages to include in each log:

```
$syslog_names = {
  'messages' => [ { 'facility' => 'any', 'level' => 'critical' }, { 'facility' =>
'authorization', 'level' => 'info' } ] ,
  'interactive-commands' => [ { 'facility' => 'interactive-commands', 'level' =>
'any'} ]
}
```

When the template is evaluated with the given hash, the template generates the following configuration data in text format:

```
system {
    syslog {
        file messages {
            any critical;
            authorization info;
        }
        file interactive-commands {
            interactive-commands any;
        }
    }
}
```

You can also create conditional constructs like the following to modify the configuration data based on the presence or absence of variables in the supplied inputs:

```
<% if condition %>
  text
<% end %>
```

For example, suppose that you are configuring a number of physical interfaces, and you only want to configure a logical interface when the relevant information is included in the supplied inputs. Consider the following hash declaration in the manifest:

```
$interfaces = {
  'ge-0/0/1' => {'unit' => 0, 'description' => 'to-B', 'family' => 'inet', 'address'
 => '198.51.100.1/30' },
  'ge-0/0/2' => {'unit' => 0, 'description' => 'to-D', 'family' => 'inet', 'address'
 => '198.51.100.5/30' },
  'ge-0/0/3' => {'description' => 'to-E'}
}
```

A template can test whether the hash for each interface contains a **unit** key and then modify the configuration output based on the result. The following ERB template generates configuration data that configures a description for each physical interface but only configures the logical interface when the **unit** key is present in the hash for that interface:

```
<interfaces>
    <% @interfaces.each do | name, hash | %>
    <interface>
        <name><%= name %></name>
        <description><%= hash['description'] %></description>
        <% if hash.has_key?('unit') %>
        <unit>
            <name><%= hash['unit'] %></name>
            <family>
                <<%= hash['family'] %>>
                    <address>
                        <name><%= hash['address'] %></name>
                    </address>
                </<%= hash['family'] %>>
            </family>
        </unit>
        <% end %>
    </interface>
    <% end %>
</interfaces>
```

The template generates the following Junos XML configuration data, which does not configure a logical unit for the ge-0/0/3 interface:

```
<interfaces>
    <interface>
        <name>ge-0/0/1</name>
        <description>to-B</description>
        <unit>
            <name>0</name>
            <family>
                <inet>
                    <address>
                        <name>198.51.100.1/30</name>
                    </address>
                </inet>
            </family>
        </unit>
    </interface>
    <interface>
        <name>ge-0/0/2</name>
        <description>to-D</description>
        <unit>
            <name>0</name>
            <family>
                <inet>
                    <address>
                        <name>198.51.100.5/30</name>
                    </address>
                </inet>
            </family>
        </unit>
    </interface>
    <interface>
        <name>ge-0/0/3</name>
        <description>to-E</description>
    </interface>
</interfaces>
```

To avoid creating ERB templates from scratch, you can copy a portion of an existing device configuration into a new ERB template file, replace the variables in the configuration data with appropriate ERB variables, and add Ruby tags as required for flow control. For an example outlining how to copy and convert a configuration into an ERB template, see Puppet + ERB Templates + Junos = Increased automation agility and flexibility.

For more information about using Puppet templates, see the official Puppet documentation at https://puppet.com/docs/puppet/latest/lang_template.html.

## Declaring the Puppet for Junos OS apply_group Resource in a Manifest

Puppet for Junos OS enables you to use Puppet to manage certain devices running Junos OS. You can use the **apply_group** defined resource type to manage generic resources in the Junos OS configuration that do not have type specifications in the **netdev_stdlib** module. An **apply_group** resource references a custom Embedded Ruby (ERB) template that generates the configuration data for the specific resource using the supplied inputs.

You declare resources of type **apply_group** in your manifest. When you declare the resource, you must define a title, which determines the group name under which the Puppet client applies the configuration changes. You must also define the **template_path** attribute to reference the desired ERB template located in the **netdev_stdlib_junos/templates** directory on the Puppet master. The **template_path** attribute follows

Puppet's normal convention of using ***module/template-filename*** for referencing template files. By default, Puppet looks for the template in the given module's **templates** directory. For example:

```
netdev_stdlib_junos::apply_group{ "services_group":
  template_path => "netdev_stdlib_junos/services.set.erb",
}
```

The **apply_group** resource type includes two optional attributes, **ensure** and **active**. The **ensure** attribute determines whether to create or delete a configuration group, and the **active** attribute determines whether the group should be active or inactive on the device. Table 13 on page 69 outlines the effects on the configuration for different attribute settings.

Setting **ensure** to **present** causes the client to create the configuration group in the Junos OS configuration at the **[edit groups *group-name*]** hierarchy level, whereas setting **ensure** to **absent** causes the client to delete the corresponding configuration group from the device configuration. Setting **active** to **true** activates the configuration group and adds the group name to the **apply-groups** statement at the top of the configuration hierarchy. Setting **active** to **false**, on the other hand, deactivates the configuration and removes the group name from the **apply-groups** statement, if configured. When you deactivate the configuration group, the device marks it with the **inactive:** tag and ignores that portion of the configuration when you commit it.

If the group name is configured under the **apply-groups** statement, the configuration inherits the statements in that configuration group. The order of the groups in the **apply-groups** statement determines the inheritance priority. The configuration data in the first group takes priority over the data in subsequent groups.

**Table 13: ensure and active Attributes for apply_group Resources**

| Attribute Settings | Configuration Group | **apply-groups** Statement |
|---|---|---|
| **ensure => present**<br>**active => true**<br><br>(Default) | Create or modify the configuration group and ensure it is active | Add the group name to the **apply-groups** statement, if not already present. |
| **ensure => present**<br>**active => false** | Create or modify the configuration group and deactivate it | Remove the group name from the **apply-groups** statement, if present. |
| **ensure => absent** | Delete the configuration group | Remove the group name from the **apply-groups** statement, if present. |

**NOTE:** If **active** is set to **true** but **ensure** is set to **absent**, the client still deletes the group name from the **apply-group** statement, because the configuration group does not exist.

If the **apply_group** resource uses an ERB template that references Puppet variables, you must declare the necessary variables for that node in the manifest. Puppet variables are prefixed with a dollar sign ($).

When the Puppet client node downloads the catalog, it applies the configuration changes generated by the template at the **[edit groups *group-name*]** hierarchy level in the configuration and updates the group name in the **apply-groups** statement as instructed. The client also stores a copy of the configuration group in a **/var/tmp/*group-name*** file on the device, which can be useful for troubleshooting any issues.

The following steps outline how to add an **apply_group** resource to your manifest. In this example, the **apply_group** resource references the following ERB template named **services.set.erb** in the **netdev_stdlib_junos/templates** directory:

```
<% @services.each do | service | %>
set system services <%= service[0] %> <%= service[1] %>
<% end %>
```

1. Declare any Puppet variables that are used by the ERB template to generate the configuration data for that node.

   ```
   $services = [ [ 'ftp' ], [ 'ssh' ], [ 'telnet' ], [ 'netconf', 'ssh' ] ]
   ```

2. Declare an **apply_group** resource, and define the group name under which the configuration changes are applied.

   ```
   netdev_stdlib_junos::apply_group{ "services_group":

   }
   ```

3. Define the **template_path** attribute, and reference the desired ERB template located in the **netdev_stdlib_junos/templates** directory.

   ```
   netdev_stdlib_junos::apply_group{ "services_group":
     template_path => "netdev_stdlib_junos/services.set.erb",
   }
   ```

4. (Optional) Define the **ensure** attribute as **present** or **absent** to specify whether to create or delete the configuration group.

   If you omit the attribute, it defaults to **present**.

```
netdev_stdlib_junos::apply_group{ "services_group":
  template_path => "netdev_stdlib_junos/services.set.erb",
  ensure        => present,
}
```

5. (Optional) Define the **active** attribute as **true** or **false** to specify whether to activate or deactivate the configuration group.

   If you omit the attribute, it defaults to **true**.

   ```
   netdev_stdlib_junos::apply_group{ "services_group":
     template_path => "netdev_stdlib_junos/services.set.erb",
     ensure        => present,
     active        => true,
   }
   ```

An **apply_group** resource in a sample manifest file is presented here:

```
node "jd.example.com" {
  netdev_device { $hostname: }

  $services = [ [ 'ftp' ], [ 'ssh' ], [ 'telnet' ], [ 'netconf', 'ssh' ] ]

  netdev_stdlib_junos::apply_group{ "services_group":
    template_path => "netdev_stdlib_junos/services.set.erb",
    ensure => present,
    active => true,
  }
}
```

## Example: Using the Puppet for Junos OS apply_group Resource to Configure Devices Running Junos OS

**IN THIS SECTION**

Puppet for Junos OS enables you to use Puppet to manage certain devices running Junos OS. This example shows how to use the **apply_group** defined resource type with an Embedded Ruby (ERB) template to configure a BGP resource, which does not have a type specification in the **netdev_stdlib** module.

### Requirements

This example uses the following hardware and software components:

- MX80 router running Junos OS Release 14.2R2 with the **jpuppet** software package installed and a Junos OS user account for Puppet.
- Puppet master with the Juniper Networks NETCONF Ruby gem and **netdev_stdlib_junos** module version 2.0.3 installed.

### Overview

This example creates a Puppet manifest that uses the **apply_group** resource to configure statements for internal and external BGP peering for the puppet-client.example.com node. The **apply_group** resource references the **bgp.set.erb** ERB template, which generates the configuration data for the resource. The template is located in the **modules/netdev_stdlib_junos/templates** directory.

The Puppet manifest declares the **$bgp** variable, which contains the node-specific configuration values that the template uses to generate the configuration data for that node. The data is provided in a hash that uses the BGP group names as keys. Each key maps to another hash that contains the details for that group including the group type, and the IP addresses and AS number of the peers. When the template is referenced, it iterates over the hash and generates the configuration data as Junos OS **set** commands.

The title for the **apply_group** resource defines the **bgp_group** group name under which the configuration changes are applied at the **[edit groups]** hierarchy level. The **template_path** attribute is set to **netdev_stdlib_junos/bgp.set.erb**, which references the **bgp.set.erb** template. The **ensure** attribute is set to **present** to instruct the client to create the configuration on the device, and the **active** attribute is set to **true** to make sure that the configuration is active and that the group name is configured under the **apply-groups** statement. Both attributes are optional in this case, because they are set to the default values.

When the client downloads the catalog, it adds the configuration data generated by the template under the **[edit groups bgp_group]** hierarchy level and configures the **apply-groups** statement to include the

**bgp_group** group name. If the commit succeeds, the configuration inherits the statements in the configuration group.

**NOTE:** This example assumes that the local autonomous system number is already defined on the device.

## Configuration

### *Creating the ERB Template*

#### Step-by-Step Procedure

To create and stage the ERB template:

1. Create a new template file named **bgp.set.erb**, and add the text and Ruby tags required to generate the desired configuration data for the BGP resource.

```
<% @bgp.each do | name, hash | %>
    set protocols bgp group <%=name%> type <%= hash['type']%>
    set protocols bgp group <%=name%> local-address <%= hash['local-address']%>
    set protocols bgp group <%=name%> peer-as <%= hash['peer-as']%>
    <% hash['neighbor'].each do | neighbor | %>
        set protocols bgp group <%=name%> neighbor <%=neighbor%>
    <% end %>
<% end %>
```

2. Place the template file in the **modules/netdev_stdlib_junos/templates** directory on the Puppet master.

### *Creating the Manifest*

#### Step-by-Step Procedure

To declare the **apply_group** resource in a Puppet manifest and reference the ERB template:

1. Create the manifest file and define the client node.

```
node 'puppet-client.example.com'{
  netdev_device { $hostname:}

  # variable declarations and resources

}
```

2. Declare any Puppet variables that are used by the template to configure that node.

```
$bgp = {
    'internal' =>  {
        'type' => 'internal',
        'neighbor' => [ '10.0.0.3', '10.0.0.4' ],
        'local-address' => '10.0.0.2',
        'peer-as' => '64501'
     },
     'external' => {
         'type' => 'external',
         'neighbor' => [ '10.1.12.1', '10.1.12.5' ],
         'local-address' => '10.0.0.2',
         'peer-as' => '64502'
     }
}
```

3. Declare the **apply_group** resource and its title, which defines the group name under which the configuration data will be added at the **[edit groups *group-name*]** hierarchy level.

```
netdev_stdlib_junos::apply_group{ "bgp_group":

}
```

4. Set the **apply_group template_path** attribute to reference the **bgp.set.erb** template.

```
  netdev_stdlib_junos::apply_group{ "bgp_group":
    template_path => "netdev_stdlib_junos/bgp.set.erb",
  }
```

5. (Optional) Set the **apply_group ensure** attribute to **present** to create the configuration group.

```
netdev_stdlib_junos::apply_group{ "services_group":
  template_path => "netdev_stdlib_junos/services.set.erb",
  ensure        => present,
}
```

6. (Optional) Set the **apply_group active** attribute to **true** to activate the configuration.

```
netdev_stdlib_junos::apply_group{ "bgp_group":
  template_path => "netdev_stdlib_junos/bgp.set.erb",
  ensure        => present,
  active        => true,
}
```

***Results***

On the Puppet master, review the manifest. If the manifest does not display the intended code, repeat the instructions in this example to correct the manifest.

```
node 'puppet-client.example.com'{
  netdev_device { $hostname:}

  $bgp = {
      'internal' =>  {
          'type' => 'internal',
          'neighbor' => [ '10.0.0.3', '10.0.0.4' ],
          'local-address' => '10.0.0.2',
          'peer-as' => '64501'
       },
       'external' => {
           'type' => 'external',
           'neighbor' => [ '10.1.12.1', '10.1.12.5' ],
           'local-address' => '10.0.0.2',
           'peer-as' => '64502'
       }
  }

  netdev_stdlib_junos::apply_group{ "bgp_group":
    template_path => "netdev_stdlib_junos/bgp.set.erb",
    ensure        => present,
    active        => true,
  }
}
```

## Verification

IN THIS SECTION

To verify that the commit was successful and the configuration reflects the new BGP resource, perform these tasks:

### *Verifying the Commit*

#### Purpose

Verify the commit by reviewing the commit history for the Puppet node.

#### Action

From operational mode, you can enter the **show system commit** command to verify that the catalog changes were successfully committed.

`puppet@puppet-client>` **show system commit**

```
0   2015-10-14 16:14:56 PDT by puppet via netconf
    Puppet agent catalog: 1444894500
...
```

#### Meaning

The commit log indicates that the Puppet client successfully applied the configuration changes generated by the template.

### *Verifying the Configuration*

#### Purpose

Verify that the BGP configuration group is in the active configuration on the device and that the configuration group name is configured for the **apply-groups** statement.

#### Action

From operational mode, enter the **show configuration groups bgp_group** and the **show configuration apply-groups** commands.

`puppet@puppet-client>` **show configuration groups bgp_group**

```
protocols {
    bgp {
        group internal {
            type internal;
            local-address 10.0.0.2;
            peer-as 64501;
            neighbor 10.0.0.3;
            neighbor 10.0.0.4;
        }
        group external {
            type external;
            local-address 10.0.0.2;
            peer-as 64502;
            neighbor 10.1.12.1;
            neighbor 10.1.12.5;
        }
    }
}
```

`puppet@puppet-client>` **show configuration apply-groups**

```
apply-groups [ global re0 re1 bgp_group ];
```

## apply_group

### Syntax

```
netdev_stdlib_junos::apply_group { "group-name":
    template_path => "netdev_stdlib_junos/template-filename",
    ensure => (present | absent),
    active => (true | false),
}
```

### Release Information

Defined resource type introduced in **netdev_stdlib_junos** module version 2.0.2.

### Description

Defined resource type in the Juniper Networks **netdev_stdlib_junos** Puppet module that enables you to manage network resources that do not have type specifications in the **netdev_stdlib** module. **apply_group** enables you to create generic resources as groups under the **[edit groups *group-name*]** hierarchy level and apply those groups to the configuration of devices running Junos OS.

**apply_group** references an Embedded Ruby (ERB) template, which takes inputs defined in the manifest and generates the Junos OS configuration data that is configured on the client node. ERB templates referenced by an **apply_group** resource must be placed in the **netdev_stdlib_junos/templates** directory.

### Attributes

***group-name***—Name of the group at the **[edit groups]** hierarchy level in the Junos OS configuration under which the configuration changes are applied.

**active**—(Optional) Specify whether to activate or deactivate the corresponding configuration group. A value of **true** activates the configuration group and adds the group name to the **apply-groups** statement in the configuration. A value of **false** deactivates the configuration group and removes the group name from the **apply-groups** statement if configured.

**Default: true**

**ensure**—(Optional) Specify whether to create or delete the configuration group. A value of **present** creates the configuration group. A value of **absent** deletes the configuration group and removes the group name from the **apply-groups** statement if configured.

**Default: present**

**template_path**—Reference to an ERB template. The value uses the Puppet convention for referencing template files, which is ***module/template-filename***. The module name is **netdev_stdlib_junos**, and the template filename is an ERB template file residing in the **netdev_stdlib_junos/templates** directory.

### Usage Examples

Consider the following ERB template, **interface.set.erb**, which iterates over a collection of interfaces. When rendered, the template generates configuration data that configures each interface with a description and a logical unit that has a protocol family and an address.

```
<% @interfaces.each do | name, hash | %>
set interfaces <%= name %> description <%= hash['description'] %> unit <%=
hash['unit']%> family <%= hash['family'] %> address <%= hash['address']%>
<% end %>
```

The following Puppet manifest uses an **apply_group** resource to configure the specified interfaces under the **[edit group interface_group]** hierarchy level. **apply_group** references the **interface.set.erb** ERB template in the **netdev_stdlib_junos/templates** directory.

```
node "jd.example.com" {
  netdev_device { $hostname: }

  # Variables passed to the template file
  $interfaces = {
  'ge-1/1/1' => {'unit' => 0, 'description' => 'to-B', 'family' => 'inet', 'address'
 => '198.51.100.1/30' },
  'ge-1/2/1' => {'unit' => 0, 'description' => 'to-D', 'family' => 'inet', 'address'
 => '198.51.100.5/30' }
  }
  netdev_stdlib_junos::apply_group { "interface_group":
    template_path => "netdev_stdlib_junos/interface.set.erb",
    ensure        => present,
    active        => true,
  }
}
```

Puppet renders the configuration data in the template using the inputs defined in the manifest.

```
set interfaces ge-1/1/1 description to-B unit 0 family inet address 198.51.100.1/30
set interfaces ge-1/2/1 description to-D unit 0 family inet address 198.51.100.5/30
```

The Puppet client configures the data under the **[edit groups interface_group]** hierarchy level and adds the group name to the **apply-groups** statement.

`puppet@jd.example.com>` **show configuration groups interface_group**

```
interfaces {
    ge-1/1/1 {
        description to-B;
        unit 0 {
            family inet {
                address 198.51.100.1/30;
            }
        }
    }
    ge-1/2/1 {
        description to-D;
        unit 0 {
                family inet {
                    address 198.51.100.5/30;
                }
            }
        }
    }
```

`puppet@jd.example.com>` **show configuration apply-groups**

```
apply-groups [ global re0 re1 interface_group ];
```

RELATED DOCUMENTATION

Puppet Manifests for Devices Running Junos OS | 37

Puppet netdev Resources | 47

# Monitoring and Troubleshooting Puppet for Junos OS

**IN THIS SECTION**

## Reporting for Puppet Agents Running Junos OS

You can require a Puppet agent to compile reports containing the log messages and metrics that are generated during configuration updates. To require that the Puppet agent report to the server after each transaction, you must set the agent report value to true in the **puppet.conf** file. If you enable reporting, by default, the agent node sends a YAML-formatted transaction report to the same server from which it downloads its configuration.

Puppet log messages can identify the source, severity level, and timestamp of the message, information about the operation or error that generated the message, and any tags associated with that operation or error. The Puppet agent always generates log messages with a severity level of notice, info, or err as part of a normal update. To generate log messages with a severity level of debug, you must specify the **--debug** option when you run the Puppet agent.

The Junos OS provider code for the **netdev_stdlib_junos** module designates log entries specific to Junos OS processing with **source: JUNOS**. Table 14 on page 81 describes the Puppet agent reporting logs generated for Junos OS operations.

**Table 14: Puppet Agent Reporting Logs for Devices Running Junos OS**

| Severity Level | Operation | Message Content | Tags |
|---|---|---|---|
| debug | configuration changes | Junos OS configuration changes in XML format. | debug, config, changes |
| debug | operational updates | Information concerning the operation, for example: "Opening a local connection: jex.example.com". | debug |
| err | commit operation failed | Reason for failed commit. | config, fail |
| info | commit operation requested | Number of configuration changes. | config, commit |
| notice | configuration changes | Junos OS configuration changes in a diff format. | config, changes |
| notice | commit operation successful | Commit success message. | config, success |

The following examples show sample log messages generated by a Puppet agent while performing a configuration update.

- The following sample log message shows that the Puppet agent requested a commit operation involving one change to the configuration:

```
    - !ruby/object:Puppet::Util::Log
      level: !ruby/sym info
      message: Committing 1 changes.
      source: JUNOS
      tags:
        - info
        - config
      - commit
    time: 2012-11-12 10:32:33.594720 -05:00
```

- The following sample log message shows that the Puppet agent requested the specified update to the configuration. The message only displays the configuration differences.

```
  - !ruby/object:Puppet::Util::Log
    level: !ruby/sym notice
    message: "\n[edit interfaces ge-0/0/20 unit 0 family ethernet-switching]\n+
    native-vlan-id Pink;"
    source: JUNOS
    tags:
      - notice
      - config
      - changes
    time: 2012-11-12 10:32:33.877671 -05:00
```

- The following sample debug log message shows that the Puppet agent requested the specified update to the configuration. This is the same configuration request as in the previous example, but in this case, the message displays the configuration data using XML format. To generate log messages with a severity level of debug, you must specify the **--debug** option when you run the Puppet agent.

```
  - !ruby/object:Puppet::Util::Log
    level: !ruby/sym debug
    message: |-
      <configuration>
        <interfaces>
          <interface>
            <name>ge-0/0/20</name>
            <unit>
              <name>0</name>
              <family>
                <ethernet-switching>
                  <native-vlan-id>Pink</native-vlan-id>
                </ethernet-switching>
              </family>
            </unit>
          </interface>
        </interfaces>
      </configuration>
    source: JUNOS
    tags:
      - debug
      - config
      - changes
    time: 2012-11-12 10:32:33.597816 -05:00
```

- The following sample log message shows a successful commit operation on the agent node:

```
- !ruby/object:Puppet::Util::Log
  level: !ruby/sym notice
  message: "OK: COMMIT success!"
  source: JUNOS
  tags:
    - notice
    - config
    - success
  time: 2012-11-12 10:32:38.945565 -05:00
```

## Troubleshooting Puppet for Junos OS Errors

**IN THIS SECTION**

The following sections outline errors that you might encounter when using Puppet to manage devices running Junos OS. These sections also present potential causes and solutions for each error.

### Troubleshooting Junos OS Configuration Exclusive Lock Errors

**Problem**

**Description:** The Puppet agent cannot obtain an exclusive lock on the configuration. Thus, the dependency on the **netdev_device** fails causing the Puppet agent to skip configuration updates for all netdev resources.

**Cause**

Another user currently has the exclusive lock on the candidate configuration or is modifying the configuration.

The following sample error output indicates that the configuration database is locked by another user:

```
err: JUNOS: configuration database locked by:
  jeremy terminal p0 (pid 1469) on since 2012-11-12 15:57:29 UTC
      exclusive {master:0}[edit]

err: /Stage[main]/Database_switch/Netdev_device[jd]: Could not evaluate: Unable
to obtain Junos configuration exclusive lock
notice: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/11]: Dependency
Netdev_device[jd] has failures: true
warning: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/11]: Skipping
because of failed dependencies
notice: /Stage[main]/Database_switch/Netdev_vlan[Yellow]: Dependency
Netdev_device[jd] has failures: true
warning: /Stage[main]/Database_switch/Netdev_vlan[Yellow]: Skipping because of
failed dependencies
notice: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/1]: Dependency
Netdev_device[jd] has failures: true
warning: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/1]: Skipping
because of failed dependencies
notice: /Stage[main]/Database_switch/Netdev_vlan[Blue]: Dependency Netdev_device[jd]
 has failures: true
<…output omitted…>
```

The following sample error output indicates that the configuration database has modifications in progress:

```
err: JUNOS: configuration database modified

err: /Stage[main]/Database_switch/Netdev_device[jd]: Could not evaluate: Unable
to obtain Junos configuration exclusive lock
notice: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/11]: Dependency
Netdev_device[jd] has failures: true
warning: /Stage[main]/Database_switch/Netdev_l2_interface[ge-0/0/11]: Skipping
because of failed dependencies
<…output omitted…>
```

**Solution**

Wait until the lock on the configuration is released. When the Puppet agent retrieves the configuration and can obtain an exclusive lock on the configuration database, the agent updates the system configuration accordingly.

## Troubleshooting Junos OS Configuration Load Errors

**Problem**

**Description:** The Puppet agent is unable to load the requested configuration changes into the candidate configuration.

**Cause**

The configuration change might contain invalid syntax, elements, or values.

The following sample error output indicates that the Puppet agent attempted to set the VLAN ID to 9999, which is out of the accepted range of 1 through 4094:

```
notice: /Stage[main]//Node[jd.example.com]/Netdev_vlan[Bad_VLAN]/ensure: created
info: JUNOS: Committing 1 changes.
err: JUNOS: ERROR: Configuration change
<?xml version="1.0"?>
<load-configuration-results>
  <rpc-error>
    <error-severity>error</error-severity>
    <error-info>
      <bad-element>9999</bad-element>
    </error-info>
    <error-message>Value 9999 is not within range (1..4094)</error-message>
  </rpc-error>
  <load-error-count>1</load-error-count>
</load-configuration-results>
```

**Solution**

Correct the corresponding Puppet manifest file so that it contains valid configuration changes for the agent node.

## Troubleshooting Junos OS Configuration Commit Errors

**Problem**

**Description:** The Puppet agent is unable to commit the requested configuration changes.

**Cause**

The configuration change might contain invalid syntax, elements, or values.

The following sample error output indicates that the Puppet agent attempted to associate an interface with a nonexistent VLAN:

```
notice:
/Stage[main]//Node[jd.example.com]/Netdev_l2_interface[ge-0/0/21]/description:
description changed '' to 'Puppet created netdev_l2_interface: ge-0/0/21'
notice:
/Stage[main]//Node[jd.example.com]/Netdev_l2_interface[ge-0/0/21]/untagged_vlan:
untagged_vlan changed '' to 'I_do_not_exist'
info: JUNOS: Committing 1 changes.
notice: JUNOS:
[edit interfaces ge-0/0/21 unit 0]
+    description "Puppet created netdev_l2_interface: ge-0/0/21";
[edit interfaces ge-0/0/21 unit 0 family ethernet-switching]
+       vlan {
+           members I_do_not_exist;
+       }
err: JUNOS: ERROR: Configuration change
<?xml version="1.0"?>
<commit-results>
  <rpc-error>
    <error-severity>error</error-severity>
    <source-daemon>eswd</source-daemon>
    <error-message>Interface <ge-0> vlan member <i_do_not_exist>
undefined</i_do_not_exist></ge-0></error-message>
  </rpc-error>
  <rpc-error>
    <error-severity>error</error-severity>
    <error-message>configuration check-out failed</error-message>
  </rpc-error>
</commit-results>
```

**Solution**

Correct the corresponding Puppet manifest file so that it contains valid configuration changes for the agent node.

## Troubleshooting Junos OS Configuration Errors

**Problem**

**Description:** The log files indicate that the Puppet agent successfully committed the configuration, but the agent node does not reflect the desired configuration changes.

#### Cause

There can be multiple reasons why the agent node does not reflect the correct configuration.

- The Puppet manifest contains incorrect configuration information.
- The Puppet agent has not yet performed the configuration update for the latest catalog.

  To verify that the Puppet agent has downloaded and committed a specific catalog, issue the **show system commit** operational mode command on the agent node running Junos OS to view the commit history and catalog versions.

  `root@jd.example.com>` **show system commit**

```
0   2013-01-29 10:50:17 EST by puppet via netconf
    Puppet agent catalog: 1359474609
1   2013-01-29 10:49:54 EST by root via cli
2   2013-01-29 10:48:00 EST by puppet via netconf
    Puppet agent catalog: 1359474408
3   2013-01-29 10:47:37 EST by root via cli
4   2013-01-29 10:46:57 EST by puppet via netconf
    Puppet agent catalog: 1359474408
```

#### Solution

If the Puppet manifest file contains incorrect configuration changes, correct the file to include the desired configuration changes for the agent node.

If the Puppet agent has not yet installed the changes in the latest catalog, wait until the update is made and then verify the configuration.

### Troubleshooting Agent Errors on an EX4300 Switch

#### Problem

**Description:** On an EX4300 switch, the Puppet agent reports errors during a run which involves configuring a large number of number of VLANs. For example, you might see a “Could not send report” or "Could not run: failed to allocate memory" message.

#### Cause

Memory limitation on EX4300 devices.

#### Solution

Divide the VLAN configuration across multiple manifest files and apply each manifest file in a separate Puppet agent run.

For example, suppose you have 1024 VLANs. You can split the VLAN configuration across four manifest files (vlan1.pp, vlan2.pp, vlan3.pp, and vlan4.pp) so that each manifest file contains configuration for 256

VLANs. Then run the Puppet agent four times, changing the node definition in the main manifest file as follows on each agent run:

- First agent run:

```
node <node-name> {
   netdev_device { $hostname: }
   import 'vlan1'
}
```

- Second agent run:

```
node <node-name> {
   netdev_device { $hostname: }
   import 'vlan1'
   import 'vlan2'
}
```

- Third agent run:

```
node <node-name> {
   netdev_device { $hostname: }
   import 'vlan1'
   import 'vlan2'
   import 'vlan3'
}
```

- Fourth agent run:

```
node <node-name> {
   netdev_device { $hostname: }
   import 'vlan1'
   import 'vlan2'
   import 'vlan3'
   import 'vlan4'
}
```

## Troubleshooting Connection and Certificate Errors on Puppet Clients

IN THIS SECTION

The following sections outline errors that you might encounter on Puppet clients running Junos OS. These sections also present potential causes and solutions for each error.

### Puppet Client Request Certificate Error

**Problem**

**Description:** The Puppet client generates an error that it cannot request a certificate from the Puppet master.

```
% puppet agent --test
Info: Creating a new SSL key for puppet-client.example.com
Error: Could not request certificate: Invalid argument - connect(2)
Exiting; failed to retrieve certificate and waitforcert is disabled
```

**Cause**

The Puppet master might not be running an instance of the puppet master process.

On the Puppet master, review the list of active processes to determine whether the puppet master process is running. The output should include the **puppet** process if it is already running.

`[root@puppet-master ~]#` **ps aux | grep puppet**

```
root      3328  0.0  0.0 103308   848 pts/0    S+   12:42   0:00 grep puppet
```

Alternatively, on the Puppet client, telnet to the Puppet master on port 8140. If the puppet master process is not running, the connection fails.

`%` **telnet puppet-master.example.com 8140**

```
Trying 198.51.100.1...
telnet: connect to address 198.51.100.1: Connection refused
telnet: Unable to connect to remote host
```

**Solution**

If the Puppet master is not running an instance of the puppet master process, start the process by issuing the **puppet master** command with any required options. Then verify that the process is running.

`[root@puppet-master ~]#` **puppet master *options***

`[root@puppet-master ~]#` **ps aux | grep puppet**

```
puppet    1785  0.0  4.4 437540 45028 ?        Ssl  11:21   0:01
/opt/puppet/embedded/bin/puppet
root      3328  0.0  0.0 103308   848 pts/0    S+   12:42   0:00 grep puppet
```

### Puppet Client No Certificate Found Error

**Problem**

**Description:** The Puppet client generates a **no certificate found** error and fails to download the catalog from the Puppet master.

```
Exiting; no certificate found and waitforcert is disabled
```

**Cause**

The error might indicate that the certificate for the Puppet client is not signed.

**Solution**

On the Puppet master, sign outstanding client certificate requests using the **puppet cert sign** command. For example:

`[root@puppet-master]#` **puppet cert sign puppet-client.example.com**

```
Notice: Signed certificate request for puppet-client.example.com
Notice: Removing file Puppet::SSL::CertificateRequest puppet-client.example.com
at '/var/lib/puppet/ssl/ca/requests/puppet-client.example.com'
```

See the official Puppet documentation for detailed information about Puppet commands.

## RELATED DOCUMENTATION

Setting Up the Puppet Configuration File on the Puppet Master and Puppet Agents Running Junos OS | **32**

Puppet Manifests for Devices Running Junos OS | **37**