

Junos Platform Automation and DevOps

STUDENT GUIDE—Volume 2 of 2

Revision V-17.a

Education Services Courseware

Junos Platform Automation and DevOps

V-17.a

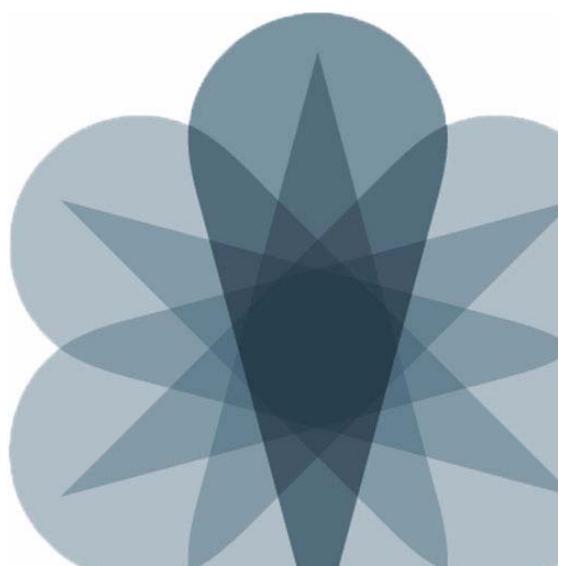
Student Guide
Volume 2



Worldwide Education Services

1133 Innovation Way
Sunnyvale, CA 94089
USA
408-745-2000
www.juniper.net

Course Number: EDU-JUN-JAUT



This document is produced by Juniper Networks, Inc.

This document or any part thereof may not be reproduced or transmitted in any form under penalty of law, without the prior written permission of Juniper Networks Education Services.

Juniper Networks, Junos, Steel-Belted Radius, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo, the Junos logo, and JunosE are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Junos Platform Automation and DevOps Student Guide, Revision V-17.a

Copyright © 2017 Juniper Networks, Inc. All rights reserved.

Printed in USA.

Revision History:

Revision 11.a—September 2011.

Revision 14.a—August 2015.

Revision V-17.a—June 2017.

The information in this document is current as of the date listed above.

The information in this document has been carefully verified and is believed to be accurate for software JJunos OS Release 17.1R1, PyEZ 2.0, Python 2.7, and Ansible 2.3. Juniper Networks assumes no responsibilities for any inaccuracies that may appear in this document. In no event will Juniper Networks be liable for direct, indirect, special, exemplary, incidental, or consequential damages resulting from any defect or omission in this document, even if advised of the possibility of such damages.

Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

YEAR 2000 NOTICE

Juniper Networks hardware and software products do not suffer from Year 2000 problems and hence are Year 2000 compliant. The Junos operating system has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

SOFTWARE LICENSE

The terms and conditions for using Juniper Networks software are described in the software license provided with the software, or to the extent applicable, in an agreement executed between you and Juniper Networks, or Juniper Networks agent. By using Juniper Networks software, you indicate that you understand and agree to be bound by its license terms and conditions. Generally speaking, the software license restricts the manner in which you are permitted to use the Juniper Networks software, may contain prohibitions against certain uses, and may state conditions under which the license is automatically terminated. You should consult the software license for further details.

Contents

Chapter 9: Junos Automation Commit and Op Scripts.....	9-1
Junos Automation Scripts Overview	9-3
Junos OS Commit Scripts	9-20
Junos Op Scripts	9-35
Lab: Creating Commit and Op Scripts	9-56
Chapter 10: Junos Automation Event and SNMP Scripts.....	10-1
Introduction to Event Scripts	10-3
Junos OS Event Policies	10-11
Junos OS Event Scripts	10-32
Junos OS SNMP Scripts	10-39
Lab: Junos Event Policies and Event Scripts	10-50
Chapter 11: YANG.....	11-1
YANG Overview	11-3
YANG Modules	11-9
YANG Statements and Syntax	11-18
Chapter 12: OpenConfig.....	12-1
OpenConfig Overview	12-3
OpenConfig Installation	12-10
Using OpenConfig	12-16
OpenConfig Telemetry	12-29
Lab: Implementing OpenConfig	12-38
Chapter 13: Junos Extension Toolkit	13-1
Overview of JET	13-3
Creating Signed JET Applications	13-11
Creating Unsigned JET Apps	13-36
Creating JET Notification Apps	13-46
Chapter 14: The Junos OS REST API	14-1
The Junos OS REST API Overview	14-3
Installing the Junos OS REST API	14-7
Using The Junos OS REST API	14-11
Agenda: The Junos OS REST API	14-15
Lab: Implementing the Junos OS REST API	14-24
Appendix A: Zero Touch Provisioning (ZTP).....	A-1
Understanding Zero Touch Provisioning	A-3
ZTP in Action: A Working Example	A-14
Acronym List	ACR-1

Course Overview

This five-day course provides students with knowledge of how to automate Junos using DevOps automation tools, protocols, and technologies. Students receive hands-on development experience with tools and languages relevant to automating the Junos OS platform in a DevOps environment. The course includes an introduction to the Junos XML API, and NETCONF but focuses on using Python, PyEZ, and Ansible to automate Junos. The course introduces students to Junos commit, operation (op), event, and SNMP scripts. JSON, YAML, and Jinja2 are introduced as these languages facilitate Junos automation. The course also introduces the Junos Extension Toolkit and related APIs. Finally, the course discusses the use of JSANPy and Junos ZTP autoinstallation tools. Through demonstrations and hands-on labs, students will gain experience in automating the Junos operating system and device operations.

This course uses Junos OS Release 17.1R1, PyEZ 2.0, Python 2.7, and Ansible 2.3.

Course Level

Junos Platform Automation (JAUT) is an intermediate-level course.

Intended Audience

This course benefits individuals responsible for configuring and monitoring devices running the Junos OS.

Prerequisites

Students should have intermediate-level networking knowledge and an understanding of the Open Systems Interconnection (OSI) model and the TCP/IP protocol suite. Students should also have familiarity with XML basics and have introductory knowledge of the Python programming language. Students should also attend the *Introduction to the Junos Operating System* (IJOS) course prior to attending this class. Lastly, a high-level understanding of object-oriented programming is a plus, but not a requirement.

Objectives

After successfully completing this course, you should be able to:

- Describe the NETCONF protocol.
- Explain the capabilities of the Junos OS XML API.
- Describe the use of XSLT, SLAX, and XPath in the XML API.
- Describe the Junos Automation UI and explain
 - the role of gRPC, NETCONF, and REST in Junos Automation.
- Identify the languages, frameworks, management suites, and tools used in automating Junos.
- Describe the YANG Protocol and explain the capabilities of YANG.
- Use the YANG model to issue Junos commands and to configure Junos.
- Explain the benefits of using JSON and YAML.
- List where JSON and YAML are used in Junos Automation.
- Convert between JSON, YAML, and XML.
- Describe the features and benefits of using Python in Junos automation.
- Configure Junos devices to use Python and create simple Python scripts.
- Describe the function of Junos operation, commit, event, and SNMP scripts.
- Implement Junos operation, commit, event, and SNMP scripts using Python.
- Identify how Junos automation uses Jinja2 and create Python scripts that use Jinja2.
- Explain how PyEZ makes Junos automation easier
- Use PyEZ to gather facts from Junos, perform configuration tasks, and use PyEZ to manipulate the file system and perform system upgrades to Junos.
- Implement OpenConfig in the Junos OS.
- Describe the process of implementing custom YANG modules.

- Implement a translation script for a custom YANG module.
- Explain the use of the Junos REST API in automation.
- Use the Junos REST API to get information from Junos.
- Describe what JET is and what it includes.
- Create a project in the JET IDE.
- Execute scripts using on-box and off-box automation.
- Describe how Ansible is used in Junos automation and install Ansible.
- Create Ansible playbooks to automate Junos.
- Describe how JSNAPy can help automate Junos devices.
- Implement JSNAPy into a Junos environment.
- Describe how ZTP works.
- Configure in-band ZTP and out-of-band ZTP.

Course Agenda

Day 1

- Chapter 1: Course Introduction
- Chapter 2: Junos Automation Architecture and Overview
- Chapter 3: NETCONF and the XML API
 - Lab 1: Exploring the XML API

Day 2

- Chapter 4: JSON and YAML
 - Lab 2: Using JSON and YAML
- Chapter 5: Python and Junos PyEZ
 - Lab 3: Implementing Python and Junos PyEZ in Junos
- Chapter 6: Jinja2 and Junos PyEZ
 - Lab 4: Using Jinja2 Templates with PyEZ

Day 3

- Chapter 7: Using Ansible to Automate Junos
 - Lab 5: Automating Junos with Ansible
- Chapter 8: Junos Automation with JSNAPy
 - Lab 6: Using JSNAPy

Day 4

- Chapter 9: Junos OS Commit and Op Scripts
 - Lab 7: Junos Commit and Op Scripts
- Chapter 10: Junos OS Event and SNMP Scripts
 - Lab 8: Junos Event Policies and Scripts
- Chapter 11: YANG
- Chapter 12: OpenConfig
 - Lab 9: Implementing OpenConfig

Day 5

- Chapter 13: Junos Extension Toolkit (JET)
- Chapter 14: The Junos OS REST API
 - Lab 10: Implementing the Junos REST API
- Appendix A: Zero Touch Provisioning

Document Conventions

CLI and GUI Text

Frequently throughout this course, we refer to text that appears in a command-line interface (CLI) or a graphical user interface (GUI). To make the language of these documents easier to read, we distinguish GUI and CLI text from chapter text according to the following table.

Style	Description	Usage Example
Franklin Gothic	Normal text.	Most of what you read in the Lab Guide and Student Guide.
Courier New	<p>Console text:</p> <ul style="list-style-type: none">• Screen captures• Noncommand-related syntax <p>GUI text elements:</p> <ul style="list-style-type: none">• Menu names• Text field entry	<p>commit complete</p> <p>Exiting configuration mode</p> <p>Select File > Open, and then click Configuration.conf in the Filename text box.</p>

Input Text Versus Output Text

You will also frequently see cases where you must enter input text yourself. Often these instances will be shown in the context of where you must enter them. We use bold style to distinguish text that is input versus text that is simply displayed.

Style	Description	Usage Example
Normal CLI	No distinguishing variant.	Physical interface:fxp0, Enabled
Normal GUI		View configuration history by clicking Configuration > History.
CLI Input	Text that you must enter.	lab@San_Jose> show route
GUI Input		Select File > Save, and type config.ini in the Filename field.

Undefined Syntax Variables

Finally, this course distinguishes syntax variables, where you must assign the value (undefined variables). Note that these styles can be combined with the input style as well.

Style	Description	Usage Example
<u>CLI Undefined</u>	Text where the variable's value is the user's discretion or text where the variable's value as shown in the lab guide might differ from the value the user must input according to the lab topology.	Type set policy <u>policy-name</u>. ping 10.0.<u>x.y</u>
<u>GUI Undefined</u>		Select File > Save, and type <u>filename</u> in the Filename field.

Additional Information

Education Services Offerings

You can obtain information on the latest Education Services offerings, course dates, and class locations from the World Wide Web by pointing your Web browser to: <http://www.juniper.net/training/education/>.

About This Publication

This course was developed and tested using the software release listed on the copyright page. Previous and later versions of software might behave differently so you should always consult the documentation and release notes for the version of code you are running before reporting errors.

This document is written and maintained by the Juniper Networks Education Services development team. Please send questions and suggestions for improvement to training@juniper.net.

Technical Publications

You can print technical manuals and release notes directly from the Internet in a variety of formats:

- Go to <http://www.juniper.net/techpubs/>.
- Locate the specific software or hardware release and title you need, and choose the format in which you want to view or print the document.

Documentation sets and CDs are available through your local Juniper Networks sales office or account representative.

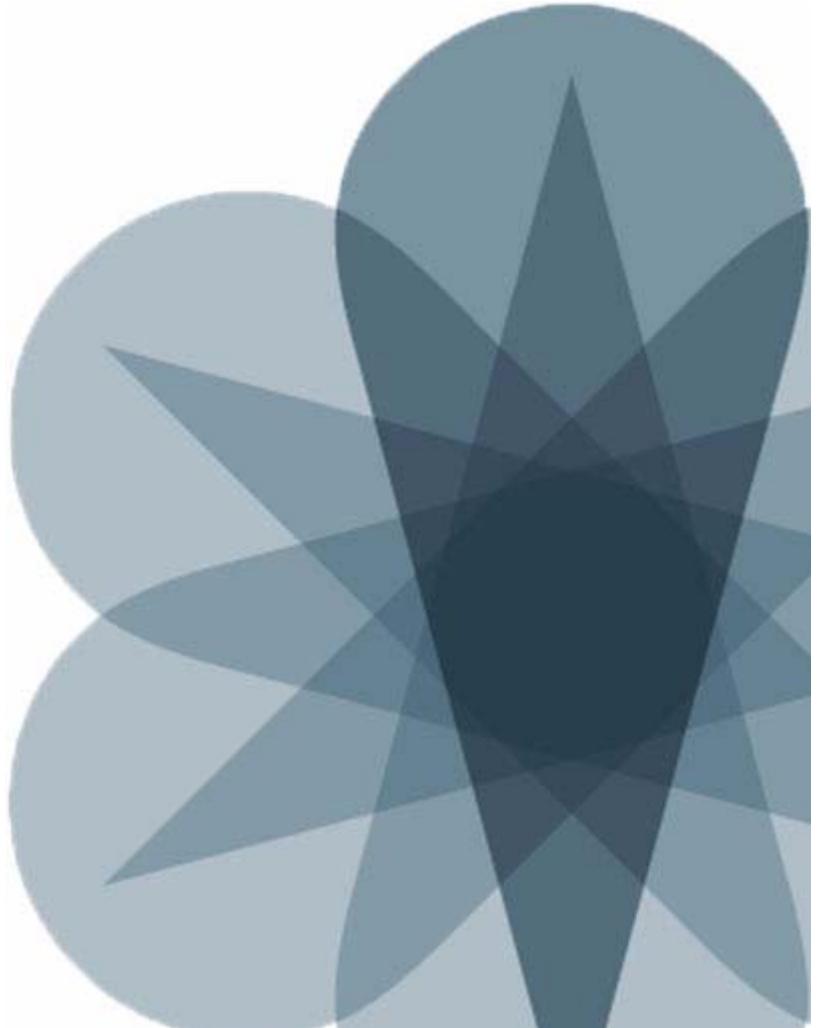
Juniper Networks Support

For technical support, contact Juniper Networks at <http://www.juniper.net/customers/support/>, or at 1-888-314-JTAC (within the United States) or 408-745-2121 (outside the United States).



Junos Platform Automation and DevOps

Chapter 9: Junos Automation Commit and Op Scripts



Objectives

- After successfully completing this content, you will be able to:
 - Create Junos OS commit scripts
 - Create Junos OS op scripts

We Will Discuss:

- Creating Junos OS commit scripts; and
- Creating Junos OS op scripts.

Agenda: Junos OS Commit and Op Scripts

- Junos Automation Scripting Overview
- Junos OS Commit Scripts
- Junos OS Op Scripts

© 2017 Juniper Networks, Inc. All rights reserved.

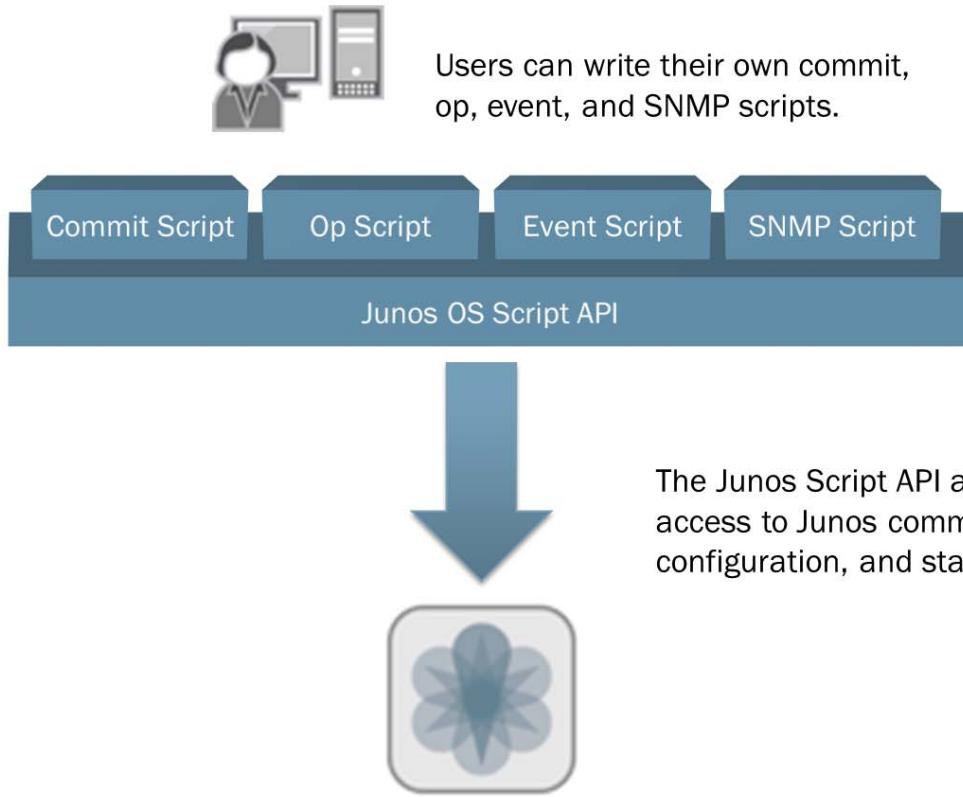
 Worldwide Education Services

www.juniper.net | 3

Junos Automation Scripts Overview

The slide lists the topics we will discuss. We discuss the highlighted topic first.

Junos OS Script API



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

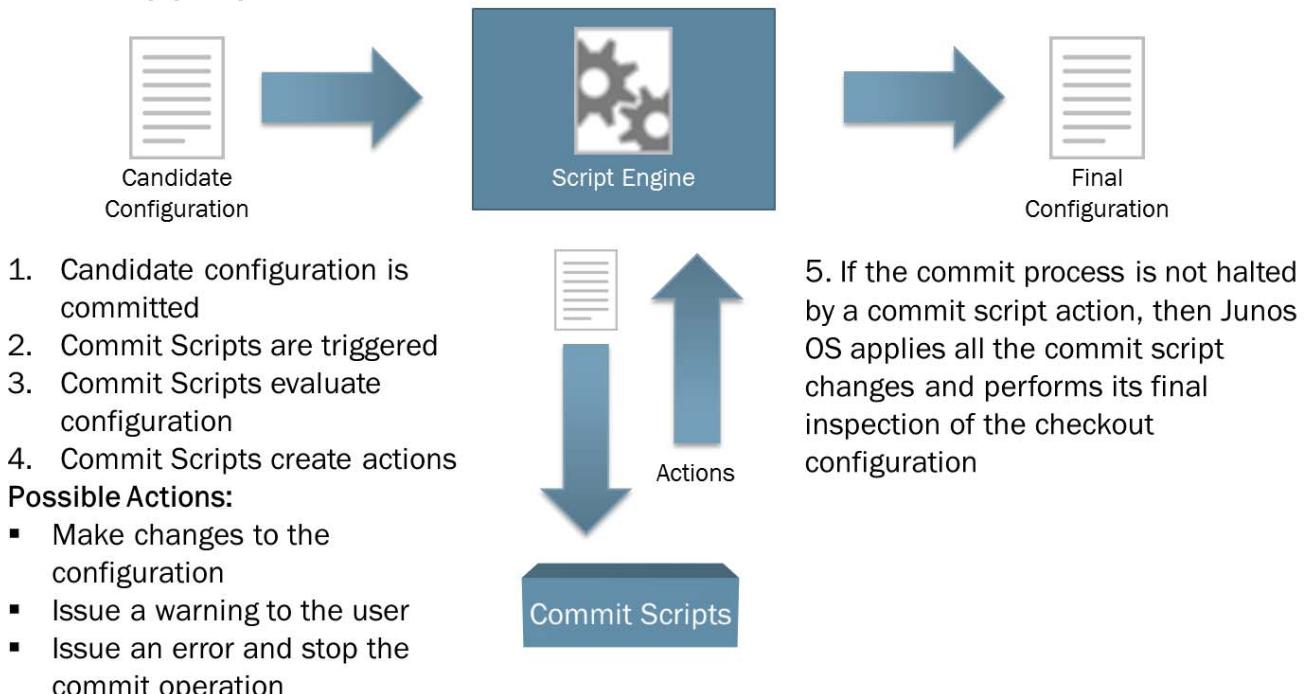
www.juniper.net | 4

Junos OS Script API

This chapter focuses on using Python as a method to automate Junos OS. Junos OS supports four kinds of scripts—commit, op, event, and SNMP. These scripts use the Junos Script application programming interface (API). All four script types are capable of leveraging the Junos XML API. Because it uses a documented API, anyone can write their own commit, op, event, and SNMP scripts. You will use one of these four types of scripts when automating Junos OS. In the next few pages, you will see the different ways you can use scripts. You will focus on Junos automation commit and op scripts in this chapter, and on SNMP and event scripts in the next chapter.

Commit Script Processing Overview

Commit scripts evaluate the candidate configuration and take appropriate action



Commit Script Processing Overview

Commit scripts are used to evaluate the candidate configuration and take appropriate action. When a user attempts to commit a candidate configuration, the script engine processes the candidate configuration through each configured commit script. The script engine uses the actions each commit script produces and creates a final configuration.

Commit Script Example (1 of 2)

- Sometimes you commit bad code

```
[edit]
lab@SaoPaulo# set protocols bgp group core family inet-vpn any
A configuration change is made and
[edit] committed the configuration successfully.

[edit]
lab@SaoPaulo# commit and-quit
commit complete
Exiting configuration mode

lab@SaoPaulo> Unfortunately, this configuration causes some serious errors.
Feb 23 14:14:31 SaoPaulo rpd[4518]: bgp_process_caps: mismatch NLRI with 172.16.1.2
(Internal AS 65000): peer: <inet-unicast>(1) us: <inet-vpn-unicast inet-vpn-multicast>(12)
Feb 23 14:14:31 SaoPaulo rpd[4518]: bgp process caps:2463: NOTIFICATION sent to 172.16.1.2
(Internal AS 65000): code 2 (Open Message Error) subcode 7 (unsupported capability) value 12
Feb 23 14:14:39 SaoPaulo sshd: sendmsg to 10.210.210.23 failed: No route to host
Feb 23 14:14:39 SaoPaulo sshd: rad_send_request: Tried all servers unsucessfully
Feb 23 14:15:03 SaoPaulo rpd[4518]: bgp_process_caps: mismatch NLRI with 172.16.1.2
(Internal AS 65000): peer: <inet-unicast>(1) us: <inet-vpn-unicast inet-vpn-multicast>(12)
Feb 23 14:15:03 SaoPaulo rpd[4518]: bgp process caps:2463: NOTIFICATION sent to 172.16.1.2
(Internal AS 65000): code 2 (Open Message Error) subcode 7 (unsupported capability) value 12
Feb 23 14:15:28 SaoPaulo sshd: sendmsg to 10.210.210.23 failed: No route to host
Feb 23 14:15:28 SaoPaulo sshd: rad_send_request: Tried all servers unsucessfully
Feb 23 14:15:35 SaoPaulo rpd[4518]: bgp_process_caps: mismatch NLRI with 172.16.1.2
(Internal AS 65000): peer: <inet-unicast>(1) us: <inet-vpn-unicast inet-vpn-multicast>(12)
```

Outage Caused by Configuration Change

Previously, you could not have a device check a configuration to ensure it was appropriate for your environment before committing. In the example shown on the slide, a configuration change has been committed. However, the error messages show that the router can no longer reach its authentication server; it is isolated from at least parts of the network. This particular configuration caused serious problems in the router environment. The next slide illustrates how to test configurations before committing to avoid such outages.

Commit Script Example (2 of 2)

- Use a commit script to check the configuration

```
[edit]
lab@SaoPaulo# set protocols bgp group core family inet-vpn any
                  Make a configuration change and attempt to commit.

[edit]
lab@SaoPaulo# commit
[edit protocols bgp group core family]
  Incorrect family configuration.
error: 1 error reported by commit scripts
error: commit script failure
```

The commit script issues an error and does not allow the configuration to be committed.

Add Configuration Checks to the Commit Procedure

Now, however, you can use a commit script to check the configuration and ensure that it is appropriate for your environment. If it is not, you can have the commit script issue a warning, or issue an error and abort the configuration. Here, we commit the same inappropriate configuration. This time, the commit script issues an error and does not allow the configuration to be committed.

Pass Variables to Commit Script

With the **apply-macro** command, you can shrink this configuration dramatically!



Which would you rather type?!

```

interfaces {
    so-0/1/1 {
        unit 0 {
            family inet {
                rpf-check;
                filter {
                    input classify-gold;
                }
                address 10.0.22.1/30;
            }
        }
    }
}

interfaces {
    so-0/1/1 {
        disable;
        unit 0 {
            apply-macro customer-interface {
                asn 65111;
                cos gold;
                customer-id ab21332;
                routes full-routes;
                v4-address 10.0.22.1/30;
            }
        }
    }
}

customer-standard;
fiers {
    cp default;
}

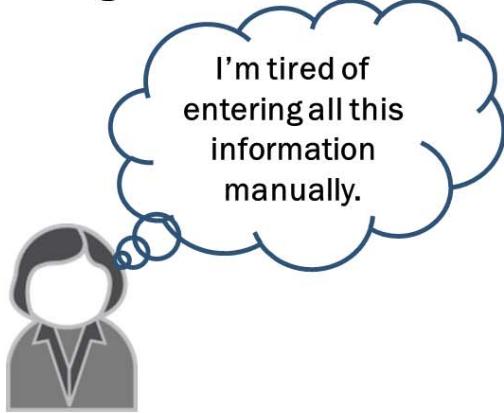
```

Pass Variables to Commit Script

The list of Junos Platform Automation solutions is vast; you can use the `apply-macro` command to pass parameters to commit scripts. You can then program the knowledge about the configuration template into the commit script and let it create the correct configuration. You can even program Junos OS to expand the configuration in the background so that it doesn't clutter up the candidate configuration.

Autogenerate Configuration

Many companies provision new links with the same template configuration.



Junos Platform Automation allows you instruct Junos to simply “do the right thing” with the new link.

```

interfaces {
    so-0/1/1 {
        unit 0 {
            family inet {
                rpf-check;
                filter {
                    input classify-gold;
                }
                address 10.0.22.1/30;
            }
        }
    }
}
protocols {
    bgp {
        group full-routes {
            export full-routes;
            neighbor 10.0.22.2 {
                import ab21332_65111;
                peer-as 65111;
            }
        }
    }
}
class-of-service {
    interfaces {
        so-0/1/1 {
            scheduler-map customer-standard;
            unit 0 {
                classifiers {
                    dscp default;
                }
            }
        }
    }
}

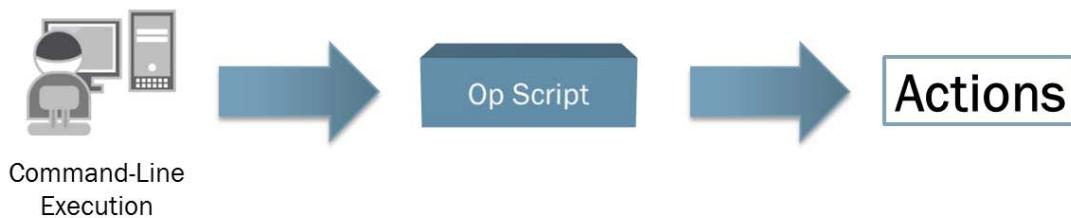
```

Autogenerate Configuration

Here's another example: Many companies provision new links with the same template configuration. Frequently, portions of the configuration go in several places: interior gateway protocol (IGP), BGP, MPLS, RSVP, and so on. Currently, you may have to enter this information manually. But what if you make a typo? These errors might create a huge problem. Junos Platform Automation, however, allows you instruct Junos OS to simply “do the right thing” with the new link.

Op Script Processing

- Op scripts allow you to add your own commands to the operational-mode CLI



- Op script Example:

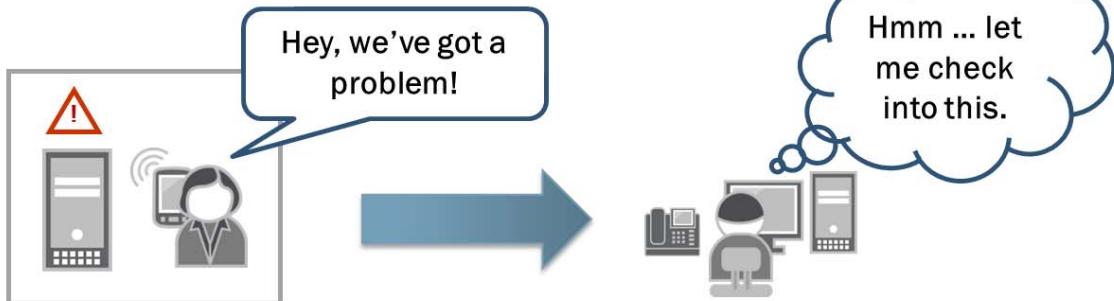
```
lab@SaoPaulo> op terse-int customer-code XCDT
Interface          Admin Link Proto    Local           Remote
so-0/1/2            up     up
so-0/1/2.501        up     up     inet      172.31.31.1/30
```

Op Script Processing

Op scripts allow you to add your own commands to the operational-mode command-line interface (CLI). As is done with other CLI commands, you can add arguments to and produce output using these commands. However, unlike other CLI commands in Junos OS, these commands can be customized. For instance, the custom command shown on the slide may be used to view interface information based on the customer code.

Troubleshooting with Op Scripts (1 of 2)

- Troubleshooting before Op scripts



Standard operating procedure

Run the following commands on the problem interface:

- ping
- show interface extensive
- show bgp neighbor

Standard Troubleshooting

Another way you can use OP scripts is for troubleshooting. Many businesses have routine commands they run in response to certain situations. For example, when a user calls with a problem, standard operating procedure may be to run `ping` and determine first if the customer device is pingable, then run `show interfaces extensive` to look for errors, and then run `show bgp neighbor` to determine whether the BGP session is up.

Troubleshooting with Op Scripts (2 of 2)

- One command does it all

```
lab@SaoPaulo> op diag-interface interface so-0/1/2.501
```

Ping Results:

```
Pinging 172.31.31.2 with 5 56-byte packets.  
0 of 5 successful. Packet loss: 100%.  
Failure reported: no response.
```

Interface information:

```
Logical interface so-0/1/2.501 (Index 71) (SNMP ifIndex 133)  
(Generation 136)
```

```
Description: XCDT-blah
```

```
Flags: Point-To-Point SNMP-Traps Encapsulation: FR-NLPID
```

```
Traffic statistics:
```

BGP information:

```
Peer: 172.31.31.2+179 AS 65113 Local: 172.31.31.1+64562 AS 65000
```

```
Type: External State: Established Flags: <Sync>
```

```
Last State: OpenConfirm Last Event: RecvKeepAlive
```

```
Last Error: None
```

```
Options: <Preference LogUpDown PeerAS Refresh>
```

```
Options: <>
```

```
Holddown: 90 Preference: 170 [...]
```

One Command Does it All

Instead of having a technician run each of these commands individually, the technician can execute a single script that runs the correct commands and then highlights the vital data.

Event Script Processing

- Event scripts are triggered by events on the device



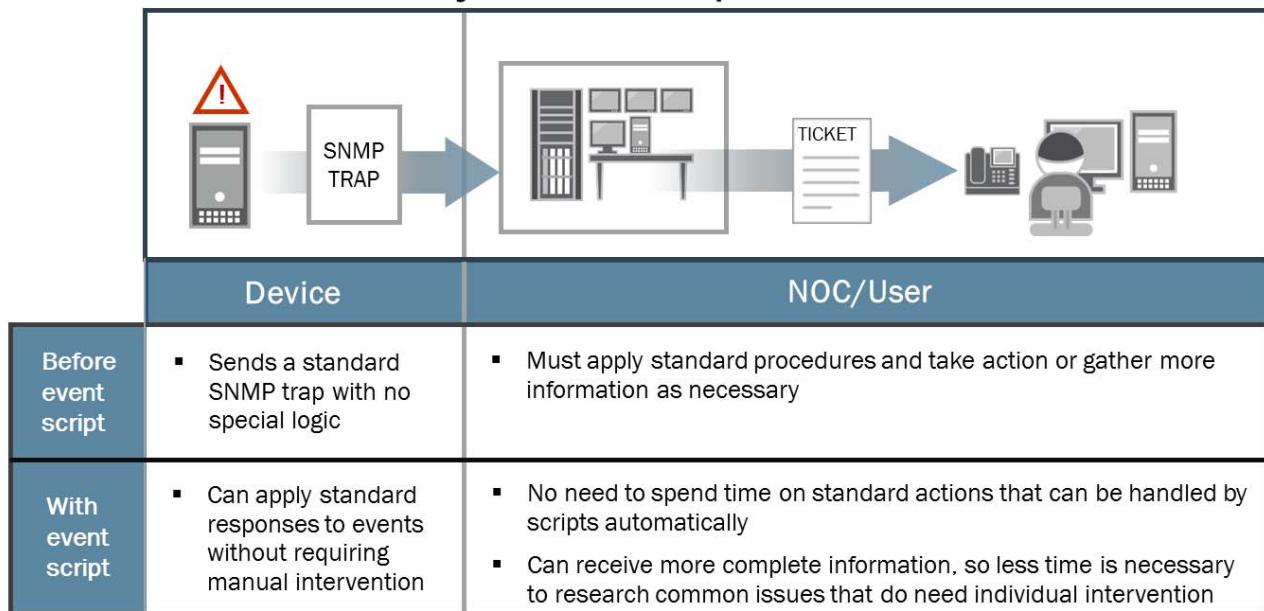
- When an event occurs, you can have an event script...
 - Perform further troubleshooting to decide whether action is needed prior to sending a warning to your NOC
 - When appropriate, take action to attempt to correct an error

Event Script Processing

Events scripts are related to op scripts, but they are triggered by events on the device. The log output on the slide shows an event script being triggered because a link is down.

Event Script Solutions

- You can use on box Junos Platform Automation to customize the way Junos responds to events



Event Script Solutions

Again, many companies have standard procedures for dealing with events that are reported through monitoring software. Previously, users would receive a standard SNMP trap with no special logic applied. The user would need to apply their standard company procedures to follow up.

Sometimes those responses involve taking immediate action; other times the required response is to gather more data to determine the severity of a problem. Event scripts allow you to add intelligence to Junos OS so that it automatically applies these standard responses to events without requiring manual intervention.

You can have event scripts run any operational-mode command, change the configuration, or send a further notification through syslog or SNMP. When the user does receive a notification, the user does not need to spend time executing standard actions because they have already been handled by the script. Event scripts can provide the user with more complete information so that less time is needed to research complex issues.

If you have trouble keeping the three types of scripts separate and knowing which is appropriate to use when, think about what triggers each to execute. Commit scripts execute once a configuration is committed; op scripts execute when a user runs the script; and event scripts are triggered by a specific event occurring on the device.

Now that you understand how Junos OS commit, OP, and event scripts are used, you will move on to the capabilities of Python.

SNMP Script Processing

- SNMP Scripts are triggered by an SNMP manager calling an SNMP agent
- SNMP Scripts are used to create and handle custom SNMP OIDs



SNMP Script Processing

If you monitor and manage your network through SNMP, you can create scripts that are triggered by unhandled SNMP OIDs (Simple Network Management Protocol Object Identifiers). Unhandled OIDs as you will experience them are OIDs that Juniper has not yet defined. Your SNMP manager can request information for a unhandled OID; when Junos OS receives the request, it matches it to a Junos OS SNMP script, then retrieves and returns the relevant information.

Executing Unsigned Python Scripts (1 of 4)

- Junos OS does not execute unsigned scripts by default
- Additional steps to executing unsigned scripts
 - Set language statement
 - Enable individual scripts
 - Store script in appropriate directory
 - Script must meet permission requirements

Executing Unsigned Python Scripts: Part 1

Your scripts will most often be running in your development environment while in development, but once your scripts are finished, they can be moved to the device running Junos OS, where they are stored and executed locally.

By default, you cannot execute unsigned Python scripts on devices running Junos OS. To enable the execution of unsigned Python automation scripts, you must configure the `language python` statement, as seen in the previous slide.

You must also enable each individual script by configuring the script filename under the hierarchy level appropriate to the script. You need to store each file in specific directories depending the use of the script. Finally, the scripts need to meet permission requirements.

Each of these steps are discussed in the next three slides.

Executing Unsigned Python Scripts (2 of 4)

- Enable individual scripts

```
[edit system scripts commit]
```

```
lab@vMX-1#set file filename
```

```
[edit system scripts op]
```

```
lab@vMX-1#set file filename
```

```
[edit event-options event-script]
```

```
lab@vMX-1#set file filename
```

```
[edit system scripts snmp]
```

```
lab@vMX-1#set file filename
```

Enabling Individual scripts: Part 2

In order for scripts to be executed on a device running Junos OS, each needs to be listed individually in Junos OS configuration hierarchy. Commit scripts need to be specified in the [edit system scripts commit] hierarchy. Junos OS op, event, and SNMP scripts also need to be specified under the appropriate hierarchy.

Executing Unsigned Python Scripts (3 of 4)

- Save scripts to the appropriate location

Script Type	Hard Drive	Flash Storage
Commit	/var/db/scripts/commit	/config/scripts/commit
OP	/var/db/scripts/op	/config/scripts/op
Event	/var/db/scripts/event	/config/scripts/event
SNMP	/var/db/scripts/snmp	/config/scripts/snmp

- Add the following if storing scripts on flash storage

[edit]

```
lab@vMX-1# set system scripts load-scripts-from-
flash
```

Save Scripts To the Appropriate Location: Part 3

Python automation scripts must be stored in the appropriate directory on the device. The slide shows the appropriate location based on script type. Note that there is a separate location if you are using flash storage.

Junos OS supports using symbolic links for files in the script directories, but the device will only execute the script at the target location if it is signed.

Executing Unsigned Python Scripts (4 of 4)

- Unsigned script permission requirements

- Script owner is either *root* or *user* in Junos OS super-user login class
- Only the script owner has *write* permission for the file
- Script executer needs *read* permission for the file
- Event and SNMP scripts need to configure the *python-script-user username* statement

Unsigned Script Permission Requirements

In order for scripts to run on a device running Junos OS, there are three permission requirements:

1. The script owner either needs to be the root user or a user in the Junos OS super-user login class.
2. The only user account that can have write permission for the file is the owner of the file.
3. The user account used to execute the script needs to have execute permissions.

These permissions are strict to help keep scripts and the devices running Junos OS safe from malicious intent.

Note: Event and SNMP scripts, by default, still execute under the privileges of the user and group *nobody*. To execute the scripts using the access privileges of a specific user, you must configure the *python-script-user username* statement at the *[edit event-options event-script file filename]* hierarchy level for event scripts, or the *[edit system scripts snmp file filename]* hierarchy level for SNMP scripts, and specify a user configured at the *[edit system login]* hierarchy level.

Agenda: Junos OS Commit and Op Scripts

- Junos Automation Overview
- Junos OS Commit Scripts
- Junos OS Op Scripts

Junos OS Commit Scripts

The slide highlights the topic we discuss next.

Junos OS Commit Scripts

- Are executed whenever a device configuration is committed
- Commit scripts can:
 - Generate and display custom warning messages
 - Generate and log custom system log (syslog) messages
 - Change the configuration to conform to the custom configuration rules
 - Generate a commit error and halt the commit operation

Junos OS Commit Scripts

Junos OS commit scripts are executed when a configuration is commuted. Commit scripts can evaluate the configuration and, depending on what it finds, can generate warnings, add messages to a log file, modify the configuration, or generate an error that halts the commit operation.

Pre-Inheritance and Post-Inheritance Candidate Configurations

- Pre-Inheritance candidate configuration is the configuration before group information is applied

```
[edit]
user@host# show
```

- Post-Inheritance candidate configuration is the configuration after group information is applied

```
[edit]
user@host# show | display commit-scripts view
```

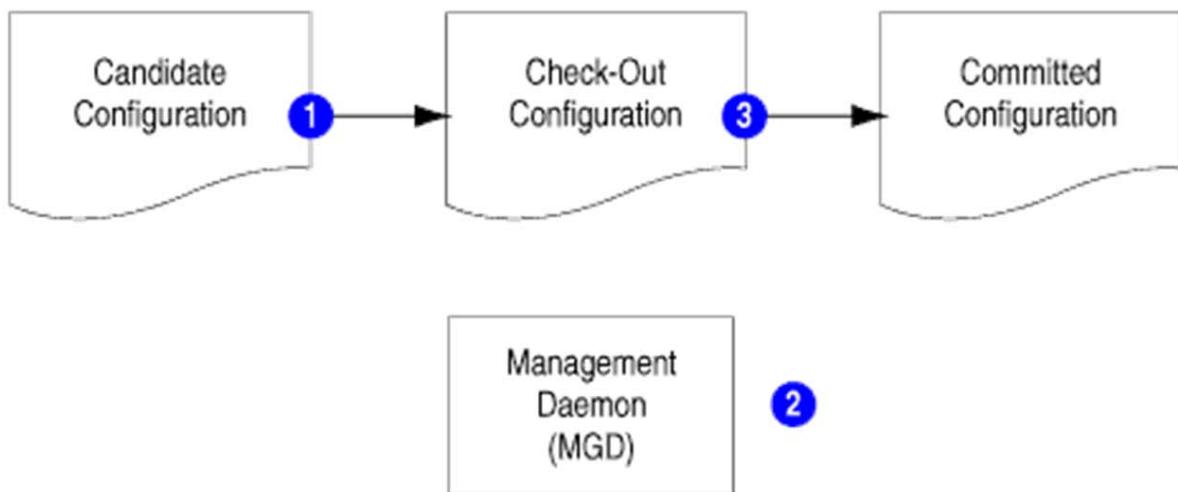
Pre-Inheritance and Post-Inheritance Candidate Configurations

When you perform a commit operation, Junos OS takes the pre-inheritance candidate configuration and merges it with any group information in the configuration. You can see the pre-inheritance configuration by issuing the `show` command at the `[edit]` hierarchy.

After the pre-inheritance configuration is merged with the group configuration information, it becomes the post-inheritance candidate configuration. You can see the post-inheritance configuration by issuing the `show | display commit-scripts view` command from the `[edit]` hierarchy, as seen in the slide.

At times, a commit script requires access to the pre-inheritance candidate configuration rather than the post-inheritance configuration it receives by default. Post-inheritance candidate configuration is submitted to the mgd process in XML format so the output of the `show | display commit-scripts view` command is also in XML format.

Standard Commit Model



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 23

The Standard Commit Model

The standard Junos OS commit model performs the following basic steps:

1. When the candidate configuration is committed, it is copied to become the checkout configuration.
2. The mgd process validates the checkout configuration.
3. If no error occurs, the checkout configuration is copied as the current active configuration.

When using commit scripts, this basic format changes. These differences will be addressed shortly.

Permanent Changes and Transient Changes

- Persistent changes are made to the candidate configuration
- Transient changes are made to the checkout configuration and will not appear in the candidate configuration
- Transient changes will not persist once the commit script is deleted or deactivated

Permanent Changes and Transient Changes

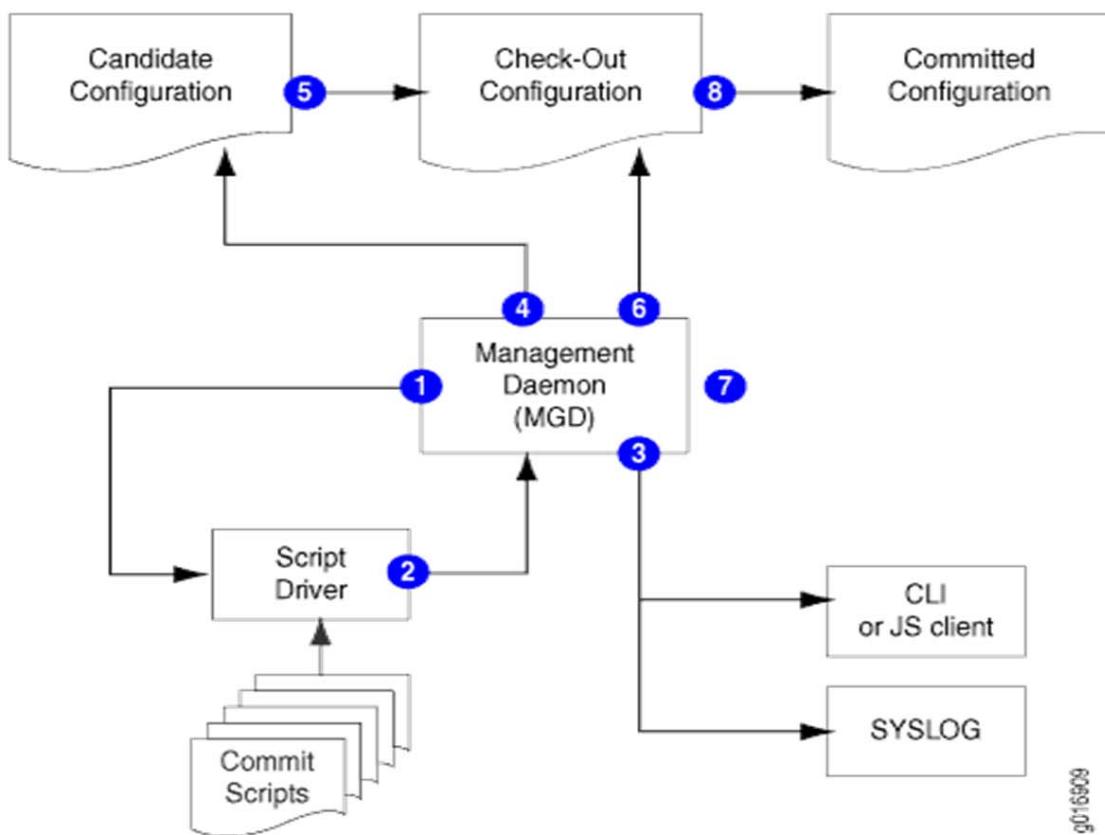
Configuration changes made by commit scripts can be persistent or transient.

A persistent change remains in the candidate configuration and affects routing operations until you explicitly delete it, even if you subsequently remove or disable the commit script that generated the change and reissue the commit command. In other words, removing the commit script does not cause a persistent change to be removed from the configuration.

A transient change, in contrast, is made in the checkout configuration but not in the candidate configuration. The checkout configuration is the configuration database that is inspected for standard Junos OS syntax just before it is copied to become the active configuration on the device. If you subsequently remove or disable the commit script that made the change and reissue the commit command, the change is no longer made to the checkout configuration, and so does not affect the active configuration. In other words, removing the commit script effectively removes a transient change from the configuration.

A common use for transient changes is to eliminate the need to repeatedly configure and display well-known policies, so, allowing these policies to be enforced implicitly. For example, if MPLS must be enabled on every interface with an International Organization for Standardization (ISO) protocol enabled, the change can be transient, so that the repetitive or redundant configuration data does not need to be carried or displayed in the candidate configuration. Transient changes allow you to write script instructions that apply the change only if a set of conditions is met.

Commit Model with Commit Scripts



901699

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 25

Commit Model with Commit Scripts

When commit scripts are added to the standard commit model, the process becomes more complex. The mgd process first passes an XML-formatted checkout configuration to a script driver, which handles the verification of the checkout configuration by the commit scripts. When verification is complete, the script driver returns an action file to the mgd process. The mgd process follows the instructions in the action file to update the candidate and checkout configurations, issue messages to the CLI or client application, and write information to the system log. After processing the action file, the mgd process performs the standard Junos OS validation.

In the commit script model, Junos OS performs the following steps:

1. When the candidate configuration is committed, the mgd process sends the XML-formatted candidate configuration to the script driver.
2. Each enabled commit script is invoked against the candidate configuration, and each script can generate a set of actions for the mgd process to perform. The actions are collected in an action file.
3. The mgd process performs the following actions for commit script `error`, `warning`, and `system log` messages in the action file:

`error`—The mgd process halts the commit process (that is, the commit operation fails), returns an error message to the CLI or Junos XML protocol client, and takes no further action.

`warning`—The mgd process forwards the message to the CLI or the Junos XML protocol client.

`system log`—The mgd process forwards the message to the system log process.

Continued on the next page.

Commit Model with Commit Scripts (contd.)

4. If the action file includes any persistent changes, the mgd process loads the requested changes into the candidate configuration.
5. The candidate configuration is copied to become the checkout configuration.
6. If the action file includes any transient changes, the mgd process loads the requested changes into the checkout configuration.
7. The mgd process validates the checkout configuration.
8. If there are no validation errors, the checkout configuration is copied to become the current active configuration.

Controlling Commit Script Execution

- Removing a commit script from the [edit system scripts commit file filename] hierarchy level stops its execution
- Include the optional statement to avoid error messages during the commit operation if a file no longer exists
- Deactivate and reactivate commit scripts by issuing the deactivate and activate configuration mode commands

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 27

Controlling Commit Script Execution

By default, the commit operation fails unless all scripts included at the [edit system scripts commit file] hierarchy level actually exist in the commit script directory. To enable the commit operation to succeed even if a script is missing, include the optional statement at the [edit system scripts commit file filename] hierarchy level.

For example, you could mark a script as optional if you anticipate a need to remove it from operation quickly by deleting it from the commit script directory; if you do not want to remove the commit script filename at the [edit system scripts commit file] hierarchy level, it can remain in place. To enable use of the script again later, simply replace the file in the commit script directory.

When you include the optional statement at the [edit system scripts commit file filename] hierarchy level, no error message is generated during the commit operation if the file does not exist. However as a result, you might not be aware that a script is not executed as you expect if the file isn't available.

You can also deactivate and reactivate commit scripts by issuing the deactivate and activate configuration mode commands. When a commit script is deactivated, the script is marked as inactive in the configuration and does not execute during the commit operation. When a commit script is reactivated, the script is again executed during the commit operation.

Configuring Checksum Hashes for a Commit Script

- Create the script
- Place the script in the /var/db/scripts/commit directory on the device
- Run the script through one or more hash functions to calculate hash values: MD5, SHA-1, SHA-256

```
user@host> file checksum md5
/var/db/scripts/commit/script1.py
MD5 (/var/db/scripts/commit/script1.py) =
3af7884eb56e2d4489c2e49b26a39a97
```

- Configure the script

```
[edit system scripts commit]
user@host# set file script1.py checksum md5
3af7884eb56e2d4489c2e49b26a39a97
```

Configuring Checksum Hashes for a Commit Script

You can configure checksum hashes that can be used to verify the integrity of a commit script before the script runs on the switch, router, or security device.

To configure a checksum hash:

1. Create the script.
2. Place the script in the /var/db/scripts/commit directory on the device.
3. Run the script through one or more hash functions to calculate hash values. Junos OS supports MD5, SHA-1, and SHA-256 hash functions. The example shows only MD5

```
user@host> file checksum md5 /var/db/scripts/commit/script1.py
MD5 (/var/db/scripts/commit/script1.py) = 3af7884eb56e2d4489c2e49b26a39a97
```

4. Configure the script.

```
[edit system scripts commit]
user@host# set file script1.py checksum md5 3af7884eb56e2d4489c2e49b26a39a97
```

During the execution of the script, Junos OS recalculates the checksum value using the configured hash and verifies that the calculated value matches the configured value. If the values differ, the execution of the script fails. When you configure multiple checksum values with different hash algorithms, all the configured values must match the calculated values, otherwise, the script execution fails. The commit operation also fails.

Commit Script Actions

- Generate a warning message to the committing user:

```
jcs.emit_warning()
```

- Generate an error message and cause the commit operation to fail:

```
jcs.emit_error()
```

- Generate a system log message:

```
jcs.syslog()
```

- Generate a persistent change to the configuration:

```
jcs.emit_change(content, 'change', format)
```

- Generate a transient change to the configuration:

```
jcs.emit_change(content, 'transient-change', format)
```

Commit Script Actions

Junos OS Commit scripts written in Python use functions from the `jcs` library. The slide shows the main functions you will use and what they do.

Create a Commit Script (1 of 5)

- Write a commit script that ensures that each device meets basic configuration requirements:
 1. SSH is enabled
 2. The user account `jnpr` exists on the device
 3. Each device has an IP address configured for the `fxp0` management interface

Create a Commit Script: Part 1

The next few pages will walk you through creating a commit script that completes basic configuration checks. The script ensures that SSH is enabled, that an account called `jnpr` exists on the device, and the each device has an IP address configured on the `fxp0` management interface.

Create a Commit Script (2 of 5)

1. Write the Script – Part 1

```
from junos import Junos
from junos import Junos_Context
from junos import Junos_Configuration
import jcs

def main():
    # Get configuration root object
    root = Junos_Configuration

    # Check for 'ssh' configuration
    if not(root.xpath("./system/services/ssh")):
        jcs.emit_error("SSH must be enabled.")

    # Ensure that user account jnpr exists
    if not(root.xpath("./system/login/user[name='jnpr']")):
        jcs.emit_error("The jnpr user account must be created.")
```

Write the Script - Part 1

The first part of the script imports the necessary Python libraries. The libraries will be explained as the script is discussed.

The `Junos_Configuration` object represents the Junos post-inheritance candidate configuration and is assigned to the `root` variable. The `Junos_Configuration` is in XML format. The `root.xpath()` method queries the configuration for the `[system services ssh]` hierarchy. If it is not present, then the `jcs.emit_error()` function displays the error, as shown in the slide.

A similar process is followed for the user `jnpr`. The `root.xpath()` function queries the configuration for the existence of the name `jnpr` in the `[system login user]` hierarchy, then emits an error if it is not found.

Create a Commit Script (3 of 5)

1. Write the script – Part 2

```
# Verify that fxp0 has an IP address
fxp0_interface_ip =
root.xpath("./interfaces/interface[name='fxp0']/unit[name='0'
]/family/inet/address/name")
if not(fxp0_interface_ip):
    jcs.emit_error("fxp0 must have an IP address.")

if __name__ == '__main__':
    main()
```

Write the Script - Part 2

The script uses the `root.xpath()` function next to retrieve the value in the `[interfaces interface fxp0 unit 0 family inet address name]` hierarchy. This is a good example of how XPath can be used to navigate the Junos OS configuration hierarchy.

Next, the `if not` statement checks to see if there is a value returned and stored in the `fxp0_interface_ip` variable. If there is no value in the `fxp0_interface_ip` variable, the error shown on the slide is displayed for the user.

Create a Commit Script (4 of 5)

2. Copy the script to the proper location on the device running Junos OS

```
scp CommitScript.py lab@vMX-1:/var/db/scripts/commit
```

3. Set the language statement

```
[edit system scripts]
lab@vMX-1# set language python
```

4. Enable the script

```
[edit system scripts commit]
lab@vMX-1#set file CommitScript.py
```

Create a Commit Script: part 4

After the script is created, it needs to be copied to the device running Junos OS so that it can be executed whenever the configuration is committed. Reminder: commit scripts need to be stored in the /var/db/scripts/commit directory.

If you have not yet set the language to be Python, you need to issue the `set language python` statement at the [edit system scripts] hierarchy.

Next, you need to list the `CommitScript.py` file in the [edit system scripts commit] hierarchy, as shown in the slide.

Create a Commit Script (5 of 5)

5. Test the script using commit check

```
[edit]
lab@vMX-1# commit check
error: The jnpr user account must be created.
error: 1 error reported by commit scripts
error: commit script failure
```

Create a Commit Script: Part 5

Once the script is created and enabled in Junos OS, configuration commit the script or issue the `commit check` command. The `commit check` command should give you the error shown on the slide if you don't have a `jnpr` user account created. To fix the error, either modify the script or add the user account specified.

Agenda: Junos OS Commit and Op Scripts

- Junos Automation Overview
- Junos OS Commit Scripts
- Junos OS Op Scripts

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 35

Junos Op Scripts

The slide highlights the topic we discuss next.

Junos OS OP Scripts

- Junos OS Op scripts are executed in the Junos CLI by the user, or upon login, or as part of another script
- Op scripts can:
 - Create custom operational mode commands
 - Execute a series of operational mode commands
 - Customize the output of operational mode commands
 - Gather operational information and iteratively narrow down the cause of a network problem
 - Perform controlled configuration changes
 - Monitor status of a device by creating a general operation script that checks network warning parameters

Junos Op Scripts Overview

Junos OS operation (Op) scripts automate network and device management and troubleshooting. Op scripts can perform any function available through the remote procedure calls (RPCs) supported by either the Junos XML management protocol or the Junos Extensible Markup Language (XML) API. Op scripts can be executed manually in the CLI, or upon user login, or they can be called from another script. They are executed by the Junos OS management process (mgd).

Create an Op Script (1 of 3)

▪ Write the Script

```
from jnpr.junos import Device

dev = Device()
dev.open()

inv = dev.rpc.get_chassis_inventory()

print "model: %s" % inv.find('chassis/description').text
print "serial-number: %s" % inv.find('chassis/serial-
number').text

dev.close()
```

Write the Op Script

Op scripts are implemented similarly to commit scripts. The main difference is that they are triggered by the user rather than upon configuration commit. Op scripts can also change the configuration. This slide and the next three slides show the basics of OP script creation.

The script in the slide retrieves data from the `get_chassis_inventory()` RPC and then displays the model and serial number.

Create an Op Script (2 of 3)

2. Copy the script to the proper location on the device running Junos OS

```
scp OpScript1.py lab@vMX-1:/var/db/scripts/op/
```

3. Set the language statement

```
[edit system scripts]
lab@vMX-1# set language python
```

4. Enable the script

```
[edit system scripts op]
lab@vMX-1#set file OpScript1.py
```

5. Commit the configuration

```
[edit]
lab@vMX-1#Commit
```

Enable the Script in Junos OS

As with commit scripts, you need to store the op script in the correct location, configure the language python statement, enable the script, and then commit it, as shown in the slide.

Create an Op Script (3 of 3)

6. Execute the script

```
[edit]
lab@vMX-1# run op OpScript1.py
model: VMX
serial-number: VM58D97E4E26
```

Execute the OP Script

Finally, you execute the script to verify that it works.

Op Script Command Line Arguments

- Allows for user input into scripts
- Can be incorporated into the Junos OS help system
 - Help information can be included in the Python script or the Junos OS configuration
- To add information to Junos OS help system from within an Op script, create a Python arguments dictionary to define command line arguments

```
arguments = { 'name': 'description', 'name2': 'description2' }
```

Command Line Arguments

Junos OS op scripts can accept command-line arguments when you invoke the script. Command line arguments allow you to pass information to the script for processing. One possible challenge with using command line interfaces is informing users of the acceptable command line arguments. Junos OS provides two mechanisms of including command line argument information into the Junos OS context-sensitive help system.

The first method of including command line argument information in the Junos OS help system is to declare an arguments dictionary inside the Python script itself. The dictionary must be called *arguments*. The dictionary can contain multiple key value pairs, as shown in the slide. The names and descriptions within each key value pair will appear in the Junos help system.

The next several slides show the creation of an Op script with command line arguments.

Script-generated Command Line Arguments Example (1 of 4)

```

from jnpr.junos import Device
from lxml import etree
import argparse

# Define arguments dictionary
arguments = {'interface': 'Name of interface to display',
'protocol': 'Protocol to display (inet, inet6)'}

def main():
    # Create a parser object to hold command line arguments
    parser = argparse.ArgumentParser(description = 'Output interface
information')

    # put command line arguments into Python Dictionary
    for key in arguments:
        parser.add_argument('+' + key, required=True,
                           help=arguments[key])
    args = parser.parse_args()

    dev = Device()
    dev.open()

```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER Worldwide Education Services

www.juniper.net | 41

Script Generated Command Line Arguments: Part 1

The script in the slide defines two arguments in the arguments dictionary. First, the arguments dictionary defines an `interface` argument that accepts the name of an interface from the user. The second argument accepts a `protocol`. In this case, users have a choice between `inet` and `inet6`.

The first statement inside of `main()` is where the `argparse` class creates a new parser object called `parser`. The `description` variable is a description of what the script does. The `argparse` class is a Python class that must be imported before it can be used.

Next, the two arguments stored in the Python arguments dictionary are added to `parser` object using the `add_argument()` method. Note that each of the arguments is prepended with a minus sign; this is a requirement of the `argparse` class. The `add_argument()` method takes three parameters. The first is the name of the argument. For our example, the arguments are `-interface` and `-protocol`. The second parameter defines whether or not the argument is required; in our example both are required. The third parameter is the help information or description that is attached to the parameter. For our script, the help information is stored in the data referenced by the `arguments[key]`. If this syntax is confusing, research how to access data in a Python dictionary.

The `args = parser.parse_args()` method generates the help file. This is not the context-sensitive help seen within Junos. This generates the Python help that appears if you run the `OpScript2 --help` command from the Linux command line. If you want more information on the `argparse` class, see <https://docs.python.org/2/howto/argparse.html>.

The last two lines create the device connection and open it. Note that we do not need to specify the host, user, or password arguments for the device, because the script will run on the Junos device.

Script-generated Command Line Arguments Example (2 of 4)

```

# Store result of RPC into lxml etree object
res = dev.rpc.get_interface_information(interface_name =
args.interface, terse=True, normalize=True)

# Print interface and interface status
print args.interface + " status: " + \
res.findtext("logical-interface/oper-status")

# Find and print interface family and IP address
for elem in res.xpath("//address-family \
[normalize-space(address-family-name)=$protocol]", \
protocol=args.protocol):
    if (elem.find("interface-address/ifa-local") is not None):
        print "inet address " + \
        elem.find("interface-address/ifa-local").text

dev.close()

if __name__ == '__main__':
    main()

```

Script Generated Command Line Arguments: Part 2

The RPC call, at the top of the slide, retrieves interface information for interfaces specified in the command line argument, which is now stored in `args.interface`. Next, the `print` statement prints the interface name and its status. The `findtext()` method searches the `res` lxml tree for the first instance in the XML element path of “`logical-interface/oper-status`” and returns the value within the `oper-status` element, which will either be “`up`” or “`down`”.

In the next section, the Python `for` loop uses the `xpath()` method. The lxml `xpath()` method can take two arguments; if the `xpath()` method uses two arguments, the second argument stores a variable. The variable in the second argument is converted to an XPath variable for use in the first argument. XPath variables are preceded by a dollar sign (\$). The XPath variable in the first argument is `$protocol`. The value of `$protocol` comes from the second argument, `protocol`. However, `protocol` is assigned its value from `args.protocol`, which was passed into the script as a command line parameter by the user. The `$protocol` argument will either hold `inet` or `inet6`. Also, note the `normalize-space` statement. The `normalize-space` statement is necessary to take out extra white space that may have come with the `$Protocol` or exists within the XML tree. When you boil that script down, the `for` loop parses the interface, looks through the protocols and searches for one that matches what was passed into the script by the user.

If the `for` loop finds a matching protocol, the `if` statement then uses the lxml `find()` method to find the “`interface-address/ifa-local`” element. Previously, you saw the `findtext()` method, which finds the text contained in an element; the `find()` method finds the element itself. So if the “`interface-address/ifa-local`” element exists or “`is not None`”, then the `print` statement prints the text contained within `ifa-local` which, is the IP address.

If the XML hierarchy is difficult to follow, issue the `show interfaces interface-name | display xml` command and you will be able to see what the XPath statements

Script-generated Command Line Arguments

Example (3 of 4)

- Test the script and command line arguments

```
[edit]
lab@vMX-1# run op OpScript2.py
usage: OpScript2.py [-h] -interface INTERFACE
                    -protocol PROTOCOL
OpScript2.py: error: argument -interface is required
error: op script fails with exit code 0x200

[edit]
lab@vMX-1# run op OpScript2.py interface ge-0/0/0.0
protocol inet
ge-0/0/0.0 status: up
inet address 172.17.1.1/24
```

Test the script and command line arguments

If you try to run the script by itself, without any arguments, you'll receive the error message shown on the slide. Notice the second example, after the `interface` and `protocol` arguments are added, there is no error.

Look closer at the usage specified at the top of the script. It shows that the `interface` and `protocol` arguments should be preceded with a minus (-) sign. In the second example, minus signs are not used.

Why does this work? Recall that the Python script adds the minus sign to the argument in the Python script before it is added to the Python argparse object. This is done to maintain consistency within Junos OS. Typically we do not add a minus sign or double minus sign before arguments in Junos as is common in Linux or Unix environments.

Script-generated Command Line Arguments Example (4 of 4)

▪ Test context-sensitive help

```
lab@vMX-1> op OpScript2.py ?
```

Possible completions:

< [Enter] >	Execute this command
<name>	Argument name
detail	Display detailed output
interface	Name of interface to display
invoke-debugger	Invoke script in debugger mode
protocol	Protocol to display (inet, inet6)
	Pipe through a command

```
lab@vMX-1> op OpScript2.py
```

Test the Context-Sensitive Help

When you issue the `op OpScript2.py ?` command, you will see the list of possible completions. Notice that both the `interface` and `protocol` arguments are listed and, unlike the Python `usage` message on the previous page, there is no need for arguments to be preceded by a minus sign. Recall that this information comes from the `arguments` dictionary within the Python script and not from the Python `argparse` object.

The next few slides detail how to use Junos OS configuration instead of the Python `arguments` dictionary to provide Junos OS context-sensitive help.

Configuration Generated Command Line Arguments (1 of 5)

- Template to define command line arguments within the Junos OS

```
[edit system scripts op file filename]  
arguments {  
    argument-name {  
        description descriptive-text;  
    }  
}
```

Template to Define Command Line Arguments in Junos OS Configuration

The slide shows the basic format for defining context-sensitive help within Junos OS configuration.

Configuration Generated Command Line Arguments (2 of 5)

- Example: define arguments in Junos OS configuration

```
[edit]
lab@vMX-1# show system scripts op file OpScript2
arguments {
    interface {
        description "Name of interface - ge-0/0/0.0";
    }
    protocol {
        description "Protocol - inet or inet6";
    }
}
```

Example of Configured Context-sensitive Help

The slide shows an example of how to define the two arguments in Junos OS configuration that are needed for the OpScript2.py file.

Configuration Generated Command Line Arguments (3 of 5)

- Context-sensitive help based on Junos OS configuration

```
lab@vMX-1# run op OpScript2.py ?
```

Possible completions:

<[Enter]>	Execute this command
<name>	Argument name
detail	Display detailed output
interface	Name of interface - ge-0/0/0.0
invoke-debugger	Invoke script in debugger mode
protocol	Protocol - inet or inet6
	Pipe through a command
[edit system scripts op]	

```
lab@vMX-1# run op OpScript2.py
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 47

Context-sensitive Help Results

The slide shows the context-sensitive help in use. The output has the same result as the context-sensitive help defined within the Python script.

Configuration Generated Command Line Arguments (4 of 5)

- Adding script descriptions to context-sensitive help

```
[edit system scripts op]
lab@vMX-1# show
file OpScript1.py {
    description "Show device information";
}
file OpScript2.py {
    description "Show interfaces information";
    arguments {
        interface {
            description "Name of interface - ge-0/0/0.0";
        }
        protocol {
            description "Protocol - inet or inet6";
        }
    }
}
```

Adding Context-sensitive Script Descriptions

In addition to defining context-sensitive help, you can also add context-sensitive script descriptions using the format shown in the slide.

Configuration Generated Command Line Arguments (5 of 5)

▪ Context-sensitive script descriptions

```
[edit system scripts op]
```

```
lab@vMX-1# run op ?
```

Possible completions:

<script>	Name of script to run
OpScript1.py	Show device information
OpScript2.py	Show interfaces information
invoke-debugger	Invoke script in debugger mode
url	Execution of remote op script

```
[edit system scripts op]
```

```
lab@vMX-1# run op
```

Context-sensitive Script Description Results.

The slide shows the result of the configuration from the previous page.

Giving Additional Permissions to Op Scripts

- Default: scripts operate with permissions of user

```
user1@device> op sam  
error: permission denied: date
```

- Configure `sam.py` op script to execute `date` command with elevated permissions

```
[edit system scripts op file sam.py]  
admin@device# set allow-commands date  
admin@device# commit
```

- `user1` can now successfully execute the script

```
user1@device> op sam  
Fri May 19 11:11:33 EDT 2017
```

Giving Additional Permissions to Op Scripts.

By default, op scripts operate using the permissions of the user running the script. Occasionally you may have a script that you want a user to run, but that user does not have sufficient permissions.

The slide shows an example in which `user1` attempts to run an op script called `sample.py` but is denied because of missing permissions to run the `restart` command.

Next, note that the file `sample.py` is given permission with the `set allow-commands restart`.

Finally, in the slide `user1` now can run the `sample.py` script.

Note: The `allow-commands` statement is only supported for op scripts that are local to the device. Remote op scripts that are executed using the `op url` command do not support executing unauthorized commands even when you configure the `allow-commands` statement.

Executing an Op Script On a Remote Site (1 of 2)

1. Create script and save to the device
2. Create checksum

```
user@host> file checksum md5 /var/tmp/script1.py
MD5 (/var/tmp/script1.py) = 3af7884eb56e2d4489c2e49b26a39a97
```

3. Enable script on device

[edit]

```
user@host# set system scripts op file script1.py
user@host# commit
```

4. Configure the language python statement and allow-url-for-python statements

```
user@host# set system scripts language python
user@host# set system scripts op allow-url-for-python
```

Execute a Script On a Remote Site: Part 1

As an alternative to storing operation (op) scripts locally on the device, you can store op scripts at a remote site. To retrieve, you execute a remote op script by specifying the URL as an argument to the op command; do this when you execute the script on the command line. You can execute SLAX and XSLT op scripts from a remote site by default.

To execute Python op scripts from a remote site, configure the allow-url-for-python statement at the [edit system scripts op] hierarchy level. Because you cannot guarantee that scripts executed from remote sites are secure, we recommend that you only authorize trusted users to execute scripts with the op url command.

Note: Statements configured under the [edit system scripts op] hierarchy level are only enforced for op scripts that are local to the device. Thus, even if you configure memory allocation, script dampening, traceoptions, or other op script-specific statements within that hierarchy, Junos OS does not apply the configuration when you execute a remote script using the op url command.

Executing an Op Script On a Remote Site (2 of 2)

5. Place the script on the remote server
6. Provide the script URL and the optional hash values to administrators who will execute the script
7. Execute the script by running the op url command

```
user@host> op url https://www.juniper.net/scripts/script1.py
key md5 3af7884eb56e2d4489c2e49b26a39a97
```

Execute a Script On a Remote Site: Part 2

To prevent the execution of an op script on a remote site, configure the no-allow-url statement at the [edit system scripts op] hierarchy level.

```
user@host# set system scripts op no-allow-url
```

When you configure the no-allow-url statement, issuing the op url url operational mode command generates an error. This statement takes precedence when the allow-url-for-python statement is also present in the configuration.

Enable Traceoptions for Op Scripts

- Enable traceoptions

```
[edit system scripts op]
user@host# set traceoptions flag output
```

- Common traceoption message filters

```
[edit]
user@host# run show log op-script.log | last
user@host# run show log op-script.log | match error
user@host# run show log op-script.log | match filename
```

Enable Traceoptions

The simplest way to view the trace output of an op script is to configure the `output trace` flag and issue the `show log op-script.log | last` command. To do this, perform the command on the slide. The resulting trace messages are recorded in the `/var/log/op-script.log` file.

You cannot change the directory (`/var/log`) to which trace files are written. However, you can customize other trace file settings for local op scripts by including the following statements at the `[edit system scripts op traceoptions]` hierarchy level.

[For more information on Traceoptions for op scripts, see https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/automation-tracing-op-script-processing.html.

Common Traceoptions message filters

The slide shows some common filters you can use to filter the results generated by traceoptions.

Summary

- In this content, we:
 - Created Junos OS Commit Scripts
 - Created Junos OS Op Scripts

We Discussed:

- Creating Junos OS commit scripts; and
- Creating Junos OS op scripts.

Review Questions

1. What is the pre-inheritance candidate configuration?
2. In what directory should you store commit scripts?
3. Name the two ways to define the list of expected op script arguments to display when using context-sensitive help.

Review Questions

- 1.
- 2.
- 3.

Lab: Commit and Op Scripts

- Create Commit Scripts
- Create Op Scripts

Lab: Creating Commit and Op Scripts

This slide provides the objective for this lab.

Answers to Review Questions

1.

The pre-inheritance candidate configuration is the configuration before it is merged with any Junos group information.

2.

The directory you should store commit scripts in is the /var/db/scripts/commit directory.

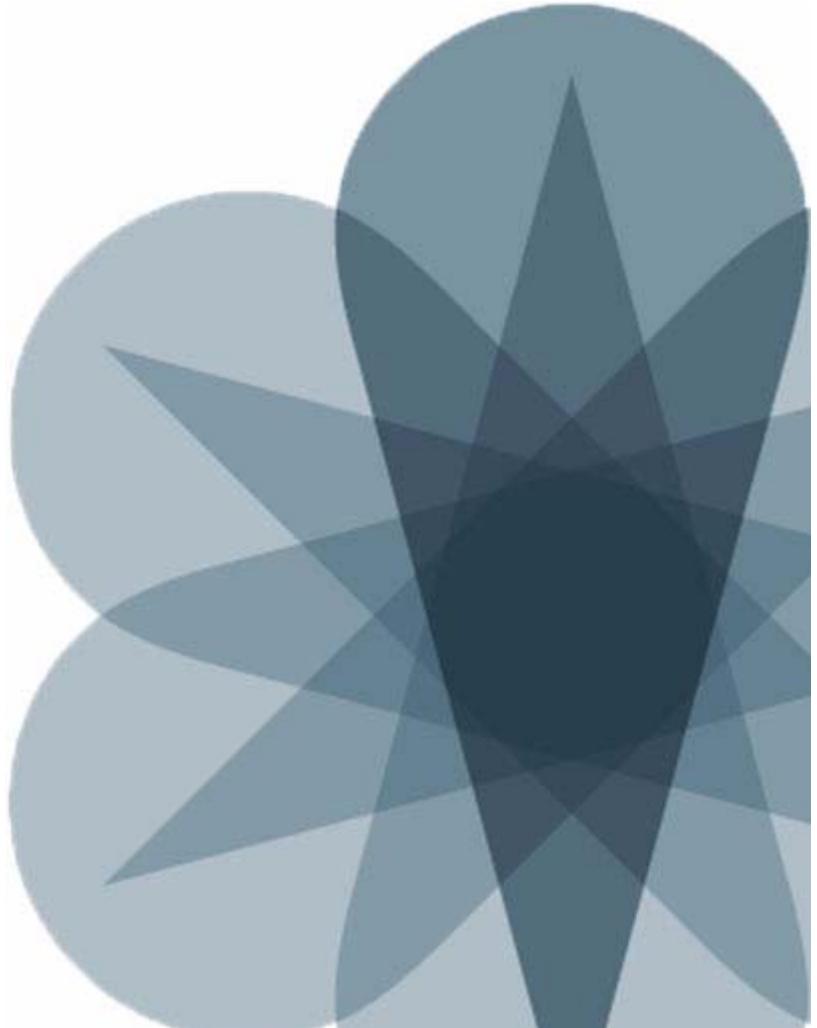
3.

The two places you can provide help information for op scripts is in the Python script and also in the Junos configuration.



Junos Platform Automation and DevOps

Chapter 10: Junos Automation Event and SNMP Scripts



Objectives

- After successfully completing this content, you will be able to:
 - Identify Junos OS events
 - Create Junos OS event policies
 - Create Junos OS event scripts
 - Create Junos OS SNMP scripts

We Will Discuss:

- Identifying Junos OS events;
- Creating Junos OS event policies;
- Creating Junos OS event scripts; and
- Creating Junos OS SNMP scripts.

Agenda: Junos OS Event and SNMP Scripts

- Junos OS Events
- Junos OS Event Policies
- Junos OS Event Scripts
- Junos OS SNMP Scripts

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

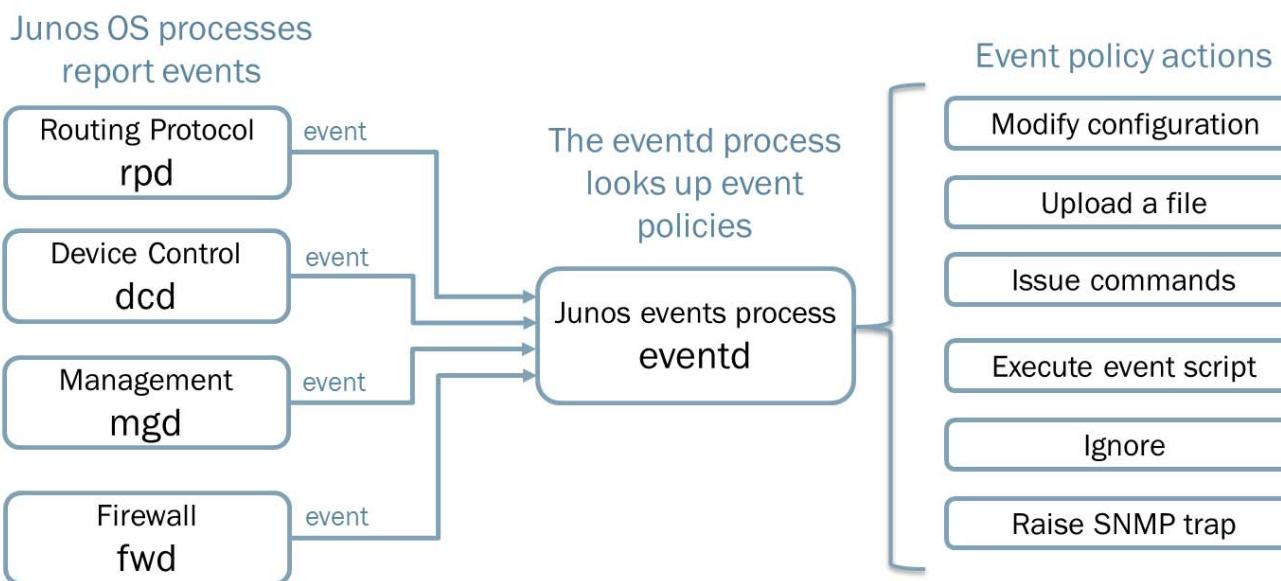
www.juniper.net | 3

Introduction to Event Scripts

The slide lists the topics we will discuss. We discuss the highlighted topic first.

Junos OS Event Model

- Junos OS processes raise events; the eventd process uses event policies to determine appropriate actions



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 4

The Junos OS Event Model

Events can originate as SNMP traps or system log messages. The event process receives event messages from other Junos OS processes, such as the routing protocol process (rpd) and the management process (mgd). Depending on the custom event policy you configure, eventd (pronounced “event-D”) searches for specific events, and in response to certain events may create a log file, invoke a Junos OS command, or invoke an event script. When an event script is invoked, event details are passed to the event script in the form of XML inputs.

In response to events, the eventd process can execute the following actions:

- Ignore the event — Do not generate a system log message for this event and do not process any further policy instructions for this event.
- Upload a file — Upload a file to a specified destination. You can specify a transfer delay, so that, on receipt of an event, the upload of the file begins after the configured transfer delay.
- Execute Junos OS operational mode commands — Execute commands on receipt of an event. The XML or text output of these commands is stored in a file, which is then uploaded to a specified URL.
- Execute Junos OS configuration mode commands — Execute commands to modify the configuration on receipt of an event.
- Execute Junos OS event scripts — Execute event scripts on receipt of an event.
- Raise an SNMP trap.

Don't Get Your Events Mixed Up

- Event Notifications or Events
 - System log messages and SNMP traps
- Event Policies
 - Defined actions to take when event notifications are received
- Event Scripts
 - Junos automation scripts that can be triggered by event policies



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 5

Event Notifications

Event notifications are most often called events. Event notifications are messages sent to the Junos OS by Junos OS processes, or SNMP traps that are recorded in log file.

Event Policies

Junos OS event policies are policies you define within Junos OS that describe what you want to have happen when specific event notifications are received.

Event Scripts

Event policies can trigger many different actions, one of which is to run a script. Scripts that are triggered by an event policy are called event scripts.

Response to an Event

Event policies define the way that Junos OS should respond to a given event. When you configure event policies, the Junos OS processes every event through the policies in sequential order, applying all matching policies.

In this chapter, you will learn how to identify events, how to trigger events, how to create event policies, and how to create event scripts.

Viewing a List of Events

▪ View list of event tags

```
lab@vMX-1> help syslog
```

Syslog tag	Help
AAA_RADIUS_SERVER_STATE_CHANGE	State of the radius server has changed
ACCT_ACCOUNTING_FERROR	Error occurred during file processing
ACCT_ACCOUNTING_FOPEN_ERROR	Open operation failed on file
ACCT_ACCOUNTING_SMALL_FILE_SIZE	Maximum file size is smaller than record size
ACCT_BAD_RECORD_FORMAT	Record format does not match accounting profile
profile	
... Trimmed ...	

▪ Get more information on event tags

```
lab@vMX-1> help syslog ACCT_BAD_RECORD_FORMAT
```

Name:	ACCT_BAD_RECORD_FORMAT
Message:	Invalid statistics record: <entry>
Help:	Record format does not match accounting profile
Description:	The number of columns in the indicated record does not match the number of columns in the accounting profile.
Type:	Error: An error occurred
Severity:	warning
Facility:	LOG_PFE

Viewing a List of Events

Before you can create a policy to handle an event, you need to identify the event. One method to identify an event is to view a list of all events by issuing the `help syslog` command. You can view more information about an event by including the event tag name in the `help` command.

For example, if you want more information about the `ACCT_BAD_RECORD_FORMAT` event, issue the `help syslog ACCT_BAD_RECORD_FORMAT` command.

Filter the List of Events

■ List of filters

```
lab@vMX-1> help syslog | ?
```

Possible completions:

append	Append output text to file
count	Count occurrences
display	Show additional kinds of information
except	Show only text that does not match a pattern
find	Search for first occurrence of pattern
hold	Hold text without exiting the --More-- prompt
last	Display end of output only
match	Show only text that matches a pattern

...trimmed...

```
lab@vMX-1> show log messages | match SNMP | count
```

Count: 152 lines

Filter the List of Events

The slide shows how to view the messages syslog file and how to apply multiple filters using the pipe (|) command.

The complete list of filters is provided here:

append	Append output text to file
count	Count occurrences
display	Show additional kinds of information
except	Show only text that does not match a pattern
find	Search for first occurrence of pattern
hold	Hold text without exiting the --More-- prompt
last	Display end of output only
match	Show only text that matches a pattern
no-more	Don't paginate output
refresh	Refresh a continuous display of the command
request	Make system-level requests
resolve	Resolve IP addresses
save	Save output text to file
tee	Write to standard output and file
trim	Trim specified number of columns from start of line

Identify Events

- Event tags are in all caps

```
lab@vMX-1> show log messages | last
```

```
May 12 06:09:56 vMX-1 chassisd[4564]: CHASSISD_FRU_OFFLINE_NOTICE:  
Taking FPC 0 offline: Error  
May 12 06:09:56 vMX-1 chassisd[4564]: CHASSISD_SNMP_TRAP0: ENTITY  
trap generated: entConfigChange  
May 12 15:31:35 vMX-1 mgd[4965]: UI_DBASE_LOGOUT_EVENT: User 'lab'  
exiting configuration mode  
May 12 08:44:23 vMX-1 agentd[4769]: Error: Unrecognised command 16  
May 12 14:24:53 vMX-1 xntpd: kernel time sync enabled 6001  
May 12 14:41:57 vMX-1 xntpd: kernel time sync enabled 2001  
May 12 15:31:35 vMX-1 mgd[4965]: UI_DBASE_LOGOUT_EVENT: User 'lab'  
exiting configuration mode  
May 12 18:06:48 vMX-1 xntpd: kernel time sync enabled 6001  
May 12 18:23:52 vMX-1 xntpd: kernel time sync enabled 2001
```

Identify Events

One of the best ways to view which events are being raised is by checking the system log files. This slide shows how to view the end of the messages log file.

Not all entries in the log files are events. The events that you can manipulate through an event policy appear in all caps. For example: `UI_DBASE_LOGOUT_EVENT`:

Syslog Messages with Structured Data

- Default syslog messages

```
lab@vMX-1> show log messages | match SNMP
May 12 03:41:53 vMX-1 mib2d[4581]: SNMP_TRAP_LINK_DOWN:
ifIndex 518, ifAdminStatus up(1), ifOperStatus down(2),
ifName ge-0/0/0
```

- Specify structured-data format to show attributes

[edit]

```
lab@vMX-1# set system syslog file messages structured-data
lab@vMX-1> show log messages | match SNMP
<28>1 2017-05-13T01:24:53.501Z vMX-1 mib2d 4577
SNMP_TRAP_LINK_DOWN [junos@2636.1.1.2.108 snmp-interface-
index="518" admin-status="down(2)" operational-
status="down(2)" interface-name="ge-0/0/0" ifIndex 518,
ifAdminStatus down(2), ifOperStatus down(2), ifName ge-0/0/0
```

Syslog Messages with Structured Data

Viewing log files in structured view can be helpful. The slide shows the `SNMP_TRAP_LINK_DOWN` event. This event occurs whenever an interfaces goes down.

The structured-data format provides more information without adding significant length, and makes it easier for automated applications to extract information from a message.

The structured-data format complies with Internet [draft-ietf-syslog-protocol-21.txt](#). The draft establishes a standard message format regardless of the source or transport protocol for logged messages.

To output messages to a file in structured-data format, include the structured-data statement at the `[edit system syslog file filename]` hierarchy level:

```
[edit system syslog file filename]
facility severity;
structured-data {
    brief;
}
```

The optional `brief` statement suppresses the English-language text that appears by default at the end of a message to describe the error or event. For more information about structured-data format, see: https://www.juniper.net/documentation/en_US/junos/topics/task/configuration/syslog-single-chassis-system-structured-data-format-configuring.html.

Generating Events

- On a nonproduction device, use the logger shell program to generate events

```
logger -e EVENT ID -p SYSLOG PRIORITY -d DAEMON -a  
ATTRIBUTE=VALUE MESSAGE
```

- Example using logger

```
user@R1> start shell  
% logger -e VRRPD_NEW_MASTER  
% exit  
exit
```

Generating Events

The testing process is an essential part of creating event policies and event scripts, but testing some events can be tricky if they are difficult to manually generate. Luckily, Junos ships with a test utility known as *logger* that can artificially generate any system event. With logger, it is possible to test event policies and event scripts successfully, no matter if the desired event is simple or arcane.

This Junos shell program, logger, is unsupported and should not be used on production devices. But logger is well-suited for use in lab environments where event policies and event scripts are being developed and verified.

The slide shows the proper syntax for using logger and an example.

It is also possible to trigger events within an event policy based on a time interval or the time of day. You will see how to do this in the event policy section of this chapter.

Agenda: Junos OS Event and SNMP Scripts

- Junos OS Events
- Junos OS Event Policies
- Junos OS Event Scripts
- Junos OS SNMP Scripts

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

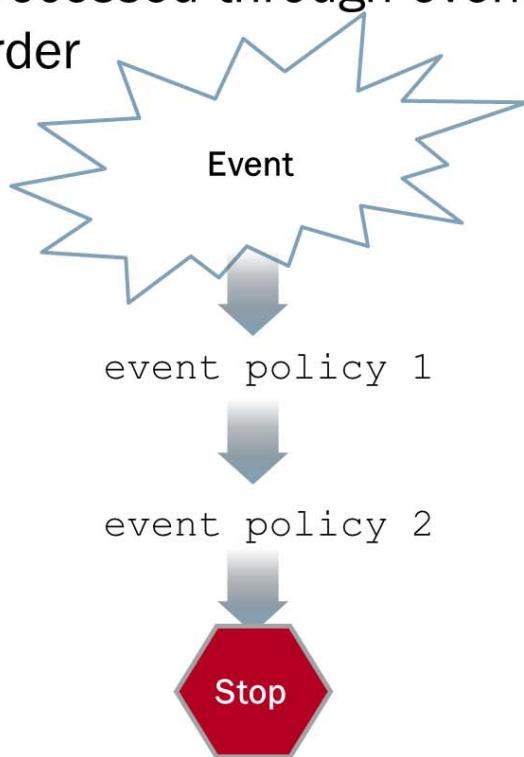
www.juniper.net | 11

Junos OS Event Policies

The slide highlights the topic we discuss next.

Response to an Event

- Events are processed through event policies in sequential order



Response to an Event

Event policies define the way Junos OS should respond to a given event. When you configure event policies, Junos OS processes every event through the policies in sequential order, applying all matching policies.

Basic Event Policy

```
[edit event-options]
lab@vMX-1# show
policy TEST {
    events TEST_EVENT;
    then {
        event-script check_syslog_severity.py;
    }
}
event-script {
    file check_syslog_severity.py;
}
```

Basic Event Policy

The slide shows a basic event policy. Event policies are stored in the [edit event-options] hierarchy. The slide shows a policy called TEST. The events statement shows that this policy will be triggered by an event called TEST_EVENT. Once a TEST_EVENT is received, an event script called *check_syslog_severity.py* is called. The event-script statement lists the file *check_syslog_severity.py* as an authorized file. The file will not be executed without being listed in the [edit event-options event-script] hierarchy. The then statement specifies an event-script as the action.

If you execute the following command, you can view the other options:

```
[edit event-options]
lab@vMX-1# set policy TEST then ?
Possible completions:
+ apply-groups           Groups from which to inherit configuration data
+ apply-groups-except   Don't inherit configuration data from these groups
> change-configuration Change configuration
> event-script          Invoke event scripts
> execute-commands      Issue one or more CLI commands
    ignore               Do not log event or perform any other action
> priority-override    Change syslog priority value
    raise-trap            Raise SNMP trap
> upload                Upload file to specified destination
[edit event-options]
```

Identifying Available Events in an Event Policy

- The set event-options policy policy-name events ? Command like the help syslog | ? Command shows available events you can use in a policy

[edit]

```
lab@vMX-1# set event-options policy TEST events ?
```

Possible completions:

```
<event>  
TEST_EVENT  
[  
          Open a set of values  
aaa_radius_server_state_change  
acct_accounting_ferror  
acct_accounting_fopen_error  
...Trimmed...
```

Identifying Available Events in an Event Policy

You can issue a command similar to the one shown on the screen to see a list of possible events.

Nonstandard System Log Messages

- This table contains nonstandard syslog messages that can be used in an event policy

Event IDs	Origin
SYSTEM	Messages from Junos OS daemons and utilities
KERNEL	Messages from the Junos OS kernel
PIC	Messages from physical interface cards (PICs)
PFE	Messages from the Packet Forwarding Engine
LCC	On a TX Matrix router, messages from a line-card chassis (LCC)
SCC	On a TX Matrix router, messages from a switch-card chassis (SCC)

Nonstandard System Log Messages

There are some nonstandard system log messages that can also be used as events to trigger event policies. The slide shows a list of the event IDs and corresponding descriptions.

Nonstandard System Log Message Example

- Nonstandard syslog messages use an event-id and attribute-match statement

```
[edit event-options]
policy kernel-policy {
    events KERNEL;
    attributes-match {
        KERNEL.message matches "exited on signal 11";
    }
    then {
        raise-trap;
    }
}
```

Nonstandard System Log Message Example

To base your event policy on a nonstandard event type, include the events event-id statement and the attributes-match statement with the event-id.message matches "message" attribute at the [edit event-options policy policy-name] hierarchy level.

Variables within Event Policies

Symbol	Meaning
\$\$	The event that is triggering a policy. Events listed at the [edit event-options policy <i>policy-name</i> events] hierarchy
\$event	The most recent event that matches the event name
\$*	The most recent event that matches any of the correlating events

```
[edit event-options]
policy p1 {
    events [ e1 e2 e3 ];
    within 60 events [ e4 e5 e6 ];
    then {
        execute-commands {
            commands {
                "show interfaces {$$.interface-name}";
                "show interfaces {$e4.interface-name}";
                "show interfaces {$*.interface-name}";
            }
        }
    }
}
... Trimmed...
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 17

Variables Within Event Policies

To help reference events within an event policy, Junos OS allows variables. The slide shows a summary of how variables are used in this context.

In the `show interfaces {$$.interface-name}` command, the value of the `interface-name` attribute of event `e1`, `e2`, or `e3` is substituted for the `{$$.interface-name}` variable.

In the `show interfaces {$e4.interface-name}` command, the value of the `interface-name` attribute of the most recent `e4` event is substituted for the `{$e4.interface-name}` variable.

In the `show interfaces {$*.interface-name}` command, the value of the `interface-name` attribute of the most recent `e4`, `e5`, or `e6` event is substituted for the `{$*.interface-name}` variable. If one of `e4`, `e5`, or `e6` occurs within 60 seconds of `e1`, `e2`, or `e3`, the value of the `interface-name` attribute for that correlating event (`e4`, `e5`, or `e6`) is substituted for the `{$*.interface-name}` variable. If the correlating event does not have an `interface-name` attribute, the software does not execute the `show interfaces {$*.interface-name}` command.

If both `e4` and `e5` occur within 60 seconds of `e1`, then the value of the `interface-name` attribute for `e4` is substituted for the `{$*.interface-name}` variable. This is because the event process (eventd) searches for correlating events in sequential order as configured in the `within` statement. In this case, the order is `e4 > e5 > e6`.

Correlated Event Policy using `within`

- Correlated events are events where more than one condition must be met to trigger the event policy

[edit event-options]

```
Policy policy1 {
    events [ event3 event4 event5 ];
    within 60 events [ event1 event2 ];
    then {
        ...
    }
}
```

Policy1 executes if events 3, 4, or 5 occur within 60 seconds of policy 1 or 2

Correlated Event Policy Using `within`

You can configure a policy that correlates two or more events. If the correlated events occur as specified, they cause particular actions to be taken. For example, you might want to issue certain operational mode commands when a `UI_CONFIGURATION_ERROR` event is generated within five minutes (300 seconds) after a `UI_COMMIT_PROGRESS` event. As another example, you might want to upload a particular file if a `DCD_INTERFACE_DOWN` event is generated two times within a 60-second interval.

You can configure a policy that is executed only if a specified event occurs within a specified time interval following another event. You do this by including the `within seconds events` statement.

The policy is executed only if one or more of the events in the `first events` statement occur within a configured number of seconds after one or more of the events in the `within seconds events` statement. The number of seconds can range from 60 through 604,800 seconds (one week).

For example, the policy in the slide is executed if `event3`, `event4`, or `event5` occurs within 60 seconds after `event1` or `event2`.

You can also use a `not events` statement, which causes the policy to be executed only if the events do not occur within the configured time interval.

Correlated Event Policy using `within`, `attribute match` and Variables

```
policy multiple-commits {
    events ui_commit;
    attributes-match {
        { $$ .user-name } equals { $ui_commit.user-name };
    }
    within 300 {
        trigger after 3;
        events ui_commit;
    }
    then ...
}
```

Correlated Event Policy using `within` and Variables

In the following example, the policy will execute the actions under the `then` statement if four or more commits are performed within a five minute period, and if the `username` of one or more of the correlated events is the same as the `username` of the trigger event.

The `attributes-match` statement correlates two events as follows:

- If `event1.attribute-name equals event2.attribute-name`—Execute the policy only if the specified attribute of `event1` equals the specified attribute of `event2`.
- If `event.attribute-name matches regular-expression`—Execute the policy only if the specified attribute of `event` matches a regular expression.
- If `event1.attribute-name starts-with event2.attribute-name`—Execute the policy only if the specified attribute of `event1` starts with the specified attribute of `event2`.

If the `attributes-match` statement includes the `equals` or `starts-with` options, or if it includes a `matches` option that includes a clause for an event that is not specified at the `[edit event-options policy policy-name events]` hierarchy level, you must include one or more `within` statements in the same policy configuration.

You can use event policy variables within the `attributes-match` statement to differentiate between a `trigger` event attribute and a correlated event attribute. The double dollar sign (`$$`) notation represents the event that is triggering a policy, and `{ $$.attribute-name }` resolves to the value of the attribute of the triggering event. Triggering events are those that you configure at the `[edit event-options policy policy-name events]` hierarchy level. For correlating events, the single dollar sign with the event name (`$event`) notation represents the most recent event that matches the event name, and `{ $event.attribute-name }` resolves to the value

Generating Internal Events

- Internal events using a time-interval

```
[edit event-options]
generate-event {
    my-event time-interval 3600;
}
```

- Internal events using a time-of-day

```
[edit event-options]
generate-event {
    my-event time-of-day 08:45:00;
}
```

Generating Internal Events

Internal events are events that you use to trigger a policy to be executed. They are not generated by Junos OS processes, and they do not have any associated system log messages. You can generate an internal event based on a time interval or the time of day.

In the `time-interval` statement, configure a frequency, in seconds, with which to repeatedly generate an event. The time interval can range from 60 seconds to 2,592,000 seconds (30 days).

In the `time-of-day` statement, configure a time of day for the event to occur. Use the format `hh:mm:ss`

In the slide, the first example generates the `my-event` event every 3,600 seconds, or every hour. The second example generates the `my-event` every day at 8:45 am.

Event Policies Triggered by Event Count

- You can trigger policies based on a count of occurrences

```
[edit event-options]
policy login {
    events [ RADIUS_LOGIN_FAIL TELNET_LOGIN_FAIL
SSH_LOGIN_FAIL ];
    within 120 {
        trigger after 4;
    }
    then {
        event-script login-fail.py {
            destination some-dest;
        }
    }
}
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER NETWORKS Worldwide Education Services

www.juniper.net | 21

Events Triggered by Event Count

The software counts the number of times the triggering event occurs. A triggering event can be any event configured at the [edit event-options policy policy-name events] hierarchy level. You can configure the following options:

- after event-count — The policy is executed when the number of matching events received equals event-count plus one.
- on event-count — The policy is executed when the number of matching events received equals event-count.
- until event-count — The policy is executed each time a matching event is received and stops being executed when the number of matching events received equals the event count.

The slide shows an event policy called `login`. The `login` policy is executed if five login failure events are generated within 120 seconds: (RADIUS_LOGIN_FAIL, TELNET_LOGIN_FAIL, or SSH_LOGIN_FAIL). Action is taken by executing the `login-fail.py` event script, which disables the user account.

Using Regular Expressions with Event Policies

Operator	Matches
.	One instance of any character except the space
*	Zero or more instances of the immediately preceding term
+	One or more instances of the immediately preceding term
?	Zero or one instance of the immediately preceding term
	One of the terms that appear on either side of the pipe operator
!	Any string except the one specified by the expression when the exclamation point appears at the start of the expression; specific to Junos OS
^	The start of a line, when the caret appears outside square brackets; one instance of any character that does not follow it within square brackets when the caret is the first character inside square brackets
\$	The end of a line
[]	One instance of one of the enclosed alphanumeric characters; to indicate a range of characters, use a hyphen (-) to separate the beginning and ending characters of the range (for example, [a-z0-9] matches any letter or number)
()	One instance of the evaluated value of the enclosed term; parentheses are used to indicate the order of evaluation in the regular expression

Using Regular Expressions with Event Policies

You can use regular expression matching to specify precisely which events cause a policy to be executed.

To specify the text string that must appear in an event attribute for the policy to be executed, include the matches statement at the [edit event-options policy policy-name attributes-match] hierarchy level, and specify the regular expression that the event attribute must match.

When you specify the regular expression, use the notation defined in POSIX Standard 1003.2 for extended (modern) UNIX regular expressions. Explaining regular expression syntax is beyond the scope of this course; the table in the slide specifies which characters are matched by some of the regular expression operators that can be used in the matches statement. In the descriptions, the term refers to either a single alphanumeric character or a set of characters enclosed in square brackets, parentheses, or braces.

Using Regular Expressions with Event Policies

- **Test your knowledge** – What does the following event policy do?

```
[edit event-options]
policy pol6 {
    events snmp_trap_link_down;
    within 120 events snmp_trap_link_up;
    attributes-match {
        snmp_trap_link_up.interface-name equals
        snmp_trap_link_down.interface-name;
        snmp_trap_link_down.interface-name matches "^t";
    }
    then {
        execute-commands {
            commands {
                "show interfaces {$$.interface-name}";
                "show configuration interfaces {$$.interface-name}";
            }
            output-filename config.txt;
            destination bsd2;
            output-format text; } } }
```

Using Regular Expressions with Event Policies

The following policy is executed only if the `interface-name` attribute in both traps (`SNMP_TRAP_LINK_DOWN` and `SNMP_TRAP_LINK_UP`) match each other and the `interface-name` attribute in the `SNMP_TRAP_LINK_DOWN` trap starts with letter `t`.

This means the policy is executed only for T1 (t1-) and T3 (t3-) interfaces. The policy is not executed when the `eventd` process receives traps from other interfaces.

Note that in system log files, the message tags appear in all uppercase letters. In the command-line interface (CLI), the message tags appear in all lowercase letters.

Outputting commands to a file is discussed next.

Executing Operational Commands in Event Policies

- Operational Mode commands can be output to a file

```
[edit event-options]
policy pol6 {
    events snmp_trap_link_down;
    then {
        execute-commands {
            commands {
                "show interfaces {$$.interface-name}";
                "show configuration interfaces {$$.interface-
name}";
            }
            destination bsd2;
            output-filename config.txt;
            output-format text;
        } } }
```

Executing Operational Mode Commands in Event Policies

Operational mode commands request that the device running Junos OS perform an operation or provide diagnostic output. These commands allow you to view statistics and information about the current operating status of a device. They also allow you to take corrective action, such as restarting software processes, taking a PIC offline and back online, switching to redundant interfaces, or adjusting label switching protocol (LSP) bandwidth.

You can configure an event policy that executes operational mode commands and uploads the output of those commands to a specified location for analysis.

In the `events` statement, you can list multiple events. If one or more of the listed events occurs, the `eventd` process executes the operational mode commands configured for the `commands` statement. Enclose each command in quotation marks ("").

The `eventd` process issues the commands in the order in which they appear in the configuration. For example, in the slide, the execution of `pol6` causes the `show interfaces {$$.interface-name}` command to be issued first, followed by the `show configuration interfaces {$$.interface-name}` command:

When the `eventd` process executes the commands, it uploads the file with the command output to the location specified in the `destination` statement. In the `destination` statement, include a destination name that is configured at the `[edit event-options destinations]` hierarchy, as shown in the slide.

In the `output-filename` statement, define a descriptive string that will be included in the filename.

Filename Format

- When event policies transfer data to a remote server, filenames are in the following format:

Hostname _ Date _ Filename _ Time

Example information:

Hostname	Date	Filename	Value
R1	2017-05-12	vrrp-mastership-change	10:04:00 PDT (17:04:00 UTC)

Resulting filename:

R1_20170512_170400_vrrp-mastership-change

If more than one file was transferred in the same second:

R1_20170512_170400_vrrp-mastership-change
R1_20170512_170400_vrrp-mastership-change **_001**
R1_20170512_170400_vrrp-mastership-change **_002**

Filename Format

When an event policy uploads a file to a remote server, each uploaded file includes the `hostname`, `filename` and `timestamp` in the filename to ensure that the each filename is unique. If a policy is triggered multiple times in a 1-second period, an index number is appended to the filename to ensure that the filenames are still unique. The index number range is 001 through 999.

Starting in Junos OS Release 14.1R3, the filename places the output-filename string after the timestamp.

```
hostname _YYYYMMDD _HHMMSS _output-filename _index-number
```

For example, on a device named r1 running Junos OS Release 14.1R3 or a later release, if you configure the output-filename statement as `ifl-events`, and this event policy is triggered three times within one second, the files are named:

```
r1_20060623_132333_ifl-events
r1_20060623_132333_ifl-events_001
r1_20060623_132333_ifl-events_002
```

By default, the command output format is Junos Extensible Markup Language (XML). Configure the output-format text statement to format the command output as ASCII text.

Changing the Configuration using Event Policies

```
[edit]
event-options {
    policy update-on-snmp-trap-link-down {
        events snmp_trap_link_down;
        attributes-match {
            snmp_trap_link_down.interface-name matches ge-0/3/1.0;
        }
        then {
            change-configuration {
                retry count 5 interval 4;
                commands {
                    "delete routing-options static route 10.1.10.0/24 next-hop";
                    "set routing-options static route 10.1.10.0/24 next-hop
10.1.3.1";
                }
                user-name bsmith;
                commit-options {
                    log "updating configuration from event policy update-on-
snmp-trap-link-down";
                }}}} }
```

If ge-0/3/1.0 interface goes down

change the static route; retry commit an additional five times if necessary

use user account bsmith and add a comment to the log

Changing the Configuration using Event Policies

You can configure an event policy action to modify the configuration when the policy is triggered by a single event or correlated events. Suppose you have a static route to the 10.1.10.0/24 network with a next-hop IP address of 10.1.2.1 through the exit interface ge-0/3/1. Then, at some point, this interface goes down, triggering an SNMP_TRAP_LINK_DOWN event.

This example creates an event policy named `update-on-snmp-trap-link-down`. The event policy is configured so that the `eventd` process listens for an `SNMP_TRAP_LINK_DOWN` event associated with the interface `ge-0/3/1.0`. If the interface goes down, the event policy executes a `change-configuration` action. The event policy configuration commands remove the static route through the `ge-0/3/1` exit interface and create a new static route to the same target network with a next-hop IP address of `10.1.3.1`, through the exit interface `ge-0/2/1`. The commands are executed in the order in which they appear in the event policy.

The event policy `change configuration commit` operation is executed under the `username bsmith`, with a `commit comment` that specifies that the change was made through the associated event policy. The `retry count` is set to five, and the `retry interval` is set to four seconds. If the initial attempt to issue the configuration change fails, the system attempts the configuration change five additional times and waits four seconds between each attempt.

Although not presented here, you might have a second, similar event policy that executes a `change configuration` action to update the static route when the interface comes back up. In that case, the policy would trigger on the `SNMP_TRAP_LINK_UP` event for the same interface.

Event Policies that Pass Arguments to Event Scripts

- In addition to calling a script, event policies can also pass values into a script

```
[edit event-options]
lab@vMX-1# show
policy TEST {
    events TEST_EVENT;
    then {
        event-script check_syslog_severity.py {
            arguments {
                TestValue "value1";
            }
        }
    }
} ...Trimmed...
```

Event Policies that Pass Arguments to Event Scripts

When an event policy invokes an event script, the policy can pass arguments to the script. You configure the arguments that an event policy passes to an event script within the `then` clause of the policy under the `[event-script filename arguments]` hierarchy. You can configure any number of arguments for each invoked event script.

The slide shows the `TEST` policy we saw several slides earlier, now with arguments added. The `TEST` policy has an argument called `TestValue` with a value of `value1` that will be passed to the `check_syslog_severity.py` script.

We will explain more about event scripts shortly.

Ignoring Events

- You can choose to ignore specific events
- You can choose to ignore correlated events

[edit event-options]

```
policy 1 {  
    events [ event1 event2 event3 ];  
    within 600 events [ event4 event5 ];  
    within 800 not events event6;  
    then {  
        ignore;  
    }  
}
```

Ignoring Events

You can modify an event policy to cause particular events to be ignored, or to cause all events to be ignored during a particular time interval (to allow for maintenance, for example). To configure such a policy, include the following statements at the [edit event-options] hierarchy level:

In the following policy, if any of event1, event2, or event3 has occurred, and either event4 or event5 has occurred within the last 600 seconds, and event6 has not occurred within the last 800 seconds, then the event that triggered the policy (event1, event2, or event3) is ignored, meaning system log messages are not created. In the events statement, you can list multiple events.

Note: If you include the ignore statement in a policy configuration, you cannot configure any other actions in the policy.

Event Policy Dampening

- You can reduce the number of times a script is executed by using the ignore statement

```
[edit event-options]
policy dampen-policy {
    events event1;
    within 60 events event1;
    then {
        ignore;
    }
}
policy policy1 {
    events event1;
    then {
        ... actions ...
    }
}
```

Event Policy Dampening

Sometimes events are generated repeatedly within a short period of time. In this case, it is redundant to execute a policy multiple times, once for each instance of the event. Event dampening allows you to slow down the execution of policies by ignoring instances of an event that occur within a specified time after another instance of the same event.

In the example on the slide, an action is taken only if the eventd process has not received another instance of the event within the past 60 seconds. If an instance of the event has been received any time within the last 60 seconds, the policy is not executed, and a system log message for the event is not created again.

Use Event Policy to Raise SNMP Traps

- You can use event policies to simultaneously raise traps and execute scripts

```
[edit event-options]
policy raise-trap-on-ospf-nbrdown {
    events rpd_ospf_nbrdown;
    then {
        event-script ospf.py {
            arguments {
                interface "{$$rpd_ospf_nbrdown.interface-
name}";
            }
            output-filename ospf-out;
            destination mgmt-archives;
        }
        raise-trap;
    }
}
```

Use Event Policy to Raise SNMP Traps

SNMP traps enable an agent to notify a network management system (NMS) of significant events by way of an unsolicited SNMP message. You can configure an event policy action that raises traps for events based on system log messages. If one or more of the listed events occur, the system log message for the event is converted into a trap. This enables notification of an SNMP trap-based application when an important system log message occurs. You can convert any system log message (for which there are no corresponding traps) into a trap. This is helpful if you use NMS traps rather than system log messages to monitor your network.

The example on the slide configures the event policy `raise-trap-on-ospf-nbrdown` to trigger the `RPD_OSPF_NBRDOWN` event, which indicates a terminated OSPF adjacency with a neighboring router. The event policy action raises a trap in response to the event. The device sends a notification to the SNMP manager, if one is configured under the `[edit snmp]` hierarchy level.

Additionally, the event policy executes the event script `ospf.py` in response to this event and provides the affected interface as an argument to the script. The `$$rpd_ospf_nbrdown.interface-name` argument resolves to the interface name associated with the triggering event.

The event script output is recorded in the file `ospf-out`, and the output file is uploaded to the destination `mgmt-archives`, which is configured at the `[edit event-options destinations]` hierarchy level. To invoke an event script in an event policy, the event script must be present in the `/var/db/scripts/event` directory on the hard disk, and it must be enabled in the configuration.

The Juniper Networks enterprise-specific System Log MIB, whose object identifier is `{jnxMibs 35}`, provides support for this feature.

Troubleshooting Event Policies and Scripts

- Tracing event policy processing
 - Include the traceoptions statement at the [edit event-options] hierarchy level
- Tracing Event script processing
 - Include the traceoptions flag output statement at the [edit event-options event-script] hierarchy level
- Script Permissions
 - Event scripts run under the permissions of a specified user account

```
[edit event-options event-script file filename]
```

```
user@host# set python-script-user username
```

Troubleshooting Event Policies

Event policy tracing operations track all event policy operations and record them in a log file. The logged error descriptions provide detailed information to help you solve problems faster.

By default, no events are traced. If you include the traceoptions statement at the [edit event-options] hierarchy level, the default tracing behavior is the following:

- Important events are logged in a file called eventd located in the /var/log directory.
- When the file eventd reaches 128 kilobytes (KB), it is renamed and compressed to eventd.0.gz, then eventd.1.gz, and so on, until there are three trace files. Then the oldest trace file (eventd.2.gz) is overwritten.
- Log files can be accessed only by the user who configures the tracing operation.

You cannot change the directory (/var/log) to which trace files are written. However, you can customize the other trace file settings. You can specify a different name by including the file statement at the [edit event-options traceoptions] hierarchy level. To learn more about configuring traceoptions for event policies, see https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/automation-tracing-event-policy-processing.html.

Agenda: Junos OS Event and SNMP Scripts

- Junos OS Events
- Junos OS Event Policies
- Junos OS Event Scripts
- Junos OS SNMP Scripts

Junos OS Event Scripts

The slide highlights the topic we discuss next.

Event Scripts

- Event Scripts are triggered by Event Policies
- Event scripts are used to:
 - Automatically diagnose and fix problems in the network
 - Monitor the overall status of a device
 - Run automatically as part of an event policy that detects periodic error conditions
 - Change the configuration in response to a problem

Event Scripts

Junos OS event scripts are triggered automatically by defined event policies in response to a system event and can instruct the Junos OS to take immediate action. Event scripts automate network and device management and troubleshooting. Event scripts can perform functions available through the remote procedure calls (RPCs) supported by either Junos XML management protocol, or the Junos Extensible Markup Language (XML) API. Event scripts are executed by the event process (`eventd`).

Event scripts enable you to:

- Automatically diagnose and fix problems in the network
- Monitor the overall status of a device
- Run automatically as part of an event policy that detects periodic error conditions
- Change the configuration in response to a problem

You can use event scripts to generate changes to the device configuration. Because the changes are loaded before the standard validation checks are performed, they are validated for correct syntax, just like statements already present in the configuration before the script is applied. If the syntax is correct, the configuration is activated and becomes the active, operational device configuration.

Passing Event Details to Event Scripts

- Event policies can pass two types of event details
 - Triggered event details that contain the details of the event that triggered the policy
 - Received event details that contain the details of events that happened before the triggering event
- Import event details in Python using:

```
import Junos_Trigger_Event
import Junos_Received_Events
```

- Access event details in Python using XPath:

```
Junos_Trigger_Event.xpath('//trigger-
event/process/name')[0].text
```

Passing Event Details to Event Scripts

Event policy actions can include executing one or more event scripts. When an event policy executes an event script, the event process forwards the event details to the script. These event details can be captured, evaluated, and sent to log files as required.

Two types of event details can be sent to event scripts: trigger events and received events. Trigger events record the details of the event that triggered the policy. Received events record the details of events that happened before the triggering event. Trigger event details are always forwarded to event scripts. Received event details are only present when an event policy is triggered for correlated events.

Python event scripts must import the `Junos_Trigger_Event` and `Junos_Received_Events` objects to access details about the trigger event and received events. `Junos_Trigger_Event` and `Junos_Received_Events` are `lxml.etree`.

Python event scripts can extract the necessary event details from the objects using `lxml` methods, such as: `xpath()` and `find()`, `findall()`, and `findtext()`.

For example:

```
Junos_Trigger_Event.xpath('//trigger-event/process/name')[0].text
```

Passing Remote Execution Information to Event Scripts

- Event scripts can execute commands on remote devices
- It is safer to store authentication information in the Junos OS and pass it to event script
- Example

```
[edit event-options event-script file filename]
remote-execution {
    remote-hostname {
        username username;
        passphrase passphrase;
    }
}
```

Passing Remote Execution Information to Event Scripts

Python event scripts must import `Junos_Remote_Execution_Details` to access the remote execution details configured at the `[edit event-options event-script file filename remote-execution]` hierarchy level. `Junos_Remote_Execution_Details` is a generator function that produces a sequence of remote devices, which makes it easy to iterate over multiple configured hosts. You can reference the `hostname`, `username`, and `passphrase` for a configured remote host by using the `host`, `user`, and `passwd` properties.

You configure remote execution details at the `[edit event-options event-script file filename remote-execution]` hierarchy level.

For each remote device where an RPC is executed, configure the device `hostname` and the corresponding `username` and `passphrase`.

```
[edit event-options event-script file filename]
remote-execution {
    remote-hostname {
        username username;
        passphrase passphrase;
    }
}
```

Remote Execution Example

```
from junos import Junos_Remote_Execution_Details
from jnpr.junos import Device

def main():
    for remote in Junos_Remote_Execution_Details():
        hostname = remote.host
        username = remote.user
        passphrase = remote.passwd

        jdev = Device(host=hostname, user=username,
passwd=passphrase)
        jdev.open()

        inv = jdev.rpc.get_interface_information()
        #process RPC information...
        jdev.close()

if __name__ == "__main__":
    main()
```

Remote Execution Example

The script in the slide shows how the `Junos_Remote_Execution_Details` object is used to provide authentication details for a Python script.

Enabling and Executing Event Scripts

- Enable Python

```
[edit system scripts]
user@host# set language python
```

- Enable the script

```
[edit event-options event-script]
user@host# set file filename
```

- Give the script access permissions

```
[edit event-options event-script file filename]
user@host# set python-script-user username
```

This is unique
to event and
SNMP scripts

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 37

Enabling and Executing Event Scripts

By default, you cannot execute unsigned Python scripts on devices running Junos OS. To enable the execution of unsigned Python automation scripts, you must configure the `language python` statement at the `[edit system scripts]` hierarchy level, as shown on the slide.

You must enable an event script before it can be executed. To enable an event script, include the `file filename` statement at the `[edit event-options events-script]` hierarchy level, specifying the name of the file containing the event script. Only users who belong to the Junos super-user login class can enable event scripts.

By default, Junos OS executes Python event scripts with access privileges of a generic, unprivileged user and group `nobody`. Starting in Junos OS Release 16.1R3, you can specify the user under whose access privileges the Python script will execute. To execute a Python event script under the access privileges of a specific user, configure the `python-script-user username` statement at the `[edit event-options event-script file filename]` hierarchy level.

Tracing Event Script Processing

- Event script tracing operations track all event script operations
- Event script events are logged in `escript.log` file
- To enable trace options

```
[edit event-options event-script]
User@host# set traceoptions flag output
```

- View log file

```
[edit]
user@host# run show log escript.log | last
```

Tracing Event Script Processing

Event script tracing operations track all event script operations and record them in a log file. The logged error descriptions provide detailed information to help you solve problems faster.

The default operation of event script tracing is to log important events in a file called `escript.log` located in the `/var/log` directory. When the file `escript.log` reaches 128 kilobytes (KB), it is renamed with a number 0 through 9 (in ascending order) appended to the end of the file, and then compressed. The resulting files are `escript.log.0.gz`, then `escript.log.1.gz`, until there are ten trace files. Then the oldest trace file (`escript.log.9.gz`) is overwritten.

Enable trace options by including the `traceoptions flag output` statement at the `[edit event-options event-script]` hierarchy level:

```
[edit event-options event-script]
user@host# set traceoptions flag output
```

Display the resulting trace messages recorded in the `/var/log/escript.log` file. To display the end of the log, issue the `show log escript.log | last` operational mode command:

```
[edit]
user@host# run show log escript.log | last
```

For more information about tracing event script processing, see: https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/automation-tracing-event-script-processing.html.

Agenda: Junos OS Event and SNMP Scripts

- Junos OS Events
 - Junos OS Event Policies
 - Junos OS Event Scripts
- Junos OS SNMP Scripts

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 39

Junos OS SNMP Scripts

The slide highlights the topic we discuss next.

SNMP Script Overview

- Map unused OIDs to Python Scripts
- Scripts need to be given permission of a user to execute

```
[edit system scripts snmp file filename]
user@host# set python-script-user username
```

SNMP Script Overview

Junos OS SNMP scripts, which are supported in Junos OS Release 16.1 and later releases, provide the flexibility to support custom MIBs until they are implemented in the Junos operating system. SNMP scripts are triggered automatically when the SNMP manager requests information from the SNMP agent for an object identifier (OID) that is mapped to an SNMP script for an unsupported OID. The script acts like an SNMP subagent, and the system sends the return value from the script to the network management system (NMS).

By default, Junos OS executes Python SNMP scripts with the access privileges of the generic, unprivileged user and group nobody. Starting in Junos OS Release 16.1R3, you can specify the user under whose access privileges the Python script will execute. To execute a Python SNMP script under the access privileges of a specific user, configure the `python-script-user username` statement at the `[edit system scripts snmp file filename]` hierarchy level.

```
[edit system scripts snmp file filename]
user@host# set python-script-user username
```

Note: to enable a user who does not belong to the file's user or group class to execute an unsigned Python automation script, the script file permissions must include read permission for others.

SNMP Script Case Study (1 of 7)

- Write a Python script to handle the following unsupported OIDs
 - oid .1.3.6.1.4.1.2636.13.61.1.9.1.1
 - oid .1.3.6.1.4.1.2636.13.61.1.9.1.1.1
- Return Specific values for SNMP get actions
- Use the SNMP get-next actions to retrieve the next OID in the MIB tree of data
- Test the script generating SNMP traps

```
show snmp mib get .1.3.6.1.4.1.2636.13.61.1.9.1.1
```

SNMP Script Case Study

In this case study the object IDs (oids) listed previously are currently not handled by the Junos SNMP management information base (MIB). You want to write a script that handles those oids when called. When the first oid is called, start with a value of 10, and the second will return a value of 211. These values are arbitrary and could be anything you want.

The SNMP get-next action doesn't return user-defined values, but returns the next oid in the MIB tree of data. Because of the oids you are using, your script will specify which oids are returned when you the get-next action is used.

The SNMP traps are generated using the `show snmp mib` command. The `show snmp mib` command allows for actions of `get`, `get-next`, or `walk`. Your script only needs to handle `get` and `get-next`. Finally, issue the `show snmp mib` command, with the included OID, as shown in the slide.

SNMP Script Case Study (2 of 7)

- Create Python script part 1

```
import jcs

def main():
    snmp_action = jcs.get_snmp_action()
    snmp_oid = jcs.get_snmp_oid()

    jcs.syslog("8", "snmp_action = ", snmp_action, " snmp_oid
= ", snmp_oid)

    if snmp_action == 'get':
        if snmp_oid == '.1.3.6.1.4.1.2636.13.61.1.9.1.1':
            jcs.emit_snmp_attributes(snmp_oid, "Integer32",
"10")
        elif snmp_oid == '.1.3.6.1.4.1.2636.13.61.1.9.1.1.1':
            jcs.emit_snmp_attributes(snmp_oid, "Integer32",
"211")
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 42

Create the Python Script, Part 1

The first step is to create the Python script. The only library you need to import is the `jcs` library. The `jcs.get_snmp_action()` method will retrieve the SNMP actions of get or get-next that are passed into the script. The `jcs.get_snmp_oid()` method retrieves the oid value which is passed into the script.

The `jcs.syslog()` method outputs to the system log a priority and a series of values. In the script, pass the priority value of 8, which means it is a user process. After the priority, we pass the SNMP action and the SNMP oid. For more information on the `jcs.syslog()` method see: https://www.juniper.net/documentation/en_US/junos/topics/reference/scripting/junos-script-automation-function-jcs-syslog.html.

Next, the script using `if` statements identifies whether the action is a get or get-next action. If it is a get action, the script tries to match the oid. If it finds a matching oid, it then uses the `jcs.emit_snmp_attributes()` method to return the oid and the corresponding 32bit number.

Continued on the next page.

SNMP Script Case Study (3 of 7)

- Create Python script part 2

```
elif snmp_action == 'get-next':
    if snmp_oid == '.1.3.6.1.4.1.2636.13.61.1.9.1.1':
        jcs.emit_snmp_attributes(".1.3.6.1.4.1.2636.13.61.1.9.1.1.1",
        "Integer32", "211")

    elif snmp_oid == '.1.3.6.1.4.1.2636.13.61.1.9.1.1.1':
        jcs.emit_snmp_attributes(".1.3.6.1.4.1.2636.13.61.1.9.1.1.2",
        "Integer32", "429")

if __name__ == '__main__':
    main()
```

- Save the script as sample_snmp.py

Create the Python Script, Part 2

If the `snmp_action` is `get-next`, then the script tries to match the received oid. If the received oid ends in `9.1.1` then the `jcs.emit_snmp_attributes` returns `9.1.1.1`, which is the next oid in the SNMP tree. If the received oid ends in `9.1.1.1` then the `jce.emit_snmp_attributes` returns a trap oid ending in `9.1.1.2`.

Once you have inputed the script, save it as `samp1_snmp.py`.

SNMP Script Case Study (4 of 7)

- Download the script to the vMX-1 device

```
[lab@jaut-desktop Chapter8]$ scp snmp_sample.py lab@vMX-1:/var/db/scripts/snmp/
```

- Set the file name and the two OIDs that will trigger the script; set the priority of the second OID higher than the default of 127

```
[edit system scripts snmp]
lab@vMX-1# set file sample_snmp.py oid
.1.3.6.1.4.1.2636.13.61.1.9.1.1

[edit system scripts snmp]
lab@vMX-1# set file sample_snmp.py oid
.1.3.6.1.4.1.2636.13.61.1.9.1.1.1

[edit system scripts snmp]
lab@vMX-1# set file sample_snmp.py oid
.1.3.6.1.4.1.2636.13.61.1.9.1.1.1 priority 120
```

Download the Script

Copy the file to the vMX-1 device. Because it is an SNMP script, it needs to be stored in the /var/db/scripts/snmp/ directory.

Next, configure two oids that will trigger the script. Set the priority of the second OID higher than the default of 127. (1 is the highest priority, and 255 is the lowest priority.) You can use 120. The longer oid needs to be given priority. Because the two oids are overlapping, the longer oid would never get called unless given priority.

SNMP Script Case Study (5 of 7)

- Configure the language python statement

```
[edit system scripts]
lab@vMX-1# set language python
```

- Configure the user under whose access privileges the script executes

```
[edit system scripts]
lab@vMX-1# set snmp file sample_snmp.py python-script-user
lab
```

- Commit the configuration

```
lab@vMX-1# commit and-quit
```

Enable the Script

If you haven't already set it, set the language to Python, as shown in the slide. Because the script is written in Python, you will need to configure the user under whose access privileges the script executes, as shown in the slide.

Note: If you do not configure the `python-script-user` statement, then by default Junos OS executes Python SNMP scripts under the access privileges of the user and group `nobody`.

Finally, commit the configuration.

SNMP Script Case Study (6 of 7)

- Review configuration

```
[edit]
lab@vMX-1# show system scripts
snmp {
    file sample_snmp.py {
        oid .1.3.6.1.4.1.2636.13.61.1.9.1.1;
        oid .1.3.6.1.4.1.2636.13.61.1.9.1.1.1 {
            priority 120;
        }
        python-script-user lab;
    }
}
language python;
```

Review the Configuration

This is the final configuration format.

SNMP Script Case Study (7 of 7)

- Test the script by generating an SNMP request

```
lab@vMX-1> show snmp mib get .1.3.6.1.4.1.2636.13.61.1.9.1.1
juniperMIB.13.61.1.9.1.1 = 10
lab@vMX-1> show snmp mib get .1.3.6.1.4.1.2636.13.61.1.9.1.1.1
juniperMIB.13.61.1.9.1.1.1 = 211
lab@vMX-1> show snmp mib get-next .1.3.6.1.4.1.2636.13.61.1.9.1.1.1
snmpSetSerialNo.0 = 0

lab@vMX-1> show log messages | last
May 16 15:19:29 vMX-1 mgd[23559]: UI_CMDLINE_READ_LINE: User 'lab',
command 'show snmp mib get-next .1.3.6.1.4.1.2636.13.61.1.9.1.1 '
May 16 15:19:29 vMX-1 cscript.crypto: snmp_action = get-next
snmp_oid = .1.3.6.1.4.1.2636.13.61.1.9.1.1
May 16 15:19:29 vMX-1 cscript.crypto: snmp_action = get-next
snmp_oid = .1.3.6.1.4.1.2636.13.61.1.9.1.1.1
May 16 15:19:30 vMX-1 cscript.crypto: snmp_action = get-next
snmp_oid = .1.3.6.1.4.1.2636.13.61.1.9.1.1.2
```

Test the Script

You can generate the SNMP get and get-next action requests by issuing the `show snmp mib get` with the object id, as shown in the slide. The first two commands show the expected output. The `get` command returns the specified values.

The third command, `show snmp mib get-next .1.3.6.1.4.1.2636.13.61.1.9.1.1.1` returns an `snmpSetSerialNo.0 = 0` message. When you look at the code in the script, the get-next action looks up a related oid. When you look at the log messages, you can see the command that was issued. The command issued was a get-next action with an oid ending in 9.1.1. You can see that it triggers a get-next lookup of oid 9.1.1.1, and that in turn triggers the lookup of snmp oid 9.1.1.2. Once finished, it displays the `snmpSetSerialNo.0 = 0` message.

Summary

- In this content, we:
 - Identified Junos OS events
 - Created Junos OS event policies
 - Created Junos OS event scripts
 - Created Junos SNMP scripts

We Discussed:

- How to identify Junos Events;
- Junos OS event policies;
- Junos OS event scripts; and
- Junos OS SNMP scripts.

Review Questions

1. Why would you want to pass permissions to a Python event script?
2. Why do event scripts need additional permissions to run?
3. What is the Junos OS command to generate an SNMP trap?

Review Questions

- 1.
- 2.
- 3.

Lab: Event Scripts

- Generate and filter events
- Create event policies
- Create event scripts

Lab: Junos Event Policies and Event Scripts

This slide lists the objectives for this lab.

Review Questions

1.

It is a good idea to pass permissions to an event script so that the permissions are not stored in the script.

2.

Event scripts need additional permissions to run because, by default, they run under the user and group nobody.

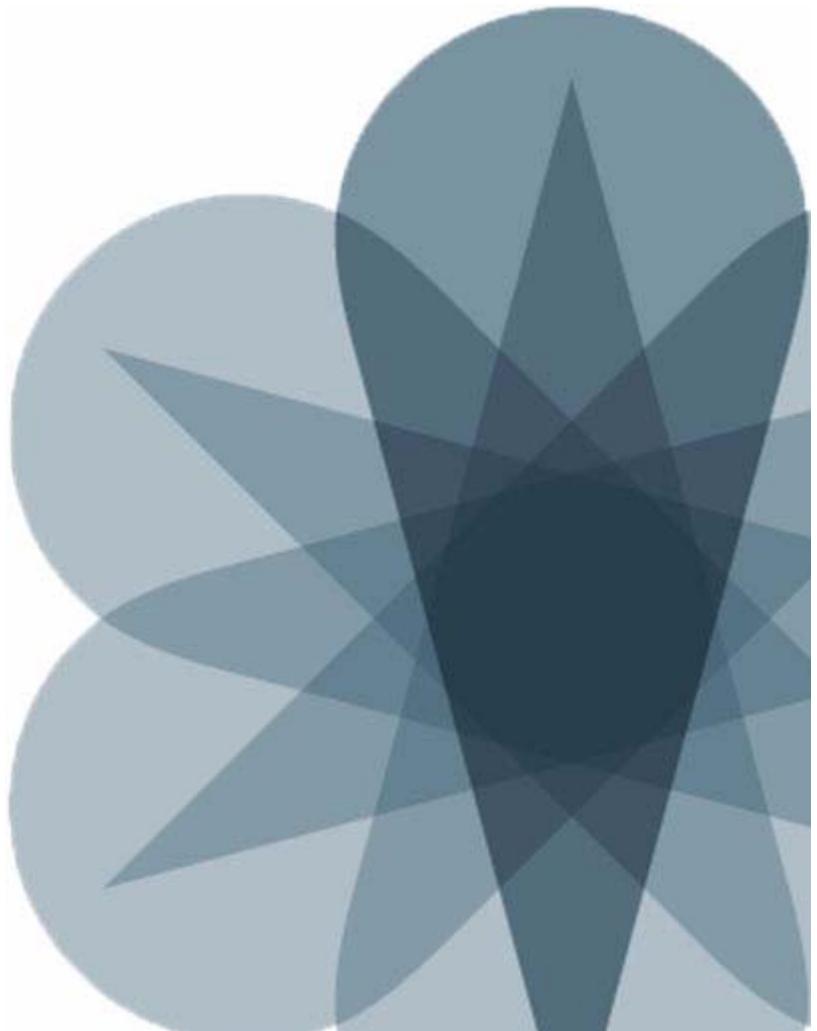
3.

The Junos OS command to generate an SNMP trap is `show snmp mib get .1.3.6.1.4.1.2636.13.61.1.9.1.1`



Junos Platform Automation and DevOps

Chapter 11: YANG



Objectives

- After successfully completing this content, you will be able to:
 - Describe the YANG Protocol
 - Explain the syntax of YANG

We Will Discuss:

- The YANG Protocol;
- The capabilities of YANG;
- How to issue Junos OS RPC commands using YANG; and
- How to configure Junos OS using YANG

Agenda: YANG

- YANG Overview
- YANG Modules
- YANG Statements and Syntax

© 2017 Juniper Networks, Inc. All rights reserved.

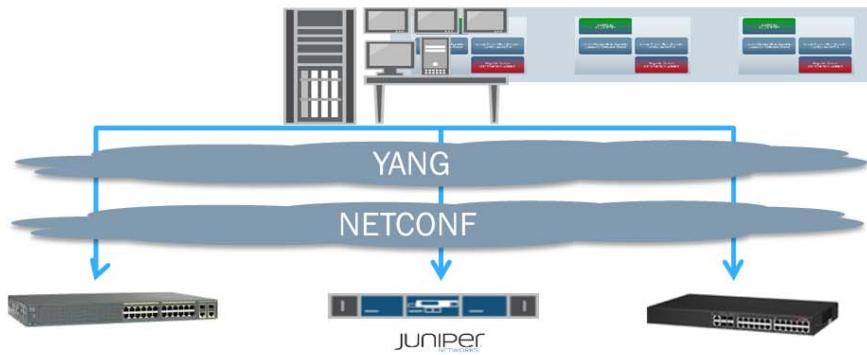
 Worldwide Education Services

www.juniper.net | 3

YANG Overview

The slide lists the topics we will discuss. We discuss the highlighted topic first.

Why YANG?



NETCONF provides a standard connection protocol to communicate with network devices

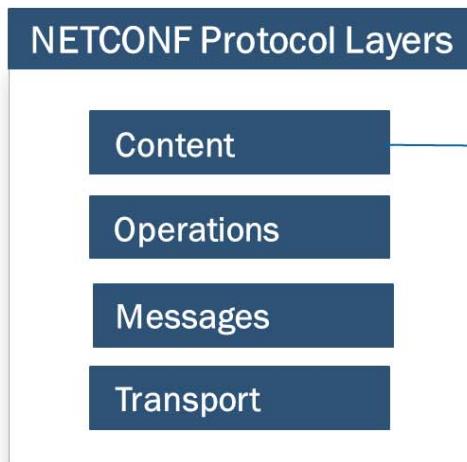
YANG provides the means to define the content carried via NETCONF, both data and operations

Why YANG?

NETCONF provides a standardized way to connect to networking devices. Previously, we learned how the NETCONF protocol specifies network operations such as `<get-config>`, `<edit-config>`, `<lock>`, and `<unlock>` to manage the configuration database. However, once you have locked the configuration and specified that you want to edit a configuration, the actual data passed at the Content layer of the NETCONF protocol is not standardized, because Content layer commands vary by vendor. YANG by itself does not accomplish the goal of complete vendor interoperability, but does get us one step closer. The YANG model does for network automation what the OSI model has done for networking in general. Just as the OSI model gives us a vendor-neutral, standardized method or model of looking at what is going on inside of a networking stack, the YANG model provides a vendor-neutral, standardized method of looking at the data that is being sent across the Content layer of a NETCONF session. As stated in RFC 6244: “YANG provides the means to define the content carried via NETCONF, both data and operations.”

We will discuss the realization of the vendor interoperability goal in the chapter on OpenConfig. But before we can talk about OpenConfig, you first need a solid understanding of YANG. Much of the content of this chapter comes from the YANG RFCs: RFC 6020, RFC 6244 and RFC 7950.

What Does the YANG Model Do?



YANG Models Contain:

1. Configuration
2. State data
3. RPCs
4. Notifications

What Does the YANG Model Do?

First, YANG can be used to build models of the active and candidate configuration. Second, YANG can be used as a model to represent state data, such as the status of OSPF neighbors, BGP links, or interface status. Third, YANG can be used to issue RPCs to the YANG API and execute most of the Junos OS operational mode commands. Finally, YANG can also be used to model a request to subscribe to device notifications.

In short, YANG can be used to model any activity or function that can be performed over a NETCONF session with a Juniper device. After explaining the semantics of YANG, we will look at some examples.

XML Schema Vs. YANG Model

XML Schema

```

<xsd:element name="get-interface-information">  <xsd:annotation>
  <xsd:documentation>Show interface information</xsd:documentation>
  <xsd:appinfo>
    <flag>vsys-ok</flag>
    <flag>current-product-support</flag>
    <require>view</require>
  </command-forwarding>
  <chassis-selector>local</chassis-selector>
  <exclude-target>multi-chassis</exclude-target>
  <exclude-target>scc</exclude-target>
  <flag>hide-args</flag>
</command-forwarding>
</xsd:appinfo>
</xsd:annotation>
<xsd:complexType>
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="undocumented" minOccurs="0"/>
      <xsd:element ref="junos:comment" minOccurs="0"/>
      <xsd:element name="routing-instance" minOccurs="0">
        <xsd:annotation>
          <xsd:documentation>Name of routing instance</xsd:documentation>
          <xsd:appinfo>
            <flag>vsys-ok</flag>
            <flag>current-product-support</flag>
            <alias>instance</alias>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="all">
              <xsd:annotation>
                <xsd:documentation>All instances</xsd:documentation>
                <xsd:appinfo>
                  <flag>vsys-ok</flag>
                  <flag>current-product-support</flag>
                </xsd:appinfo>
              </xsd:annotation>
              <xsd:enumeration value="instance-name">
                ...
              </xsd:enumeration>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...
...Trimmed...
  
```

YANG Model

```

rpc get-interface-information {
  description "Show interface information";
  input {
    leaf routing-instance {
      description "Name of routing instance";
      type string;
    }
    leaf satellite-device {
      description "Name of satellite device";
      type string;
    }
    leaf level {
      type enumeration {
        enum "extensive" {
          description "Display extensive output";
        }
        enum "statistics" {
          description "Display statistics and detailed output";
        }
        enum "media" {
          description "Display media information";
        }
      }
    }
    leaf level-extra {
      type enumeration {
        enum "detail" {
          description "Display detailed output";
        }
        enum "terse" {
          description "Display terse output";
        }
        enum "brief" {
          description "Display brief output";
        }
        enum "descriptions" {
          description "Display interface description strings";
        }
      }
    }
  }
  ...
}
...
...Trimmed...
  
```

XML Schema Vs. YANG Models

YANG models like the Junos XSD Schema illustrate the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes and describes the interaction between those nodes.

The slide shows the Junos XSD schema and the YANG file that contains the `get-interface-information` RPC data models

Advantages of YANG

- Human readable
- Better management through modules and submodules
- Ability to augment existing modules
- Apply constraints

YANG is More Readable

One of the key advantages of using YANG at the content layer of a NETCONF session is that it is more readable. You won't find a YANG equivalent of the `| display xml` or `| display xml rpc` commands used for the API because these are not needed. You can display the YANG module and easily pull the needed information directly from the YANG model itself.

Better Model Management

YANG allows you to better manage your data model by allowing you to use multiple modules and also submodules that can be included or imported by your module. You can include, import, and augment your modules. You will see the difference between import and include shortly.

Using modules and submodules enables Juniper to keep the size of the model down. As an example, the `operational-command.xsd` file used for the XML API is only 417 MB in size. The YANG equivalent file is called the `juniper-command` module. It isn't a single file but consists of over 100 different modules, most of which are under 20 MB.

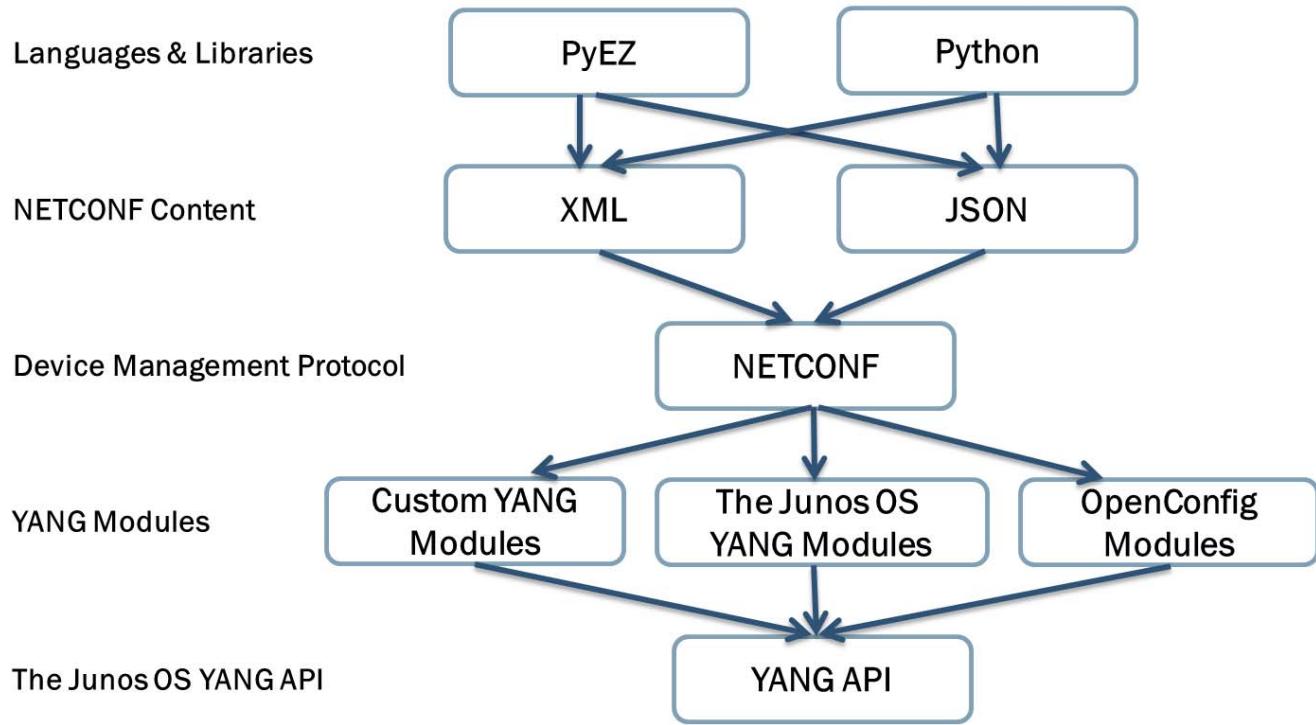
Augment Modules

Augmenting a module gives you the ability to add onto an existing module with additional nodes. The additional nodes can be structured in a way that makes them available only if specific criteria are selected.

Apply Constraints

Constraints restrict the data type and format of values that can be submitted. These constraints can be used to validate the data by either a YANG validator or the NETCONF server.

YANG Development Stack



© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 8

YANG Development Stack

When working with the Junos OS YANG API, there are three types of YANG modules that Junos supports. First, there are the native Junos OS YANG modules provided by Juniper. Second, there are custom YANG modules that developers can create. Third, there are the OpenConfig modules that provide interoperability with other vendor devices. This chapter will focus on the Junos OS native YANG modules. The custom YANG modules and OpenConfig Modules will be covered in a later chapter.

All of the communication with the YANG API is either on-box or through a NETCONF session. When communicating with the Junos OS YANG API, the RPCs can be sent over NETCONF and encapsulated in either XML or JSON, as seen in the Content layer on the slide. To help automate a Junos device, there are many different languages and libraries available. In this course, we will focus on Python and Junos PyEZ.

Agenda: YANG

- YANG Overview
- YANG Modules
- YANG Statements and Syntax

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 9

YANG Modules

The slide highlights the topic we discuss next.

YANG Modules

- YANG Modules begin with header statements

```

/*
 * Copyright (c) 2017 Juniper Networks, Inc.
 All rights reserved.
*/
module clear {
    namespace "http://yang.juniper.net/yang/1.1/jrpc";
    prefix jrpc;
    import junos-extension {
        prefix junos;
    }
    organization      "Juniper Networks, Inc.";
    description       "Junos YANG module for RPC";
}

```

Module Header Statements

In YANG, the modules and submodules contain the data model. The beginning of each module contains header statements to describe and give information about the module. The slide shows the header section of the clear.yang module.

The description is followed by the name of the module; the slide shows that this is the `clear` module. The name is followed by the namespace and its prefix. The namespace is the XML namespace, which is used to uniquely identify each module and keep identifiers in one module from conflicting with identifiers with the same name in another module. For example, the Junos OS configuration module uses the `http://yang.juniper.net/yang/1.1/jc` namespace, while the Junos OS operational commands use the `http://yang.juniper.net/yang/1.1/jrpc` namespace.

Note that these namespace identifiers just mentioned are not URLs but are URIs, or *Universal Resource Identifiers*. These identifiers do not link to an Internet address even though they have the appearance of a URL, or *Universal Resource Locator*. For more information on URIs, please see RFC 3986 - *Uniform Resource Identifier (URI): Generic Syntax*.

Instead of using the whole `http://yang.juniper.net/yang/1.1/jrpc` namespace each time, you can use the prefix `jrpc`: as shown on the slide. The module name and namespace declaration are followed by any import and include statements as well as their prefixes. We will discuss import and include statements shortly. Finally, the header can include a declaration and a description as seen in the slide.

The module revision statement notes any changes or revisions that have occurred to the module. The clear.yang module does not show any revisions.

The YANG module definition statements are the content of the module. The clear.yang file has approximately 135,000 definition lines of code. Juniper tries to keep the YANG modules under 20 MB in size for better manageability.

YANG Modules

- YANG module headers are followed by Type Definitions and Declarations

- Type Definitions – The data type definitions

```
typedef ipaddr {
    type string;
}
```

- Declarations – The main content of the YANG file

```
rpc clear-mac-rewrite-error {
    description "Clear mac-rewrite error on an interface";
    input {
        leaf interface-name {
            description "Name of interface";
            type interface-device;
        }
    }
}
```

Type Definitions

After the header information, the local type definitions are displayed. The slide shows the type definition of an IP address. Notice that an IP address is a string. We will look at how the proper format is enforced later in this chapter.

Declarations

The type definitions are followed by the actual YANG model declarations. These declarations can be either configuration or RPCs corresponding to operational mode commands. The slide shows the `clear-mac-rewrite-error` YANG RPC. Note that it allows for the input of an interface name.

The Junos OS YANG Modules

- The Junos OS has four YANG Modules

- The configuration module

- <http://yang.juniper.net/yang/1.1/jc>

- The juniper-command module

- <http://yang.juniper.net/yang/1.1/jrpc>

- The junos-extension module

- <http://yang.juniper.net/yang/1.1/je>

- The junos-extension-odl module

- <http://yang.juniper.net/yang/1.1/jodl>

Configuration and Command Modules

Juniper has four native YANG modules, which are shown in the slide. The first two modules are the configuration and juniper-command modules. Just as the Junos OS XML API has separate .xsd files for the configuration and the operational commands, YANG also has separate groups of YANG modules for the configuration and operational commands. The slide shows each of the main Junos OS YANG modules as well as their related namespace.

Junos Extension Modules

YANG has a predefined set of statements, or keywords. YANG gives YANG model creators the ability to add additional statements to the YANG language that expand functionality. These added statements are called extensions because they extend the abilities of the language. To enable YANG to work better with the Junos OS, Juniper has added some extensions specific to the Junos OS. These extensions are stored in the junos-extension module.

The junos-extension-odl Module

The junos-extension-odl module contains the Junos OS Output Definition Language (ODL) statements that can be used to create and customize formatted ASCII output for RPCs executed on devices running the Junos OS.

Let's take a closer look at the configuration module, the juniper-command module, and the junos-extension module.

The Configuration Module

- Contains the Junos OS configuration Hierarchy
- Download from:
 - <http://www.juniper.net/support/downloads/junos.html>.

- Generate the Junos configuration YANG module

```
lab@vMX-1> show system schema module
configuration format yang output-directory
/var/tmp/
```

The Junos OS Configuration YANG Module

The Junos configuration module defines the Junos OS configuration hierarchy. This file does not contain the configuration data but contains the YANG model that configuration data is validated against. This configuration module is used by the MGD to validate data it receives over a NETCONF session that is using the YANG API. You will see an example of this shortly.

The Junos OS YANG configuration module can be downloaded from <http://www.juniper.net/support/downloads/junos.html>. The configuration module can also be generated from a Junos OS device by issuing the `show system schema module configuration format yang output-directory /var/tmp/` command.

The juniper-command Module

- Operational command hierarchy
- Generate the juniper-command module

```
lab@vMX-1> show system schema module juniper-command
```

- List the YANG modules that are part of the juniper-command module.

```
lab@vMX-1> file list /var/home/[user]/*.yang
```

The juniper-command Module

The juniper-command module is not one YANG module but dozens of different YANG modules on the lab vMX device. The Juniper Networks juniper-command YANG module represents the operational command hierarchy and collective group of modules that define the remote procedure calls (RPCs) for Junos OS operational mode commands. When you generate the juniper-command module on the device, it includes both native Junos OS RPCs as well as any standard or custom RPCs that have been added to the device.

Due to the large number of operational commands on devices running the Junos OS, the juniper-command module comprises multiple modules, each in its own separate file. There is a module for each top-level operational command group (clear, file, monitor, and so on) where there is at least one command within that hierarchy with an RPC equivalent. There is also a separate module for each area within the `show` command hierarchy.

The default output directory changed in version 17.1. Starting in Junos OS Release 17.1, the juniper-command modules are placed in the current working directory, which defaults to the user's home directory. In earlier releases, the files were placed in: `/var/tmp`.

You can generate the juniper-command module on the local device using the following command:

```
lab@vMX-1> show system schema module juniper-command format yang
```

The output files are placed in the user's home directory.

```
lab@vMX-1> file list /var/home/[user]/*.yang
```

You will have a chance to see these in action soon.

Junos Extension Module

- The Junos Extension Module is imported by the Junos configuration.yang file

```
module configuration {
    namespace "http://yang.juniper.net/yang/1.1/jc";
    prefix jc;
    import junos-extension {
        prefix junos;
    }
```

 Imports the Junos extensions

The Junos Extension Module is referenced with the prefix `junos`:

```
leaf forwarding-class {
    description "Classify packet to forwarding class";
    type string {
        junos:posix-pattern "^.{1,64}$";
        junos:pattern-message "Must be string of 64 characters or less";
    }
}
```

 The Junos Extension Modules contains the Junos extensions of posix-pattern and pattern-message

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 15

The `junos-extension` Module

These extensions include the `must` and `must-message` statements, which identify configuration hierarchy constraints that contain special keywords. The module also contains statements that you can include in custom RPCs to define a CLI command for the RPC, and to specify details about the action script to invoke when the RPC is executed. The Junos OS extension module, called `junos-extension.yang`, is automatically downloaded within the Junos OS configuration YANG file. The extension file is only 57 lines long (of course this will vary by release) and follows the format of a standard YANG file. I've included it here below so that you can see it and a whole YANG file.

At the top of the slide, you see the first six lines of the Junos OS configuration.yang file. This is where the `junos-extension` YANG module is imported. Once the `junos-extension` YANG file is referenced in the import statement, it can be used within the importing module. In the slide, you can see the `posix-pattern` extension statement in use. The `pattern-message` is displayed if the string entered does not evaluate to true.

The `junos-extension` YANG module can be downloaded from <http://www.juniper.net/support/downloads/junos.html>; The `junos-extension` module can also be generated in the CLI of a device running Junos OS by issuing the `show system schema module junos-extension format yang output-directory /var/tmp/` command.

Continued on the next page.

The `junos-extension` Module (contd.)

```

/*
 * Copyright (c) 2016 Juniper Networks, Inc.
 * All rights reserved.
 */

module junos-extension {
    namespace "http://yang.juniper.net/yang/1.1/je";
    prefix junos;

    organization
        "Juniper Networks, Inc.";

    description
        "This module contains definitions for Junos YANG extensions.';

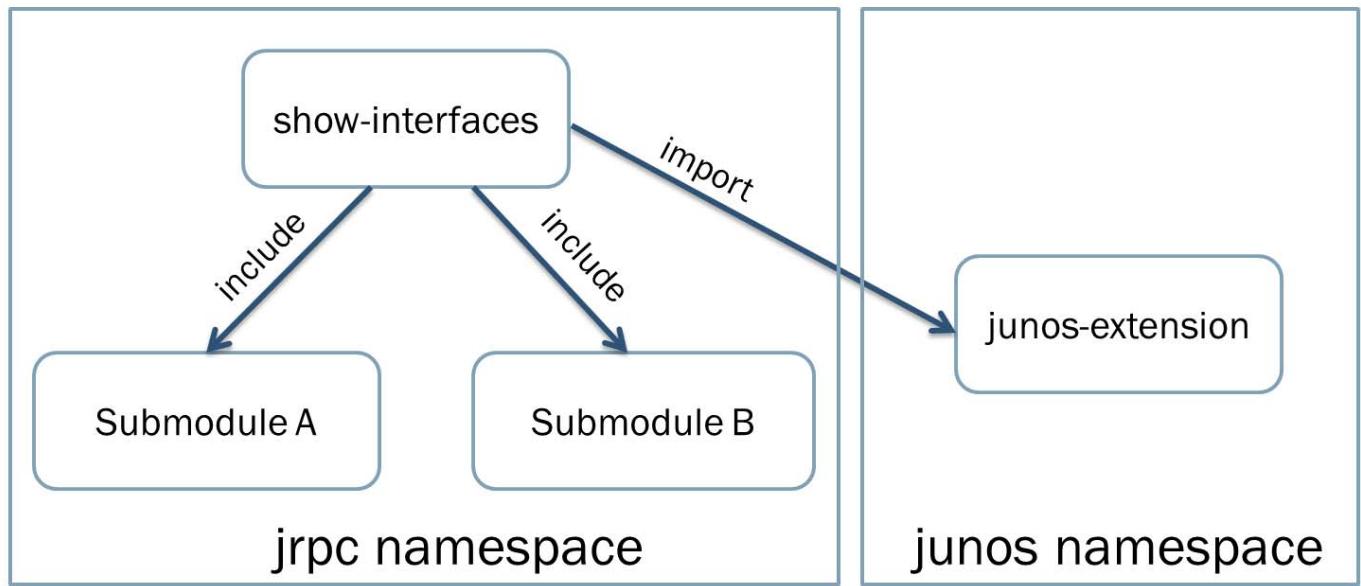
    extension must {
        argument "stmt-algebra";
        description "
            expression of configuration-statement paths having special
            keywords like any, all, unique";
    }
    extension must-message {
        argument "message-string";
        description "
            When specifying the junos: must constraint, a must-message should
            also be specified so that when the constraint is not met, the
            warning message informs users what the constraints are.

            If the must-message is missing, the input file will compile,
            but the warning message will say (null) which won't be
            helpful for users to fix the problem";
    }
    extension posix-pattern {
        argument "value";
        description "
            expression of pattern in posix format";
    }
    extension pattern-message {
        argument "value";
        description "
            Error message in case of mismatch";
    }
    extension command {
        argument "command";
        description "
            The junos cli command for this rpc";
    }
    extension action-execute {
        description "
            The junos action execute for rpc";
    }
    extension script {
        argument "script";
        description "
            The junos action execute script for rpc";
    }
}

```

Including and Importing Modules

- Include – same namespace
- Import – different namespace



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 17

Including Modules

Submodules are a way to take a model and break it up into manageable pieces. Submodules are partial modules that contribute definitions to a module. A module can include one or more submodules belonging to the same namespace as the parent module. When a module includes a submodule, the parser copies the code from the included submodule into the parent module when being parsed. You have to be careful that an included module does not call the parent module; this would create a circular reference.

In the slide, we see the `show-interfaces` module and have included two hypothetical submodules. (Currently, the Junos OS YANG modules do not have submodules.)

Importing Modules

When a module or submodule imports an external module, it creates a reference to that module, which enables the importing module to reference definitions in the external module. When you use a statement from an imported module, you have to qualify that statement with the prefix of the imported namespace, as we saw on the previous slide. As seen in this slide, the `show-interfaces` module imports the `junos-extension` module and uses `junos :` as the extension prefix.

Agenda: YANG

- YANG Overview
- YANG Modules
- YANG Statements and Syntax

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 18

YANG Statements and Syntax

The slide highlights the topic we discuss next.

Data Modeling Structures

- Leaf nodes
- Leaf-list nodes
- Container nodes
- List nodes

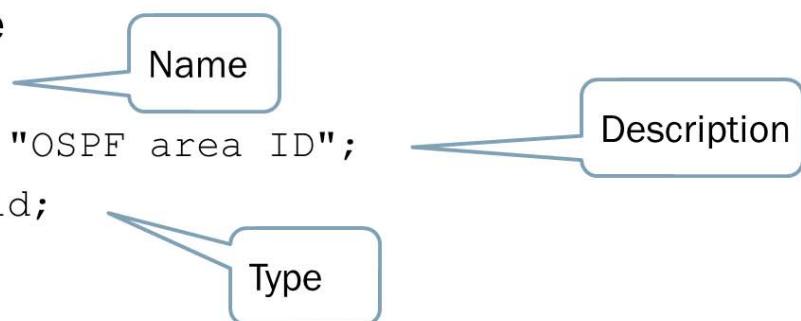
Data Modeling Structures

Within a module, there are four types of nodes you can use to build your model. The four types are listed on the slide and will be discussed next.

Leaf Nodes

- Leaf nodes contain a name, a description, and a data type
- YANG documents contain many leaf nodes
- Leaf nodes are the most basic YANG node type
- YANG Example

```
leaf area {  
    description "OSPF area ID";  
    type areaid;  
}
```



Leaf Nodes

The leaf node is perhaps the simplest of the data modeling structures. The most common statements within a leaf node are `name`, `description`, and `type`. In the slide, we see the `area` leaf node. The leaf area comes from the RPC `get-ospf3-neighbor-information`.

The `description` statement gives a human-readable description about the data the leaf node contains. The `type` statement specifies the type of data the leaf contains. There are built-in data types and data types that are user defined. We will learn more about data types on the following pages.

The examples on the slide show how leaf node data can be encoded in either XML or JSON.

Leaf Node Built-in Data Types

- Leaf nodes can specify any of the following data types

Basic Data Type	Description
binary	Any binary data
bits	A set of bits or flags
boolean	True or false
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	One of an enumerated set of strings
identityref	A reference to an abstract identity
instance-identifier	A reference to a data tree node
Int8, int16, int32, int64	8-bit, 16-bit, 32-bit, 64-bit signed integer
leafref	A reference to a leaf instance
string	A character string
uint8, uint16, uint32, uint64	8-bit, 16-bit, 32-bit, 64-bit unsigned integer
union	Choice of member types

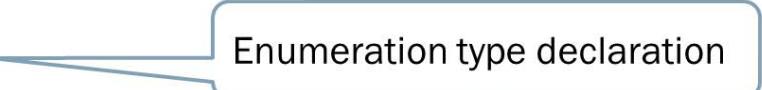
YANG Built-in Types

YANG comes with a predefined set of data types. The data types found in YANG are similar to those found in many languages, with some additional data types used specifically for YANG and networking. The table on the slide shows the YANG built-in data types. We will discuss the enum data type on the next page, and we will explore some of the other built-in types throughout the course. For an in-depth explanation of each of these built-in types, see RFC 7950, *The YANG 1.1 Data Modeling Language*.

The enum Statement

- The enum statement gives a choice of values

```
rpc get-interface-information {
    description "Show interface information";
    input {
        leaf level-extra {
            type enumeration {
                enum "detail" {
                    description "Display detailed output";
                }
                enum "terse" {
                    description "Display terse output";
                }
                enum "brief" {
                    description "Display brief output";
                }
                enum "descriptions" {
                    description "Display interface description strings";
                }
            }
        } . . . Trimmed . . . }
    }
```



Enumeration type declaration

The enum Statement

One of the more common data types is the enumeration data type, or enum. The enum data type enables the model to specify a finite list of possible string values. The slide shows the `get-interface-information` RPC, which is equivalent to the `show interfaces` CLI command.

Notice that the `leaf level-extra` has an enumeration with four defined enum strings: `detail`, `terse`, `brief`, and `descriptions`.

The `typedef` Statement

- Leaf Nodes specify a type. Types have to be defined at the top of the YAML document.
- New type defined

```
typedef hostname {
    type string;
}
```

New type defined

- The `typedef` Statement in use

```
leaf name {
    description "Name or address of server";
    type hostname;
}
```

New defined type in use

```
Juniper YANG Typedef statements
typedef daemon {
    type string;
}
typedef filename {
    type string;
}
typedef client-filename {
    type string;
}
typedef hostname {
    type string;
}
typedef ipaddr {
    type string;
}
typedef sysid {
    type string;
}
typedef interface-device {
    type string;
}
```

The `typedef` Statement

The `typedef` statement allows you to create your own data types out of built-in and other user-defined types. The top of the slide shows how a new type, called `hostname`, is defined. Notice that it is defined using the built-in type `string`.

Once a derived type is defined with the `typedef` statement, it is used the same way as a built-in type. The bottom of the slide shows a `name` leaf with a type of `hostname`.

The slide shows how a new type called `hostname` is defined and then later used in the leaf node called `name`. We were not required to create a new type called `hostname`; the leaf called `name` could have used a data type of `string`. The `hostname` data type is derived from a `string`. The reason that the `hostname` data type was created was to improve clarity about the data that the leaf node contains. If you look at the top of any of the Juniper YANG modules you will see a long list of `typedef` statements; most of these create more intuitive data types out of built-in data types, like `string` and `int16`.

Leaf Node Sub-Statements

- Leaf nodes can contain other constraints and sub statements

Statement	Description
config	If "config" is "true", the definition represents configuration.
default	Specifies a default value for a leaf node
description	Gives a human readable description of the statement
if-feature	Makes a leaf statement conditional. Evaluates to true if the feature is supported by the device
mandatory	Specifies that a node must exist in a data tree
must	Puts an Xpath constraint on data that must be true for the data to validate
reference	Holds a cross-reference to an external document
status	Specifies whether a statement is “current,” “deprecated,” or “obsolete”
type	Specifies a built-in or derived data type
units	Specifies units such as bits/second or MB
when	Makes the leaf statement conditional

The Leaf Node Substatements

In addition to the type statement, leaf nodes can contain any of the statements listed on the slide. Most of the substatements listed are self-explanatory but if you wish to go into more depth, refer to RFC 7950 - *The YANG 1.1 Data Modeling Language*.

Leaf-list Nodes

- A leaf list node is a leaf node that can hold multiple values
- Leaf-list example

```
leaf-list interface {
    description "Restrict SNMP requests to interfaces";
    type interface-name;
}
```

Leaf-list declaration

- XML representation

```
<interface>lo0</interface>
<interface>ge-0/0/0</interface>
<interface>ge-0/0/1</interface>
<interface>lt-0/0/0</interface>
```

The `leaf-list` Nodes

The `leaf-list` nodes are similar to `leaf` nodes except that they enable users to define a sequence of values of a particular type, rather than defining just a single value. The example in the slide comes from the `SNMP` container in the `configuration.yang` module. The `leaf-list interface` enables users to specify a list of interfaces. The `leaf-list interface` uses the `interface-name` data type; any valid interface name is acceptable. Notice from the XML that there is not a single value represented as there is in a `leaf` node, but a list of values is represented.

With a `leaf-list`, the model can specify a minimum number of items in the `leaf-list` with the `min-elements` statement and a maximum number of elements with the `max-elements` statement.

There are a few other differences between `leaf-list` attributes discussed here and `leaf` node attributes that were presented on the previous slide. For more information, compare 7.6.2 to 7.7.3 in RFC 7950 - *The YANG 1.1 Data Modeling Language*.

Containers

- Containers are nodes used to show hierarchy, but don't contain values
- Containers also contain other nodes and containers such as leaf nodes and leaf list nodes
- Container Example

```
container packet-statistics {
    leaf ospf-packet-type {
        type string;
    }
    leaf packets-sent {
        type uint64;
    }
    leaf packets-received {
        type uint64;
    }
    . . . trimmed . . .
}
```

Containers

A container is a data node within a module that is unique in the data tree. A container has no value, but rather a set of child nodes. The slide lists the packet statistics container from the following hierarchy:

```
rpc get-ospf3-statistics-information {
    output {
        container ospf3-statistics-information {
            container ospf-statistics {
                container packet-statistics {
```

Notice how containers can be nested.

List Nodes

- List Nodes are similar to Leaf list nodes. Leaf lists contain repeating values. List nodes contain a series of repeating nodes

```
list filter {
    key name;
    ordered-by user;
    description "Define an IPv4 firewall filter";
    uses inet_filter;
}
```

List Nodes

The list node is similar to a container in that it can contain leafs, leaf-lists and other structures. The list node provides the additional ability to define a series of entries. The slide shows a list called `filter` and is part of the `firewall` container in the `configuration.yang` file. The `list filter` enables users to configure a list of filters.

List nodes contain a `key` statement. The `key` statement lists identifiers for one or more child leafs of the list. The values of the leafs specified in the `key` are used to uniquely identify a list entry.

As seen in the slide, the `key` is `name` in this case. The `name` is the name of the filter. In the Junos OS, you cannot configure a filter without a name because the name keeps one filter distinguishable from other filters.

The filter also has an `ordered-by` statement which defines whether the order of entries within a list are determined by the user or by the system. The `ordered-by` statement can have a value `system` or `user`. If not present, ordering defaults to `system`.

The final filter has a `uses` statement. The `uses` statement references a grouping of code by taking one argument, which is the name of the grouping. The `inet_filter` contains the specific structures and statements of the model needed for setting a filter. We discuss the grouping statement on the next page.

Grouping

- Groupings are similar to functions or methods. They are used to create a reusable set of nodes that can be reused within the YANG model

```
grouping inet_filter {
    leaf name {
        description "Filter name";
        type string {
            junos:posix-pattern "!^((__.*|.{65,}))$";
            junos:pattern-message "Must be a non-reserved string of
64 characters or less";
        }
    }
    . . . Trimmed . . .
}
```

Grouping

The list filter from the previous page uses `inet_filter`. The `inet_filter` that is being referenced is the grouping `inet_filter` shown on the slide. If you look at the whole `inet_filter` grouping in the `configuration.yang` file, you will notice that it is nearly 800 lines long. The `inet_filter` grouping is also referenced six times in the `configuration.yang` file.

By using the grouping structure, six times 800, or 4800 lines in the model have been reduced to a single instance of 800 lines. Grouping also has the advantage of having a single point of management; if the `inet_filter` segment of the configuration model needs to be updated or modified, this can be updated in one location.

Choice and Case Statements

- Choice and case statements exist within a container and enables the implementation of one of a set of nodes

```
grouping inet_filter {
    list term {
        container from {
            choice protocol_choice {
                case case_1 {
                    leaf-list protocol {
                        type string;
                    }
                }
                case case_2 {
                    leaf-list protocol-except {
                        type string;
                    }
                }
            }
        }
    }
}
```

Choice and Case Statements

Sometimes in a configuration there are two or more features that are mutually exclusive; a `choice` statement allows a YANG model to specify one out of multiple sets of configuration statements.

The `choice` statement on the slide comes from the `grouping inet_filter`. This `choice` statement allows you to specify either a list of specific protocols to match or a list of specific protocols that you want to exclude from the match. The `choice` statement in the slide shows that you cannot configure both; you must choose either `case_1` or `case_2`.

The Augment Statement

- The augment statement allows vendors to add additional nodes to a model
- If the IETF or OpenConfig create YANG models that don't provide all the necessary Junos OS functionality, It can be added to with augment statements

```
augment /system/login/user {  
    when "vendor != 'Juniper'";  
    leaf uid {  
        type uint16 {  
            range "1000 .. 30000";  
        }  
    }  
}
```

The Augment Statement

You can use the `augment` statement to add nodes to an existing model. This is useful when working with devices from multiple vendors and when working with OpenConfig. This example adds the leaf node `uid` to the `/system/login/user` hierarchy only when `"vendor != 'Juniper'"`. In this case, the `uid` leaf would only be added if the model was being applied to a non-Juniper device.

Also, note that there is a `range` statement on the slide. The `range` statement is a constraint that restricts the acceptable values for `uid` to the range of 1000 through 30000.

The rpc Statement

- The rpc statement is used to define an rpc Operation

```
rpc get-log {
    description "Show contents of log file";
    input {
        leaf filename {
            description "Name of log file";
            type string;
        }
    }
    output {
        leaf file-content {
            type string;
            description "Show file contents";
        }
    }
}
```

The rpc Statement

The rpc statement is used to define an RPC operation. The rpc statement is followed by the name of the YANG RPC that is issued to the Junos OS YANG API.

The rpc statement has two optional statements: `input` and `output`. The `input` statement is used to define input parameters for the RPC. The slide shows the `leaf filename` as an input parameter. So executing this RPC in a NETCONF session using XML would look like this:

```
<rpc>
  <get-log>
    <filename>messages</filename>
  </get-log>
</rpc>
]]>]]>
```

Continued on the next page.

The rpc Statement (contd.)

The output statement defines the output parameters for the RPC operation. The output statement shows what information is returned in the <rpc-reply> tags of the NETCONF session. The <rpc-reply> for the RPC shown on the previous slide looks like this:

```
<rpc-reply>
  <file-content>
    . . . Trimmed . . .
    Mar  9 18:21:52  vMX-1 jlaunchd: icmd (PID 10257) exited with status=1
    Mar  9 18:21:52  vMX-1 jlaunchd: icmd (PID 12487) started
    Mar  9 18:25:32  vMX-1 mgd[57901]: UI_DBASE_LOGOUT_EVENT: User 'lab' exiting
      configuration mode
  </file-content>
</rpc-reply>
]]>]]>
```

Summary

- In this content, we:
 - Described the YANG Protocol
 - Explained the syntax of YANG

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 33

We Discussed:

- The YANG Protocol;
- The capabilities of YANG;
- How to issue Junos OS RPCs using YANG; and

How to configure the Junos OS using YANG

Review Questions

1. What is the name of the YANG configuration module for Junos OS?
2. Which constraint indicates that a statement is a mandatory part of a YANG RPC?
3. What does the typedef statement do?

Review Questions

- 1.
- 2.
- 3.
- 4.

Answers to Review Questions

1.

The YANG configuration module for the Junos OS is called configuration.yang.

2.

The must constraint indicates that a statement is a mandatory part of a YANG RPC.

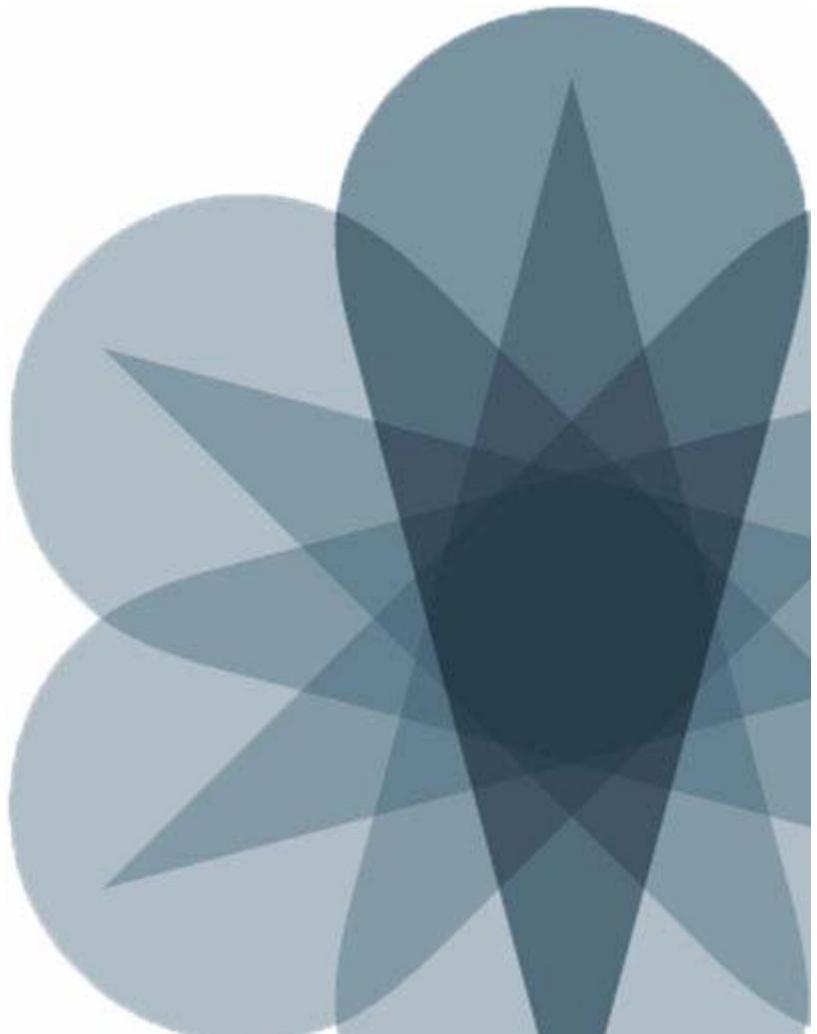
3.

The typedef statement defines a new data type in a YANG module.



Junos Platform Automation and DevOps

Chapter 12: OpenConfig



Objectives

- After successfully completing this content, you will be able to:
 - Describe the advantages of OpenConfig
 - Modify the Junos OS configuration using OpenConfig
 - Describe using OpenConfig with the Junos Telemetry Interface (JTI)

We Will Discuss:

- Advantages of OpenConfig,
- Junos OS configuration using OpenConfig, and
- Using OpenConfig with the Junos Telemetry Interface.

Agenda: OpenConfig

- OpenConfig Overview
- OpenConfig Installation
- Using OpenConfig
- OpenConfig Telemetry

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 3

OpenConfig Overview

The slide lists the topic we will discuss. We discuss the highlighted topic first.

What is OpenConfig?

OpenConfig is an informal working group of network operators sharing the goal of moving our networks toward a more dynamic, programmable infrastructure by adopting software-defined networking principles such as declarative configuration and model-driven management and operations.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 4

What Is OpenConfig?

The slide shows an autobiographical description of OpenConfig. The group of network operators working on the OpenConfig project currently include: Google, AT&T, Microsoft, British Telecom, Facebook, Comcast, Verizon, Level 3, Cox Communications, Yahoo!, Apple, Jive Communications, Deutsche Telekom / Terastream, Bell Canada, SK Telecom, Bloomberg, Netflix, and Cloudflare.

OpenConfig Objectives

- Consistent Vendor-Neutral Data Models
- Declarative Configuration
- Model Only Commonly Deployed Features
- Streaming Telemetry

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 5

Consistent Vendor-Neutral Data Models

One of the key objectives of the OpenConfig initiative is to develop vendor-neutral data models for the configuration and management of network elements. Today there are at least as many different configuration schemas for routing protocols as there are vendors. Even within each vendor there may be multiple mechanisms for configuring identical functionality across different platforms and network operating systems. This diversity significantly increases the cost for operators when developing back-end systems that interact with these network elements.

By defining a set of vendor-neutral data models for commonly used features and protocols, the configuration process can be dramatically simplified across platforms and vendors. With these models in hand, an operator can build offline schemas and serialize configuration data into the preferred format for consumption by a vendor/platform in a consistent manner without having to worry about generating the appropriate vendor-specific syntax to effect a particular configuration.

Declarative Configuration

One of the core tenets of OpenConfig operations is the use of declarative configuration. This enables operators to specify their configuration intent and have the network element determine how to implement that intent. The burden of having to define the specific actions to achieve that objective is removed from the network operator and placed back on to the network element.

From its inception, Junos OS has enabled declarative device configuration. Operators who adopt the OpenConfig models and deploy them on Junos can be assured that they will be able to apply the models in a declarative manner while the Junos translation engine performs the implementation specific actions.

Continued on the next page.

Model Only Commonly Deployed Features

The models and the functionality being defined within the OpenConfig group do not attempt to be all things to everyone. Instead, the intent is to model only commonly deployed features within operator's networks. By definition, this leaves some functionality out-of-scope for OpenConfig models. Due to the flexible nature of the Junos implementation and packaging of OpenConfig support, Juniper Networks (and customers themselves) are able to extend the capabilities of the baseline OpenConfig models and augment these models and the corresponding mapping capabilities to enable the use of Junos-specific features while retaining the vendor-independent baseline models.

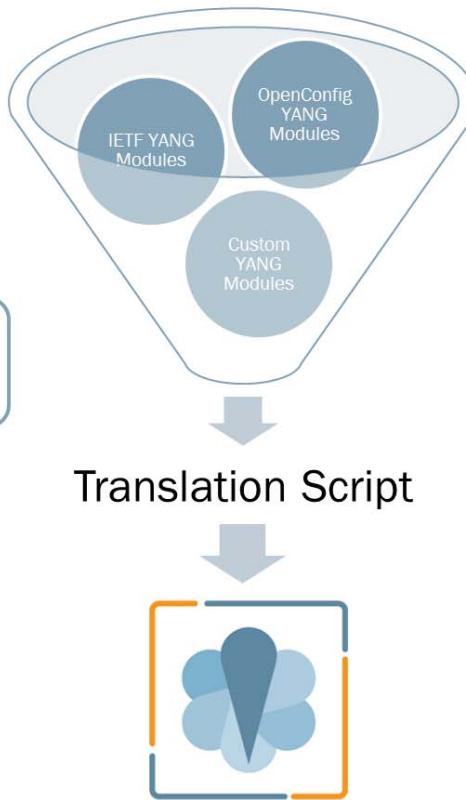
In supporting the OpenConfig models, Junos has taken care to provide clear deviations where there may be differences between the baseline definition (in terms of units, application hierarchy, etc.) and where there may be gaps in coverage. Customers may choose to extend the baseline models themselves, or Juniper Networks may opt to provide augmentations to the baseline models for commonly utilized Junos functionality which isn't in the baseline models in order to streamline deployments for customers.

Streaming Telemetry

Streaming telemetry is a new paradigm for network monitoring in which data is streamed from devices continuously with efficient, incremental updates. Operators can subscribe to the specific data items they need, using OpenConfig data models as the common interface.

Juniper Network's OpenConfig Solution

Bring your own CLI solution!



© 2017 Juniper Networks, Inc. All rights reserved.

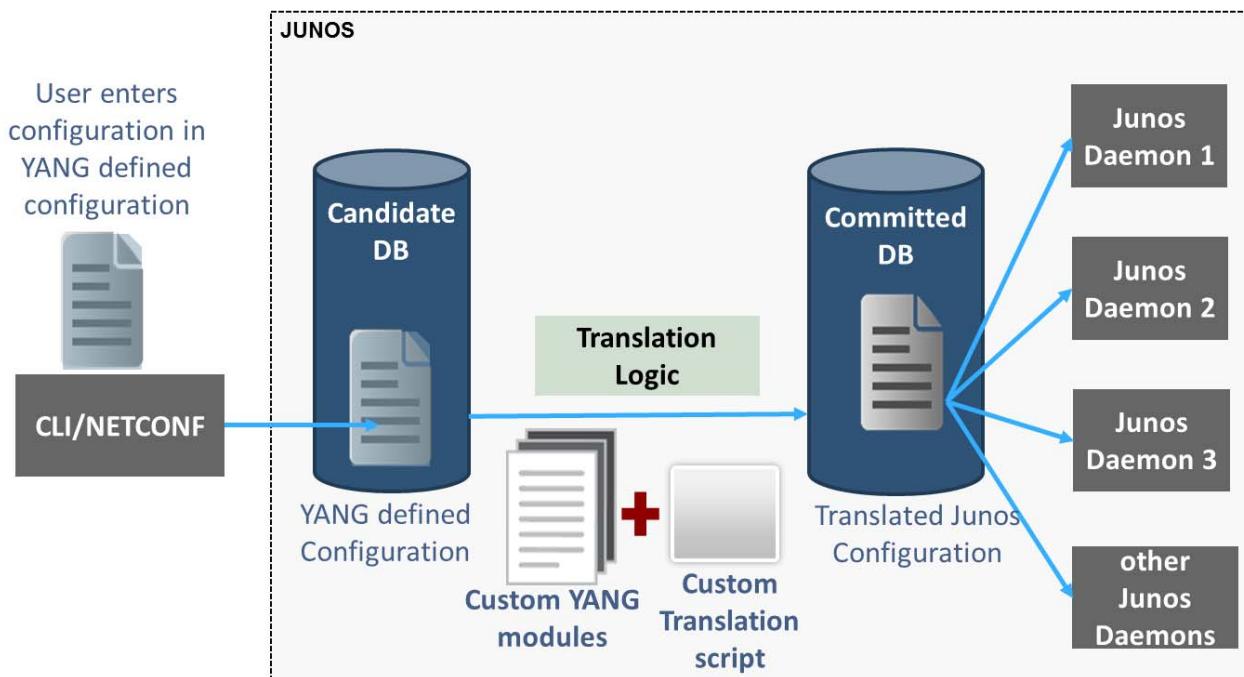
JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 7

Juniper Network's OpenConfig Solution

Some vendors may opt to implement OpenConfig natively into their networks. But because Junos OS is based on a structured data model, it is possible to translate from the OpenConfig data model (or the IETF data model, or an organization's own data model) into the Junos OS data model. The Junos OS solution allows organizations to bring the data model of their choice, then use a translation script to translate it into the documented Junos YANG data model.

The OpenConfig to Junos OS Translation Process



© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 8

The OpenConfig to Junos OS Translation Process

The Junos OS provides support for OpenConfig and IETF standardized YANG models. The Junos OS also provides translation scripts that translate from standardized models to the Junos OS model. However, If organizations have their own model, this can be used as well, as long as the organization provides the necessary translation scripts. Junos OS Technical Documentation provides instructions to organizations on how to accomplish this.

OpenConfig YANG Modules – Support in Junos 16.1R3 and 17.1R1

OpenConfig Data Model	Junos Version	OpenConfig Version
BGP	16.1R3	2.0.1
	17.1R1	2.1.1
Interfaces	16.1R3 and 17.1R1	1.0.2
LACP	16.1R3 and 17.1R1	1.0.2
Local Routing	16.1R3 and 17.1R1	1.0.0
Telemetry	16.1R3 and 17.1R1	0.2.0
RPC	16.1R3 and 17.1R1	0.1.0
LLDP	17.1R1	0.1.0
Platform	16.1R3 and 17.1R1	0.4.0
Routing Policy	16.1R3	2.0.0
	17.1R1	2.0.1
MPLS RSVP	16.1R3 and 17.1R1	1.0.1

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 9

OpenConfig YANG Modules Support in Junos 16.1R3 and 17.1R1

This slide shows a table showing the OpenConfig modules currently supported.

Agenda: OpenConfig

- OpenConfig Overview
- OpenConfig Installation
- Using OpenConfig
- OpenConfig Telemetry

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 10

OpenConfig Installation

The slide highlight the topic we discuss next.

OpenConfig Packages

- OpenConfig is supported on MX and PTX devices
- OpenConfig for Junos OS software packages have the following naming convention:
 - junos-openconfig-XX.YY.ZZ.JJ-signed.tgz (Junos OS)
 - junos-openconfig-x86-32-XX.YY.ZZ.JJ.tgz (Junos OS with Upgraded FreeBSD)
- Where:
 - XX represents the OpenConfig major release number.
 - YY represents the OpenConfig minor release number.
 - ZZ represents the OpenConfig patch release number.
 - JJ represents the Juniper Networks release number.

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 11

OpenConfig Packages

OpenConfig for Junos OS corresponds to OpenConfig YANG data module releases; OpenConfig aids translation scripts and deviation modules for each supported Junos OS release. Each package includes the following files:

OpenConfig set of data models, written in YANG.

Translation scripts that translate OpenConfig configuration schemas to Junos OS configuration schemas for each supported Junos OS release.

Deviation modules that specify the unsupported nodes within the schema for each supported Junos OS release.

Augmentation modules that specify additions to various OpenConfig specified models.

Dynamic rendering files that map operational state data for each supported Junos OS release.

Note: The `junos-openconfig-x86-32-XX.YY.ZZ.JJ.tgz` package supports both 32 and 64 bit systems.

If you have questions about whether your device uses the upgraded FreeBSD, see http://www.juniper.net/documentation/en_US/junos/topics/concept/understanding-junos-kernel-freerbsd10.html.

You can download the Junos OpenConfig packages here: <https://www.juniper.net/support/downloads/?p=openconfig>.

OpenConfig Package Releases

- OpenConfig releases separate from Junos releases
- Install new OpenConfig bundles without upgrade
- Upgrade of OpenConfig after upgrading Junos OS may be needed; see table below for compatibility:

Junos OS Release	OpenConfig Package
17.1R1	0.0.0.2
17.2R1	0.0.0.3
17.1R3	0.0.0.2

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 12

OpenConfig Package Releases

OpenConfig releases are separate from Junos releases, and new OpenConfig bundles can be installed without the need to update Junos. When you do upgrade Junos, you may also need to upgrade your OpenConfig package too. Refer to the table above for corresponding releases.

Installing OpenConfig Packages (1 of 3)

- Install OpenConfig

```
lab@vMX-1> request system software add /var/tmp/junos-
openconfig-x86-32-0.0.0.2.tgz
```

NOTICE: Validating configuration against junos-openconfig-x86-32-0.0.0.2.tgz.

NOTICE: Use the 'no-validate' option to skip this if desired.

Verified junos-openconfig-x86-32-0.0.0.2 signed by
PackageDevelopmentEc_2017 method ECDSA256+SHA256

Adding junos-openconfig-x86-32-0.0.0.2 ...

Initializing...

...Trimmed...

WARNING: cli has been replaced by an updated version:

CLI release 17.1R1.8 built by builder on 2017-02-27 22:02:12
UTC

Restart cli using the new version ? [yes,no] (yes) **yes**

Restarting cli ...

lab@vMX-1>

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 13

Installing OpenConfig Packages, Part 1

Installing an OpenConfig package is similar to installing other Junos OS packages. Issue the `request system software add` command, as seen on the slide. Once the OpenConfig package is installed, you will be prompted to answer if you would like to restart CLI; type yes.

Installing OpenConfig Packages (2 of 3)

- Verify the install and view the installed modules

```
lab@vMX-1> show system yang package
Package ID          : junos-openconfig
YANG Module(s)     : iana-if-type.yang ietf-inet-types.yang
                     ietf-interfaces.yang ietf-yang-types.yang jnx-aug-openconfig-
                     bgp.yang jnx-aug-openconfig-if-ip.yang jnx-aug-openconfig-
                     interfaces.yang jnx-aug-openconfig-lacp.yang jnx-aug-
                     openconfig-lldp.yang
                     ...Trimmed...
Translation Script(s) :openconfig-bgp.slax openconfig-
                     interface.slax openconfig-lldp.slax openconfig-local-
                     routing.slax openconfig-mpls.slax openconfig-policy.slax
Translation script status is enabled
```

Installing OpenConfig Packages, Part 2

To verify the install, issue the show system yang command. Notice that it lists several file types:

- YANG Files
 - Have a .yang extension
 - Standard OpenConfig YANG files
 - organized as distinct stand-alone areas: OpenConfig-BGP lists all the Yang required for BGP
- Deviation Files,
 - Have a .yang extension but begin with jnx-aug
 - Aligns OpenConfig files to Junos device capabilities
 - If an attribute is not supported on Junos, the attribute is declared “Not Supported” in the deviation file; if the data type of the attribute is different on Junos, it is recast in the deviation file.
- Translation Scripts
 - Have a .slax extension
 - Executed at commit time; translates request in OpenConfig namespace to Junos configurations to be processed by Junos Daemons
 - Written in SLAX (Stylesheet Language Alternative Syntax) or Python.

Installing OpenConfig Packages (3 of 3)

- Test Junos OS Context-sensitive Help

```
[edit]
lab@vMX-1# set openconfig-?
Possible completions:
> openconfig-bgp:bgp    Top-level configuration and state for
the BGP router
> openconfig-interfaces:interfaces  Top level container for
interfaces, including configuration
                                         and state data.
> openconfig-lacp:lacp  Configuration and operational state
data for LACP protocol
                                         operation on the aggregate interface
> openconfig-lldp:lldp  Top-level container for LLDP
configuration and state data
>openconfig-local-routing:local-routes  Top-level container
for local routes
...Trimmed...
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 15

Installing OpenConfig Packages, Part 3

Test Junos OS ext-sensitive Help

When you issue the `set openconfig-?` command at the `[edit]` hierarchy, you will see all of the possible OpenConfig completion options; these begin with `openconfig` because `openconfig` is the namespace.

If you attempt issue a `show ietf-?` command, you will see the `ietf-interfaces:interfaces` option. `Ietf` is the namespace for `ietf` modules, and, as of Junos OS version 17.1, the `interface` module is the only `ietf` module currently supported.

Agenda: OpenConfig

- OpenConfig Overview
- OpenConfig Installation
- Using OpenConfig
- OpenConfig Telemetry

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 16

Using OpenConfig

The slide highlights the topic we discuss next.

OpenConfig Case Study (1 of 8)

- Requirements:

- vMX-2 is configured; use OpenConfig to configure BGP on vMX-1



```
AS: 64412
[edit protocols]
lab@vMX-1# show
bgp {
    group ext-peers {
        type external;
        peer-as 65513;
        neighbor 172.17.1.2;
    }
}
```



```
AS: 64413
[edit protocols]
lab@vMX-2# show
bgp {
    group ext-peers {
        type external;
        peer-as 65512;
        neighbor 172.17.1.1;
    }
}
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER Worldwide Education Services

www.juniper.net | 17

OpenConfig Case Study, Part 1

The purpose of this case study is to show you how to use open configurations to configure a device running Junos OS. In this case study, we have two virtual vMX routers. The second vMX router is already configured for you; configure the first router using OpenConfig.

OpenConfig Case Study (2 of 8)

- Install OpenConfig Package
- Issue the following OpenConfig set commands:

```
[edit]
lab@vMX-1# set openconfig-bgp:bgp neighbors neighbor
172.17.1.2 config peer-as 65513

[edit]
lab@vMX-1# set openconfig-bgp:bgp neighbors neighbor
172.17.1.2 config peer-group ext-peers

[edit]
lab@vMX-1# set openconfig-bgp:bgp peer-groups peer-group ext-
peers config local-as 65512

[edit]
lab@vMX-1# set openconfig-bgp:bgp peer-groups peer-group ext-
peers config peer-type EXTERNAL
```

OpenConfig Case Study, Part 2

Install the OpenConfig package. If you need help, refer to the section on installing OpenConfig earlier in this chapter.

After OpenConfig is installed, enter configuration mode. At the [edit] hierarchy, issue the commands listed in the slide to configure BGP using OpenConfig.

If you want to see how the OpenConfig BGP commands map to the Junos OS BGP commands, see>:

http://www.juniper.net/documentation/en_US/junos/topics/reference/general/open-config-bgp-mapping.html.

OpenConfig Case Study (3 of 8)

- Commit the configuration

```
[edit]
lab@vMX-1# commit
commit complete
```

- Test the BGP adjacency

```
[edit]
lab@vMX-1# run show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed      History Damp State      Pending
inet.0
                  0          0          0          0          0          0          0          0
Peer           AS       InPkt     OutPkt   OutQ  Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
172.17.1.2      65513      469      469      0      1    3:29:47
0/0/0/0        0/0/0/0
```

OpenConfig Case Study, Part 3

Commit the configuration, then issue the `run show bgp summary` command to see if the adjacency formed correctly. As seen on the slide, it is correct.

OpenConfig Case Study (4 of 8)

- Show the BGP configuration using the Junos OS model

```
[edit]  
lab@vMX-1# show protocols
```

```
[edit]  
lab@vMX-1#
```

What? No configuration?!?



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 20

OpenConfig Case Study, Part 4

When you issue the `show protocols` command, you may expect to see the standard Junos OS BGP configuration. Because the configuration was created using OpenConfig, the commit is a transient commit, and can only be seen in the OpenConfig namespace.

OpenConfig Case Study (5 of 8)

- Show the BGP configuration using the OpenConfig model

```
[edit]
lab@vMX-1# run show configuration openconfig-bgp:bgp
neighbors {
    neighbor 172.17.1.2 {
        config {
            peer-group ext-peers;
            peer-as 65513;
        }
    }
}
peer-groups {
    peer-group ext-peers {
        config {
            local-as 65512;
            peer-type EXTERNAL;
        } } }
```

OpenConfig Case Study, Part 5

Using the `openconfig` namespace, you can now see the BGP configuration.

OpenConfig Case Study (6 of 8)

- Display only the Junos configuration emitted by the translation scripts

```
[edit]
lab@vMX-1# run show configuration openconfig-bgp:bgp | display
json
{
    "openconfig-bgp:bgp" : {
        "neighbors" : {
            "neighbor" : [
                {
                    "neighbor-address" : "172.17.1.2",
                    "config" : {
                        "peer-group" : "ext-peers",
                        "peer-as" : 65513
                    }
                }
            ]
        }
    }
...
Trimmed...
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 22

OpenConfig Case Study, Part 6

You can view the OpenConfig configuration in text, JSON, or XML. This slide shows the first part of the configuration in JSON.

OpenConfig Case Study (7 of 8)

- Display only the Junos configuration emitted by the translation scripts

```
[edit]
lab@vMX-1# run show configuration | display translation-
scripts translated-config
protocols {
    bgp {
        group ext-peers {
            type external;
            local-as 65512;
            neighbor 172.17.1.2 {
                peer-as 65513;
            }
        }
    }
}
```

OpenConfig Case Study, Part 7

The `run show configuration | display translation-scripts translated-config` command allows you to see the OpenConfig BGP configuration after it has been through the translation script.

OpenConfig Case Study (8 of 8)

- Display the whole configuration in Junos OS data model after translation scripts have been applied

```
lab@vMX-1# run show configuration | display translation-
scripts
## Last commit: 2017-05-19 07:42:11 UTC by lab
version 17.1R1.8;
system {
    host-name vMX-1;
...Trimmed...
}
openconfig-bgp:bgp {
    neighbors {
        neighbor 172.17.1.2 {
            config {
                peer-group ext-peers;
                peer-as 65513;
            }
...Trimmed...
```

The Junos OS configuration
is shown before the OpenConfig
configuration

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 24

OpenConfig Case Study, Part 8

Here you can see the whole configuration. Note that the OpenConfig BGP settings are at the bottom of the configuration code.

```
[edit]
lab@vMX-1# run show configuration | display translation-scripts
## Last commit: 2017-05-19 07:42:11 UTC by lab
version 17.1R1.8;
system {
    host-name vMX-1;
    root-authentication {
        encrypted-password "$5$iBEBRAj5$7cJsubUgaVzauLx1V5W2QzJ2G2TOhsbdIRFOQAgQhW/";
        ## SECRET-DATA
    }
    login {
        user lab {
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "$6$r0gGBtv8$tvl3Tj/PPnTXMPPximLayUCAF6EO1Xib/
MExLKJBL621zIKkBV4hUUIZF0toe6CoaB5DUXjs0SnCLWRBTMH1w."; ## SECRET-DATA

```

Continued on the next page.

OpenConfig Case Study, Part 8 (contd.)

```

        }
    }

services {
    ssh;
    telnet;
    netconf {
        ssh;
    }
}
syslog {
    file messages {
        any notice;
        authorization info;
    }
    file interactive-commands {
        interactive-commands any;
    }
}
ntp {
    boot-server 10.210.8.72;
    server 10.210.8.72;
}
}
chassis {
    fpc 0 {
        lite-mode;
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 172.17.1.1/24;
            }
        }
    }
    fxp0 {
        description "MGMT ADDRESS - DO NOT DELETE";
        unit 0 {
            family inet {
                address 172.25.11.1/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 192.168.1.1/32;
            }
        }
    }
}
}

```

Continued on the next page.

OpenConfig Case Study, Part 8 (contd.)

```
routing-options {
    static {
        route 0.0.0.0/0 next-hop 172.25.11.254;
    }
    autonomous-system 65512;
}

protocols {
    bgp {
        group ext-peers {
            type external;
            local-as 65512;
            neighbor 172.17.1.2 {
                peer-as 65513;
            }
        }
    }
}

openconfig-bgp:bgp {
    neighbors {
        neighbor 172.17.1.2 {
            config {
                peer-group ext-peers;
                peer-as 65513;
            }
        }
    }
    peer-groups {
        peer-group ext-peers {
            config {
                local-as 65512;
                peer-type EXTERNAL;
            }
        }
    }
}

[edit]
lab@vMX-1#
```

Custom YANG Modules

- Translation scripts

- SLAX or Python scripts map custom configuration syntax defined by YANG

Junos OS syntax

Translated data

- Translated data loaded into a configuration as a transient change during the commit operation; Junos OS invokes script to perform translation and emit transient change

- Action scripts

- SLAX or Python scripts that act as handlers for your custom YANG RPCs

Custom YANG Modules

When you add YANG data models that are not natively supported by devices running Junos OS, you must also supply a script to handle the translation logic between the YANG data model and Junos OS for that device. There are two types of scripts:

Translation scripts are Stylesheet Language Alternative SyntaX (SLAX) or Python scripts that map the custom configuration syntax defined by the YANG model to Junos OS syntax, then load the translated data into the configuration as a transient change during the commit operation. When you load and commit configuration data in the non-native hierarchies on those devices, Junos OS invokes the script to perform the translation and emit the transient change.

Action scripts are SLAX or Python scripts that act as handlers for your custom YANG RPCs. The YANG RPC definition uses a Junos OS YANG extension to reference the appropriate action script, which is invoked when you execute the RPC.

To use custom YANG data models on devices running Junos OS, you must add the YANG modules and associated scripts to the device using the `request system yang add` command. Junos OS validates the syntax of the modules and scripts, rebuilds its schema to include the new data models, and then validates the active configuration against this schema.

Although the device validates the modules and scripts as you add them, we recommend that you validate the syntax prior to merging them with the Junos OS schema by first executing the `request system yang validate` command.

Note: In multichassis systems, you must download and add the modules and scripts to each node in the system.

Note: To install OpenConfig modules that are packaged as a compressed tar file, use the `request system software add` command.

Continued on the next page.

Custom YANG Modules (contd.)

When you add YANG modules and scripts to devices running Junos OS, you must associate them with a package. Packages have a unique identifier and represent a collection of related modules, translation scripts, and action scripts. You reference the package identifier if you later update modules and scripts in that package, enable or disable translation scripts associated with the package, or delete that group of modules and scripts from the device.

When you add, update, or remove YANG modules and scripts on the device using the appropriate operational commands, you do not need to reboot the device in order for the changes to take effect. Any installed modules and scripts persist across reboots as well. Newly added RPCs and configuration hierarchies are immediately available for use, and installed translation scripts are enabled by default.

You can disable translation scripts in a package at any time without removing the package and associated files from the device, which can be useful when troubleshooting translation issues. When you disable translation for a package, you can configure and commit the statements and hierarchies added by the YANG modules in that package, but the device does not translate and commit the corresponding Junos OS configuration as a transient configuration change during the commit operation.

For more information on custom YANG scripts, see: https://www.juniper.net/documentation/en_US/junos/topics/concept/netconf-yang-modules-custom-managing-overview.html.

Agenda: OpenConfig

- OpenConfig Overview
 - OpenConfig Installation
 - Using OpenConfig
- OpenConfig Telemetry

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 29

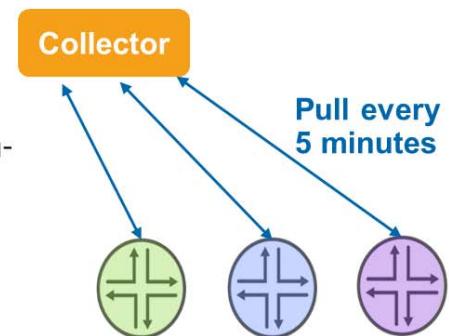
OpenConfig Telemetry

The slide highlights the topic we discuss next.

A Peek at the Past: SNMP

■ SNMP Disadvantages

- SNMP created for platforms with decade-old tech
- Non-scalable platforms with cumbersome op states, control planes, and interfaces
- Centralized approach prone to bottlenecks
- Low visibility network and infrastructure in a high-visibility world
- Challenging multi-vendor SNMP implementation
- Time-consuming bug fixes, qualification, and deployment cycles
- Uses a dated “pull” model to retrieve network information



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

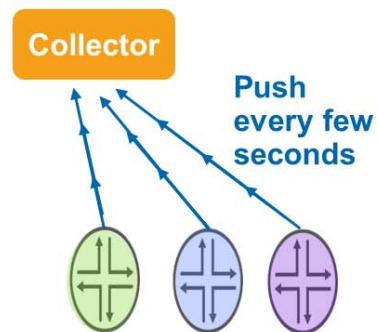
www.juniper.net | 30

SNMP Disadvantages

SNMP has a number of disadvantages, as shown in the slide. These disadvantages have given rise to a need for telemetry solutions.

Streaming Telemetry Advantages

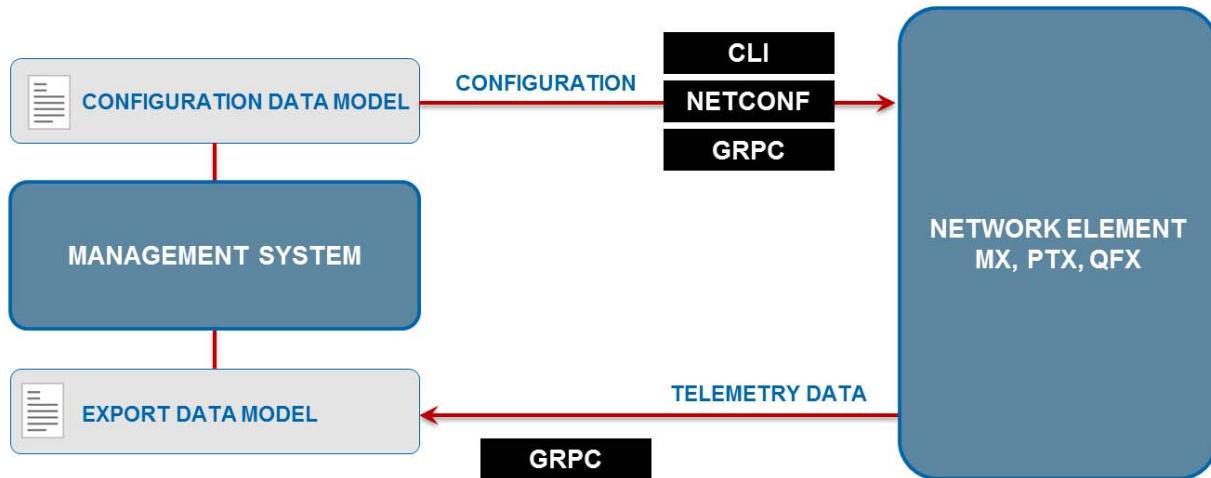
- Utilizes a push model: router streams data to the collector (like video streaming)
- Distributed approach: every module responsible to stream its own data; high frequency streaming with fewer CPU cycles
- Streaming starts at source; efficient and granular
- Scalable, operates and secures well for next-gen networking platforms
- With phase2 implementation, ST has potential to replace legacy monitoring infrastructure (CLI scrapers, Netconf, SNMP)
- ST plays well in heterogeneous networks with OpenConfig support
- Positioned to replace SNMP



Streaming Telemetry

Telemetry may not solve all the shortcomings of SNMP, but it offers numerous benefits, as seen in the slide.

Juniper Telemetry Ecosystem



© 2017 Juniper Networks, Inc. All rights reserved.

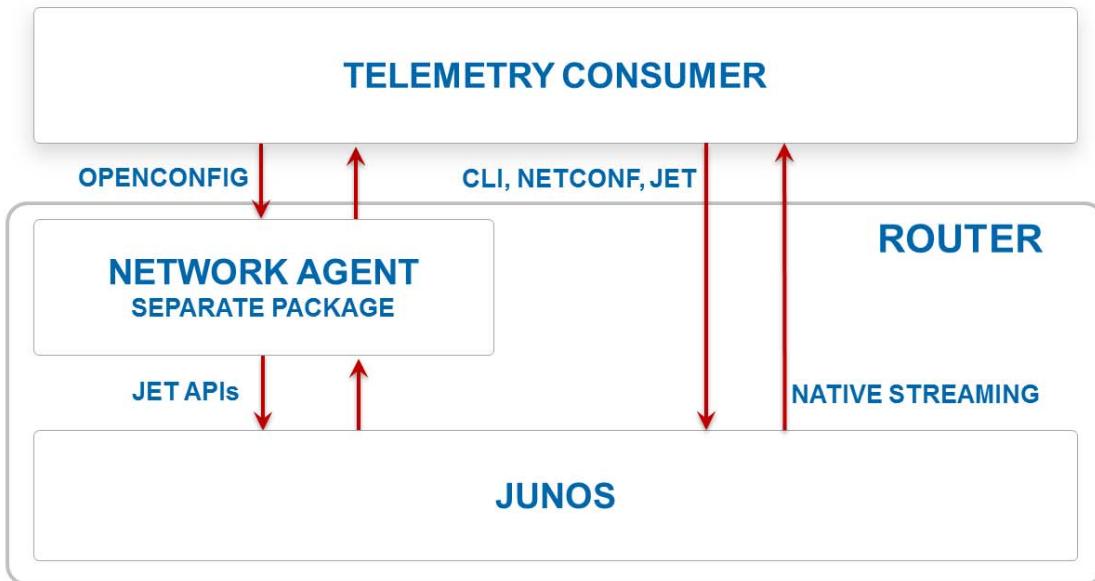
JUNIPER Worldwide Education Services

www.juniper.net | 32

Juniper Telemetry Ecosystem

The Juniper Telemetry Ecosystem allows for closed-loop automation. Closed-loop automation is depicted in the slide, where you push configuration to the device, and, based on the received ongoing telemetry, the system can modify the configuration as needed. The eventual goal is to move toward self-healing systems.

System Architecture, Detailed View



© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 33

System Architecture

You can see how the network agent both sends configuration information to the Junos device, and receives telemetry information back from the Junos OS processes.

Using OpenConfig on the Junos Telemetry Interface

- Beginning with Junos OS Release 16.1R3, you can use a set of remote procedure call (RPC) interfaces to configure the Junos Telemetry Interface (JTI) and stream telemetry data using the gRPC
- Need to install network agent on devices with upgraded FreeBSD
- Need to install Junos OpenConfig package on device
- Only gRPC is supported to stream data

Using OpenConfig on the Junos Telemetry Interface

Starting in Junos OS Release 16.1R3, you can use a set of remote procedure call (RPC) interfaces to configure the Junos Telemetry Interface and stream telemetry data using the gRPC framework. OpenConfig supports the use of vendor-neutral data models for configuring and managing multivendor networks. gRPC is an open source framework that provides secure and reliable transport of data.

Implementing OpenConfig with gRPC for Junos Telemetry Interface requires that you download and install a package called Network Agent if your Juniper Networks device is running a version of Junos OS with Upgraded FreeBSD. For all other versions of Junos OS, the Network Agent functionality is embedded in the software. Network Agent functions as a gRPC server and terminates the OpenConfig RPC interfaces. It is also responsible for streaming the telemetry data according to the OpenConfig specification.

OpenConfig for Junos OS specifies an RPC model to enable the Junos Telemetry Interface. You must download and install the OpenConfig for Junos OS package and YANG Models on your Juniper Networks device.

Per the OpenConfig specification, only gRPC-based transport is supported for streaming data. The gRPC server that is installed by the Network Agent package terminates the gRPC sessions from the management system that runs the client. RPC calls trigger the creation of Junos OS sensors that either stream data periodically or report events, which are then funneled onto the appropriate gRPC channel by Network Agent.

Note: OpenConfig for Junos OS and gRPC is supported only on MPCs on MX Series and on PTX Series routers.

Junos Telemetry Interface sensors are not supported on MX80 and MX104 routers.

For more information on the Junos Telemetry Interface, OpenConfig and gRPC see http://www.juniper.net/documentation/en_US/junos/topics/concept/junos-telemetry-interface-overview.html.

References

- OpenConfig

- OpenConfig Feature Guide

- https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/open-config/open-config-feature-guide.html

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 35

References

The slide lists useful references for more information on NETCONF, YANG, and OpenConfig.

Summary

- In this content, we:
 - Described the advantages of OpenConfig
 - Modified the Junos OS configuration using OpenConfig
 - Described the use of OpenConfig with the Junos Telemetry Interface

We Discussed:

- Advantages of OpenConfig,
- Junos OS configuration using OpenConfig, and
- Using OpenConfig with the Junos Telemetry Interface.

Review Questions

1. What is the purpose of the OpenConfig Working Group?
2. Why doesn't the translated OpenConfig configuration show up in the Junos OS candidate configuration?
3. Can the OpenConfig telemetry interface use any other protocol than gRPC?

Review Questions

- 1.
- 2.
- 3.

Lab: Implementing OpenConfig

- Modify the Junos Configuration using OpenConfig.

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 38

Lab: Implementing OpenConfig

The slide lists the objective for this lab.

Answers to Review Questions

1.

The purpose of the OpenConfig working group is to work toward the goal of “moving our networks toward a more dynamic, programmable infrastructure.”

2.

The OpenConfig configuration doesn’t show up in the Junos OS candidate configuration because it is committed as part of the transient commit.

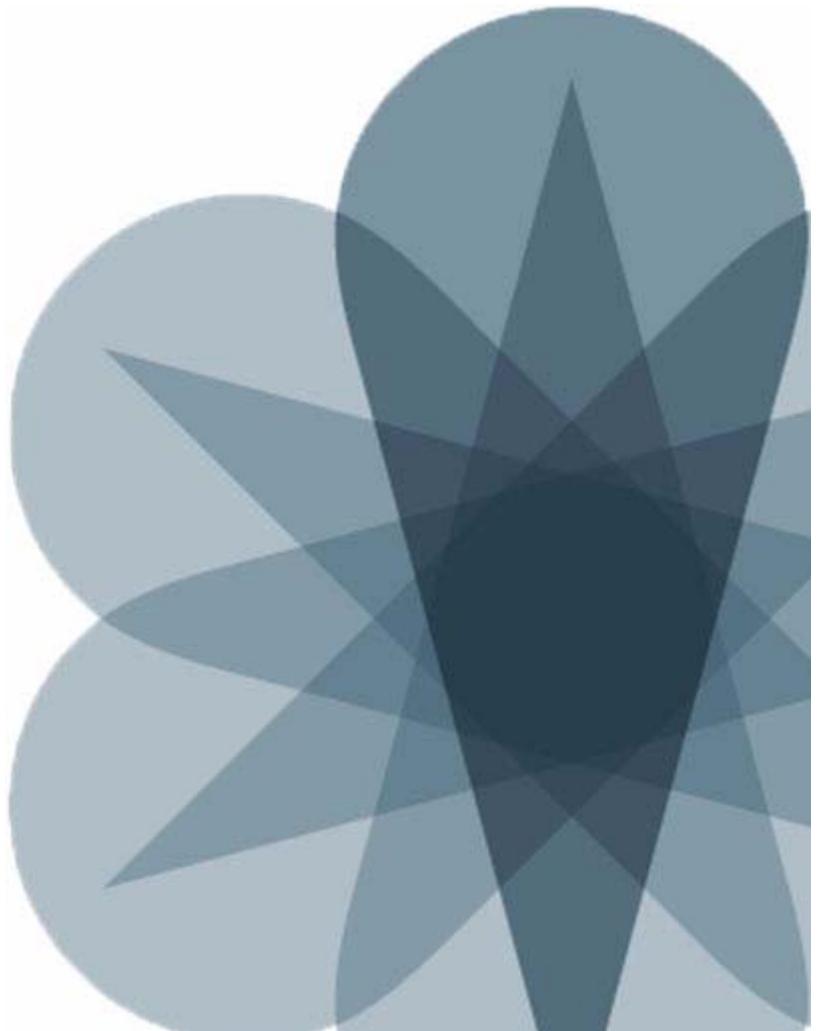
3.

The OpenConfig telemetry interface has to use the gRPC transport protocol.



Junos Platform Automation and DevOps

Chapter 13: Junos Extension Toolkit



Objectives

- After successfully completing this content, you will be able to:
 - Setup a JET VM
 - Create JET Packages
 - Use the JET API

We Will Discuss:

- Setup of a JET VM,
- Creation of JET Packages, and
- Usage of the JET API.

Agenda: Junos Extension Toolkit

- Overview of JET
- Creating Signed JET Apps
- Creating Unsigned JET Apps
- Creating JET Notification Apps

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 3

Overview of JET

The slide lists the topics we will discuss. We discuss the highlighted topic first.

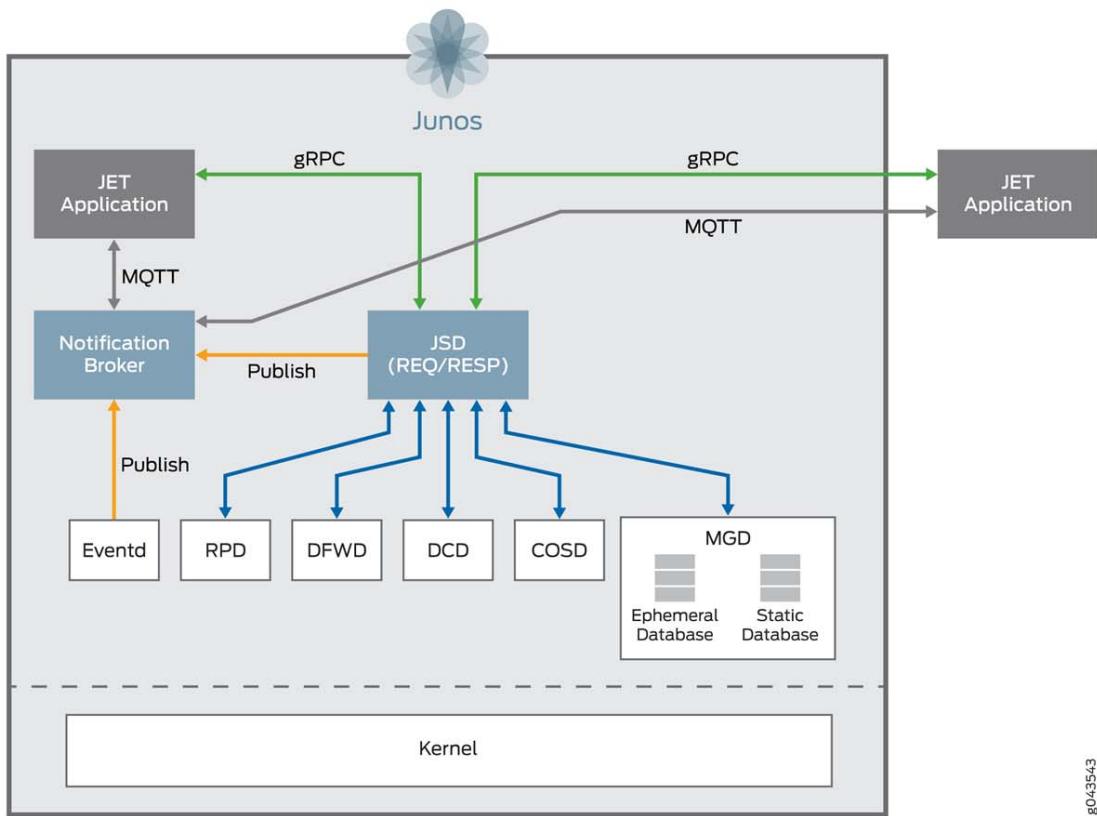
Junos Extension Toolkit (JET)

- Multiple languages for off-box development
- Write applications in Python for on-box development
- Applications written in C to run on devices that do not use the JET APIs
- An updated standards based event notification method that enables the applications to respond to selected system events

Junos Extension Toolkit (JET)

Juniper Extension Toolkit (JET), an evolution of the Junos SDK, provides a modern, programmatic interface for developers of third-party applications. It focuses on providing a standards-based interface to the Juniper Networks Junos OS for management and control plane functionality.

JET Interaction with the Junos OS



804543

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER NETWORKS Worldwide Education Services

www.juniper.net | 5

JET Interaction with the Junos OS

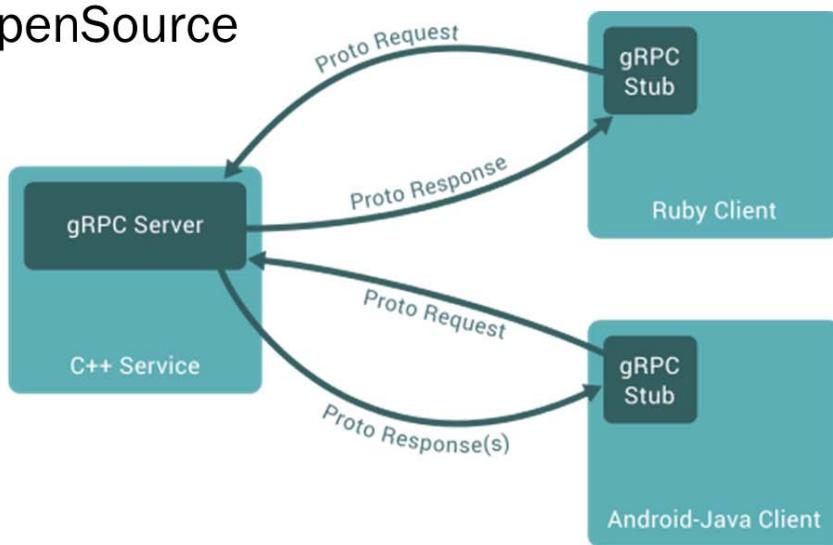
The Junos Extension Toolkit (JET) uses gRPC, a remote procedure call (RPC) framework, for cross-language services (see <http://www.grpc.io/>) as a mechanism to enable request-response services. The gRPC protocol provides an interface definition language (IDL) that enables you to define APIs. These IDL files, with a .proto file extension, are compiled using the protoc compiler to generate source code used for server and client applications. The gRPC server is part of the JET service process (jsd), which runs on Junos OS.

For event notification, JET uses the Message Queue Telemetry Transport (MQTT) protocol (see <http://mqtt.org/>). Event notification is implemented through the mosquitto notification broker (see <http://mosquitto.org/>).

Notice that the image above shows that the same protocols, gRPC, and MQTT, are used for both on-box and off-box communication.

Why gRPC

- Uses Protocol Buffers for efficient binary RPC Serialization
- Works across many languages and platforms
- OpenSource



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 6

Why gRPC?

gRPC is a high performance open source RPC framework and protocol developed by Google. gRPC uses the Protocol Buffers as a means to efficiently send RPC requests and responses.

The gRPC protocol can be used across many different platforms and the RPC requests and responses can be created in the following languages: C++, Java, Python, Go, Ruby, Node.js, C#, Android Java, Objective-C, and PHP.

More information about gRPC and Protocol Buffers can be found at: <http://www.grpc.io/>.

Why MQTT?

- Lightweight messaging protocol for publish/subscribe messages
- Designed for device automation



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 7

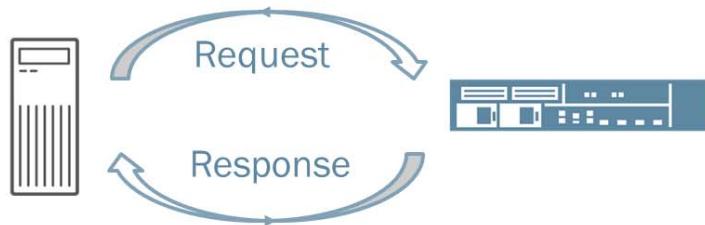
Why MQTT?

MQTT (MQ telemetry transport) is a protocol design for carrying telemetry data. MQTT is a lightweight protocol designed to use few device resources and low bandwidth. MQTT is an ISO standard (ISO/IEC PRF 20922). MQTT is implemented in the JET framework to carry telemetry data. To find out more about MQTT, see: www.mqtt.org.

JET API Overview

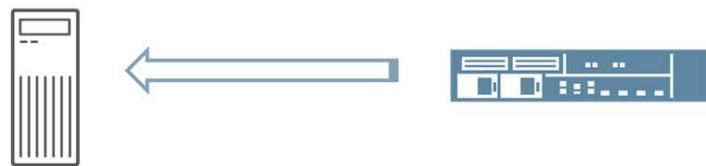
▪ JET Service APIs

Provides interfaces to access the control plane on the device and a management interface to run operational and configuration commands.



▪ JET Notification API

Provides interfaces that allow you to subscribe to events and designate a callback function to receive events when they occur.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 8

JET API Overview

The JET API includes two separate APIs; the JET Notification API and the JET Service API. Both will be discussed. The JET Notification API allows you to subscribe to events and designate a callback function to receive events when they occur.

The JET Service APIs provide access to the control plane on the device and a management interface to run operational and configuration commands. The JET Service APIs currently include the following:

- *BGP Service API*: The BGP Service API allows you to view, add, remove, modify, and monitor BGP static routes.
- *Class of Service (CoS) API*: The CoS Service API allows you to manage many aspects of the CoS features of Junos OS. Through this API, you can manage and control the following CoS related objects: Classifiers, Drop profiles, Forwarding classes, Node features, Overhead accounting, Resource limits, Rewrite features, Schedulers and scheduler maps, Traffic control profiles, and Firewall Service API.
- *Firewall Service API*: Provides the following firewall and traffic policer interfaces:
 - Add, change, delete, or replace a firewall filter.
 - Bind or unbind a firewall filter to an interface and direction.
 - Add, change, or delete a single-rate two-color policer.
 - Get statistics associated with a firewall filter counter or policer.
 - Clear statistics associated with a firewall filter counter.

Continued on the next page.

JET API Overview (contd.)

Each of the previous operations results in data that is returned from the device as confirmation. Return data can indicate the success or failure of the operation, reason for failure, an output regarding the requested configuration in multiple formats including JSON, XML, and text.

- *Interfaces Service API:* Allows you to perform the following interface-related operations on a Junos OS device:
 - Add, delete, or modify physical and logical interfaces
 - Add, delete, or modify aggregate Ethernet (AE) member interfaces
 - Query interface attributes
 - Query interface ownership
 - Query a logical interface to see if it is public (created by CLI)
 - Set interface attributes
- The requests that configure the interface objects are targeted at specific interface hierarchy levels, interface address for example. Because of this, the parent objects, logical and physical interface attributes are not affected by commands targeted at lower levels. When creating objects at lower levels, the parent object attributes can be specified explicitly or left to default values by not specifying them.
- *Management Service API:* Allows you to perform the following system management operations on a Junos OS device:
 - Commit configuration changes
 - Edit ephemeral database configuration
 - Execute operational mode commands
 - Execute configuration mode commands
 - Get configuration from ephemeral database
 - Log comments on the device regarding commit operations
- *MPLS Service API:* Allows you to gather and view information about LDP, RSVP, and VPN type label-switched paths (LSPs) including MPLS forwarding information.

The JET Service Process

- JET Service Process listens on port #32767
- Each request creates separate execution thread
- Sessions remain open as long as communication is possible.
- A single session can execute many APIs
- APIs can execute in parallel
- Maximum of 8 active sessions to single device.

The JET Service Process

To support application interaction with Junos OS, the JET service process (jsd), by default, uses TCP port 32767 to listen for and receive requests from applications to execute APIs. Whenever a request comes on the TCP port, jsd creates a separate thread to service the JET application request. The session remains established as long as the client and server are both up and able to communicate with each other. Over the lifetime of a session, jsd can execute many APIs, and it can execute APIs from multiple sessions in parallel. You can have a maximum of 8 active client sessions connected at any given time.

Agenda: Junos Extension Toolkit (JET)

- Overview of JET
- Creating Signed JET Apps
- Creating Unsigned JET Apps
- Creating JET Notification Apps

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 11

Creating Signed JET Applications

The slide highlights the topic we discuss next.

Developing a Signed JET Application

- Setting Up the JET Virtual Machine
- Developing an Application Using the JET IDE
- Requesting the Certificate Using the JET IDE
- Building and Creating a Package by Using the JET IDE
- Deploying an Application Package on a Device
Running Junos OS

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

Developing a Signed JET Application

If an application that you want to develop has a dependency on C or C++ modules or the application needs to be signed, then you must use the Juniper Extension Toolkit (JET) virtual machine (VM) for application development.

The JET VM is a 64-bit Ubuntu long-term support release. Application developers can use the JET IDE provided with the VM to develop applications. To set up the development environment, you must download the JET bundle from the Juniper Networks download site, and then, once the VM is up, install the JET toolchain, Eclipse integrated development environment (IDE), plug-ins, and other tools and libraries that are required for developing on-device or off-box applications.

The next several slides walk you through the process of developing a signed JET application. The list at the bottom of the slide will help you keep track of where we are in the process.

Setting Up the JET Virtual Machine

- Download Third Party Software
 - Download Vagrant
 - Download VirtualBox
- Download JET Vagrant setup file
- Launch the JET VM
- Download files to JET VM
 - Download JET IDL Client Library
 - Download JET Software Bundle
- Configure eclipse
- Stop the JET VM

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 13

Setting Up the JET Virtual Machine

The first step to setting up a JET virtual machine is to download the required third-party software. You will need to download the latest versions of Vagrant and Oracle VirtualBox.

Vagrant (<https://www.vagrantup.com/>) is a software application that creates and configures virtual development environments. You can think of it as a higher-level wrapper around virtualization software such as VirtualBox (<https://www.virtualbox.org/wiki/Downloads>). You can use Vagrant to manage the JET development virtual machine (VM). To download Vagrant, go to <https://www.vagrantup.com/> and download Vagrant for your system's platform (Windows, Mac, or Linux).

VirtualBox is an open source virtualization application and is currently the only virtualization application supported by the Juniper JET VM. You can download VirtualBox by going to <https://www.virtualbox.org/wiki/Downloads>. Download and install the VirtualBox package for your platform and the VirtualBox extension package. Also be sure to enable hardware virtualization support on your machine BIOS if it is not already enabled.

After downloading and configuring Vagrant and Oracle VirtualBox, you next need to download the JET-vagrant.zip file. This file instructs the Vagrant application on how to download and configure an Ubuntu virtual machine that contains the eclipse IDE and the necessary plug-ins for building a signed JET application. The JET-vagrant.zip file is downloaded from the Juniper Networks download website at <http://www.juniper.net/support/downloads/?p=jet#sw>

Continued on the next page.

Setting Up the JET Virtual Machine (contd.)

After downloading the JET-vagrant.zip file, create a jet-vagrant directory and extract the JET-vagrant.zip file you downloaded from the Juniper Networks download site to that directory. Next, using a command line interface, change to the jet-vagrant directory where you have extracted the JET-vagrant.zip file. Finally, issue the `vagrant up` command. If needed, use the following default login credentials (username: `vagrant`, password: `vagrant`)

Wait for the Ubuntu desktop screen to come up in Oracle VirtualBox VM and you see the Eclipse icon. This is the icon you use for the JET IDE. Depending upon your download speeds, processor, and RAM, this process can take 15 to 60 minutes or more.

After the JET VM is up and running you need to download the gRPC IDL library and the Juniper Software bundle. Because this course is based on Junos 17.1 the current files at this time are:

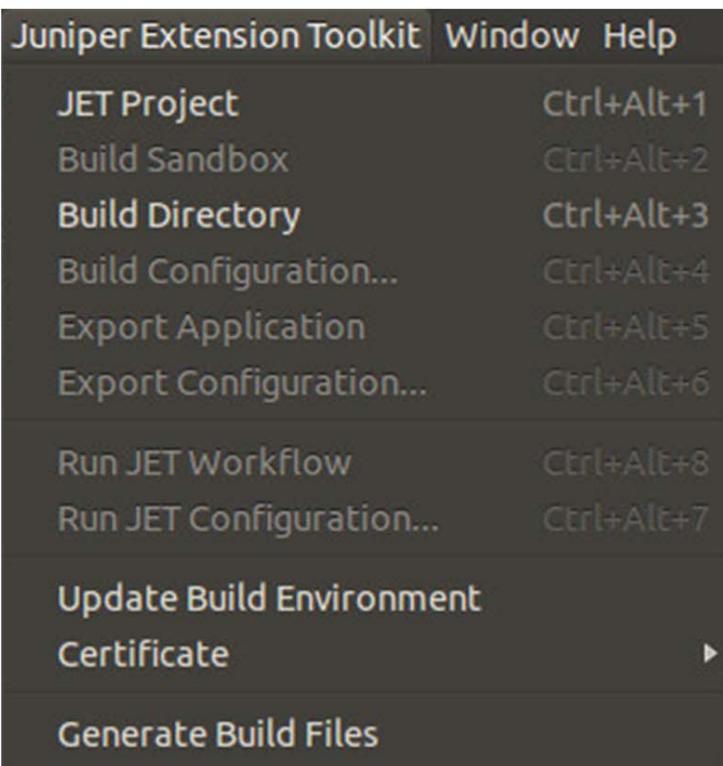
- JET IDL Client Library, filename - `jet-idl-17.1R1.8.tar.gz`
- JET Software Bundle, filename - `junos-jet-17.1R1.8.tar.gz`

These files can be downloaded from the Juniper Networks download website at <http://www.juniper.net/support/downloads/?p=jet#sw>

Once the necessary files are downloaded to the JET VM, the JET software Bundle needs to be integrated into eclipse. To configure eclipse double-click the Eclipse icon on the desktop to start the JET IDE. Next, change the eclipse perspective by selecting Juniper Extension Toolkit from the Window > Open perspective > Other menu. Once the perspective is changed, select Juniper Extension Toolkit > Update build environment.menu item. Next, fill out the Jet Bundle Package input field to download the JET bundle and click Finish to install it. In the same dialogue box as the Jet Bundle Package input field there is a JET Python Client Package field. Starting in Junos OS Release 16.2R1, there is no need to install a new Jet Python Client Package. When using Jet Bundle Package release 16.2R1 or later, the field can be ignored.

Note: When you are done using the VM and want to stop and exit Vagrant, go to the command-line prompt for your system and in the jet-vagrant directory, issue the `vagrant halt` command.

Juniper Extension Toolkit Menu



■ The JET Extension Toolkit menu is where you can access JET specific commands



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER NETWORKS Worldwide Education Services

www.juniper.net | 15

The Juniper Extension Toolkit Menu

When developing a JET application most of the commands specific to a JET application are in the Juniper Extension toolkit menu. Below is a description of each of the Juniper Extension Toolkit menu options:

- **JET Project:** Create a new project and sandbox for developing applications or select an existing project.
- **Build Sandbox:** Launch a build of the application based on the configuration that was provided in Build Configuration.
- **Build Directory:** Compile and build the application by running the make command in the directory that was selected in the Sandbox Explorer window.
- **Build Configuration:** Configure your build settings and launch a build for the current sandbox. In the Build Configuration window, you can configure the following settings:
 - Select the packages that will be included in the build.
 - Select the targeted architecture.
 - Select the packaging type (operating system).
- **Export Application:** Launch and export a build based on the configuration you provide in Export Configuration.

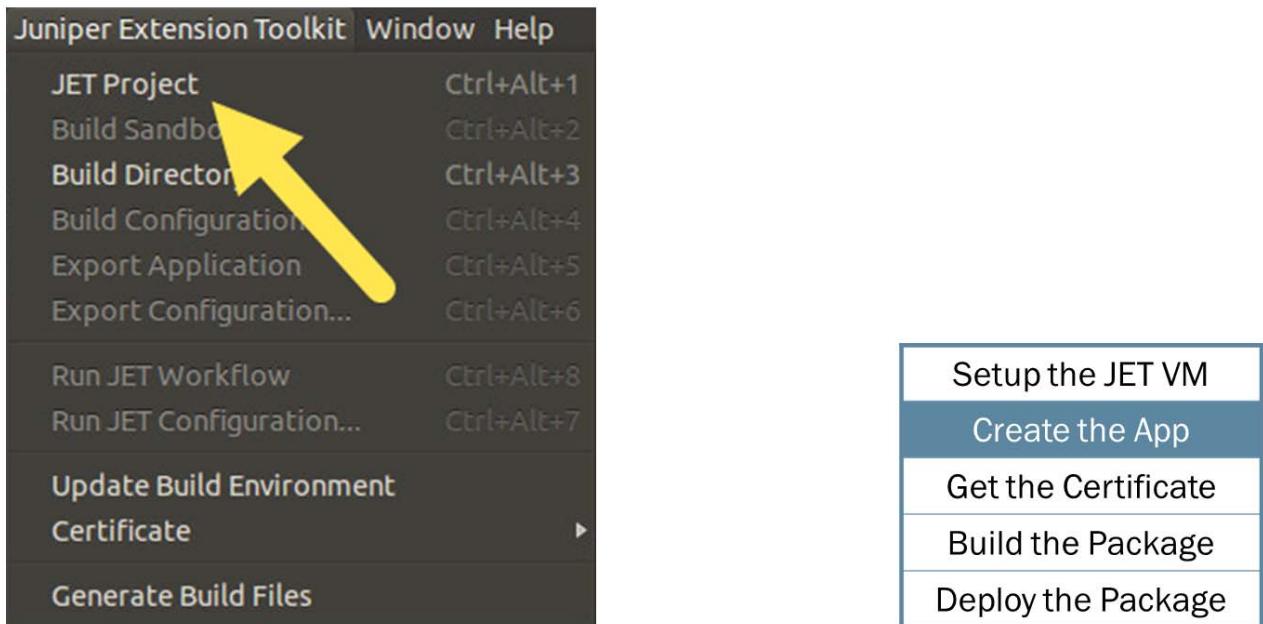
Continued on the next page.

The Juniper Extension Toolkit Menu (contd.)

- *Export Configuration:* Configure your build and export settings, build your application, and export your application. In the Export Configuration window, you can configure the following settings:
 - Select the directory location for the application once it has been built.
 - Select the packages to include in the build.
 - Select the targeted architecture.
 - Select the packaging type (operating system).
- *Run JET Workflow:* Build, install, and test the application based on the settings configured in Run JET Configuration.
- *Run JET Configuration:* Configure your build settings, configure the router setting, launch a build, and execute operational commands needed to install and test the application. In the JET Configuration window, you can configure the following:
 - In the Export Configuration tab, select the packages to include in the build, select the targeted architecture, and select packaging type.
 - In the Device Details tab, specify the router on which you want to test your application by entering in the router name, login ID, password, and pathname where you want to install the application.
 - In the Operational Commands tab, enter operational commands to run on the router prior to and after installing the application.
 - In the Device Configuration window, enter a configuration in curly bracket ({}) format to run on the router prior to and after installing the application.
- *Update Build Environment:* Update your JET development environment by installing JET bundle packages that you have downloaded.
- *Certificate:* Install a certificate, request the signing of a certificate, and generate a certificate request. See Requesting the Certificate Using the JET IDE.
- *Generate Build Files:* Generate makefiles based on input entered in a JSON file. For more information on JSON files, see JSON File Format for JET Application Packaging.

Developing an Application Using the JET IDE

- Start the JET IDE and select Juniper Extension Toolkit > JET Project



© 2017 Juniper Networks, Inc. All rights reserved.

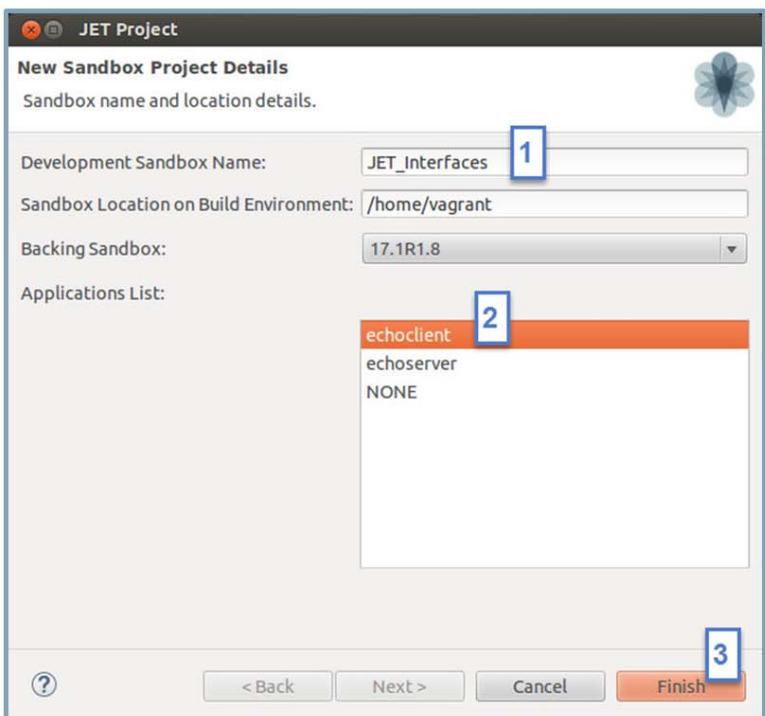
JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 17

Start the JET IDE and Open a Create a New Project

Now that the JET VM is setup and configured, lets walk through the process of creating and application. Application developers can use the integrated development environment (IDE) provided with the Juniper Extension Toolkit (JET) virtual machine (VM) to develop applications. To develop an application by using the JET IDE start the JET IDE and select Juniper Extension Toolkit > JET Project.

Name the Project



- Type in a project name,
- Select an item from the Applications List, and
- Click Finish



© 2017 Juniper Networks, Inc. All rights reserved.

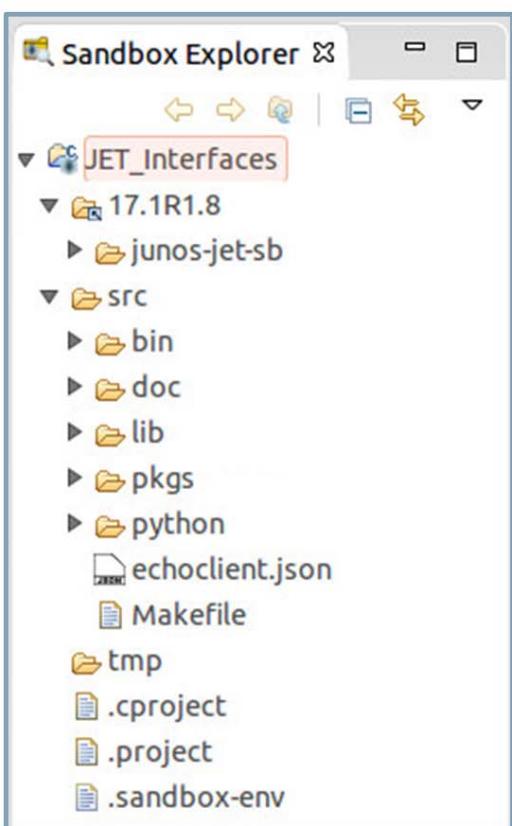
JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 18

Name the Project

To develop an application by using the JET IDE start the JET IDE and select Juniper Extension Toolkit > JET Project. Once selected you will be prompted with a dialogue box where you can type in a project name and select an item from the Applications List. If this is a new application, select none form the application list and click Finish.

Expand the Project Folder Tree



- Expand the new project name and see several directories, including the `src` directory.
- Expand the `src` directory to see several more directories, among them the `python`, `lib`, and `bin` directories.

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

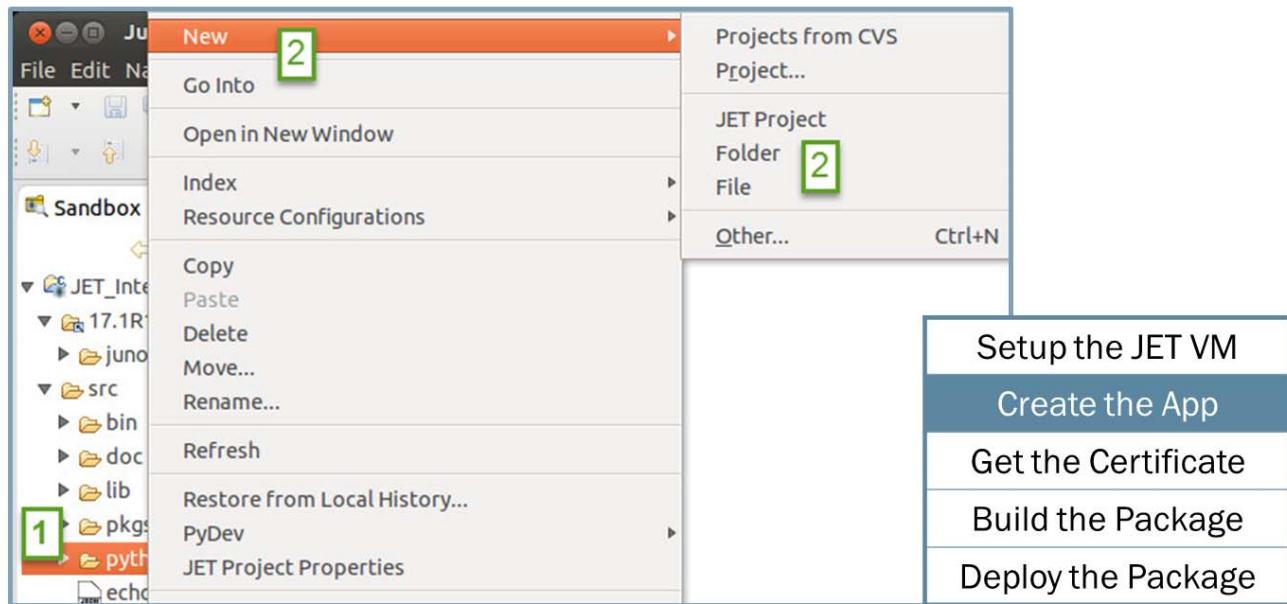
www.juniper.net | 19

Expand the Project Tree

Once you close the new project dialogue box, the new project name appears in the Sandbox Explorer window (top left pane). You can expand the new project name and see several directories, including the `src` directory. Expand the `src` directory to see several more directories, among them the `python`, `lib`, and `bin` directories. These are the directories you use to create new application components.

Create a Folder For Your Code

- To develop applications in Python, create a new application directory in the python directory and develop the application inside the newly created directory



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 20

Create a Folder For Your Code

Next, highlight the directory for your applications files (choose from among the `python`, `lib`, and `bin` directories). If your project is a Python project choose `python`. To select a file, either select the folder with your mouse and navigate to the `File > New > Folder` menu or right-click the desired folder and select `New > Folder`, type a folder name, and click `Finish`.

To develop applications in Python, create a new application directory in the `python` directory and develop the application inside the newly created directory. To create a C/C++ library, create a new library directory in the `lib` directory and develop the library inside the newly created directory. To create a C/C++ executable, create a new application directory in the `bin` directory and develop the application inside the newly created directory. For example, for a C/C++ application, you can put some libraries in the `lib` directory and other application components in the `bin` directory.

Compile the gRPC IDL Files

- Download the JET Client IDL Library
<https://www.juniper.net/support/downloads/?p=jet#sw>
- Unzip compress the files
- Compile .proto files using the protoc compiler

Setup the JET VM
 Create the App
 Get the Certificate
 Build the Package
 Deploy the Package

Generate the gRPC IDL Files

The next step is to generate and import gRPC client library code. Starting in Junos OS Release 16.2R1, you must generate and import the client library code rather than downloading pre-packaged client libraries. This process is done from the command line. Using the IDL files from the client IDL library for the services you are working in, generate client library code following the instructions at <http://www.grpc.io/docs>. Import the generated code into your sandbox using either the Import option from the File menu, or by copying the generated files and pasting them into your sandbox. You will get to see how to generate the IDL files step-by-step latter on in the chapter.

Write the Application

- Create the code
 - Create a new file by selecting subdirectory, right-click select > New > File
- Save the file. Select File > Save or press Ctrl+s
- Makefiles and the manifest file are created automatically once you have created the JSON file.

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

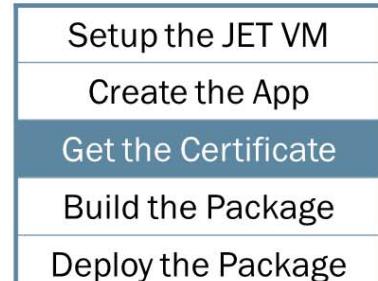
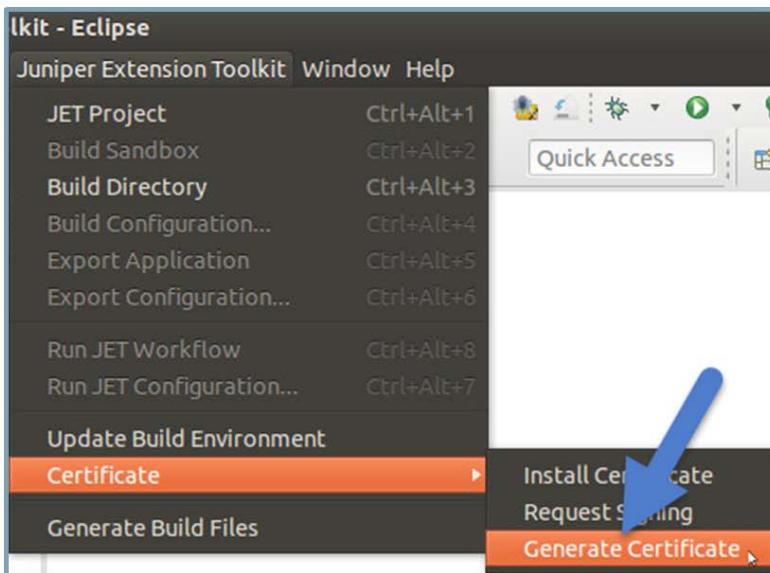
Create the Application

The next step is to code the application itself. Earlier, you created a folder for your project inside either the Python, lib, or bin folders. Next you want to add a code file to the folder for your application. Highlight the appropriate subdirectory create a file by either selecting File > New > File from the navigation menu or right-click the folder and select New > File, type a filename, and click Finish.

Create the application by writing code in the application files. To save your work, select File > Save or press Ctrl+s, or click the diskette icon.

Start Certificate Request Process

- To create a certificate request using the IDE:
 - Select Juniper Extension Toolkit > Certificate > Generate Certificate.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 23

Start Certificate Request Process

We have finished creating the application and now need to get a signed certificate from Juniper Networks that we can use to sign our applications. Without a signed application, Your Python apps will require Python to be enabled at the [edit system scripts language] hierarchy level in Junos.

In order to develop and distribute JET applications, you must install a package signing certificate onto the virtual machine (VM). You do this by creating a certificate request and sending it to Juniper Networks. When you receive the certificate, you install it in the VM.

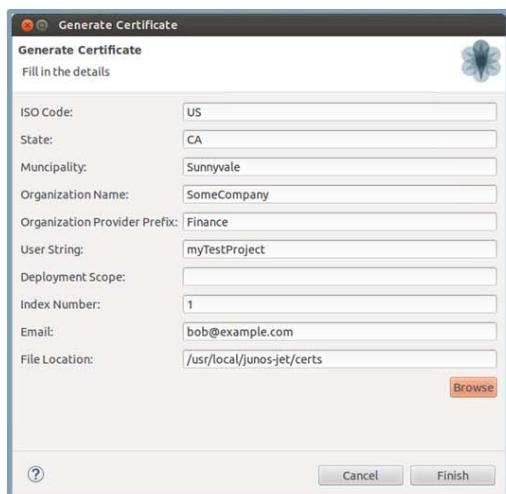
Caution: Never send your signing key to anyone, including Juniper Networks. The key enables anyone to sign applications that your router will trust. Therefore, it should be treated with the same level of security as the root password for the routers. Once you obtain your signing key, save it in a file outside of the VM.

Before you can create a certificate request, you must have the provider prefix which is a uniquely identifying prefix that represents the name of your organization. This prefix should have been provided to a contact at your organization. If you do not know this prefix, you must request it before running the jet-certificate-request command. Contact JET Certificate Processing at jet-cert@juniper.net.

From the navigation menu select Juniper Extension Toolkit > Certificate > Generate Certificate.

Fill Out Certificate Request Form

- To create a certificate request using the IDE:



- Complete the fields in the Generate Certificate Request pane
- Click Browse to select a directory in which to create the cert files. This is usually the /usr/local/junos-jet/certs directory.
- Click Finish.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER NETWORKS Worldwide Education Services

www.juniper.net | 24

Fill Out Certificate Request Form

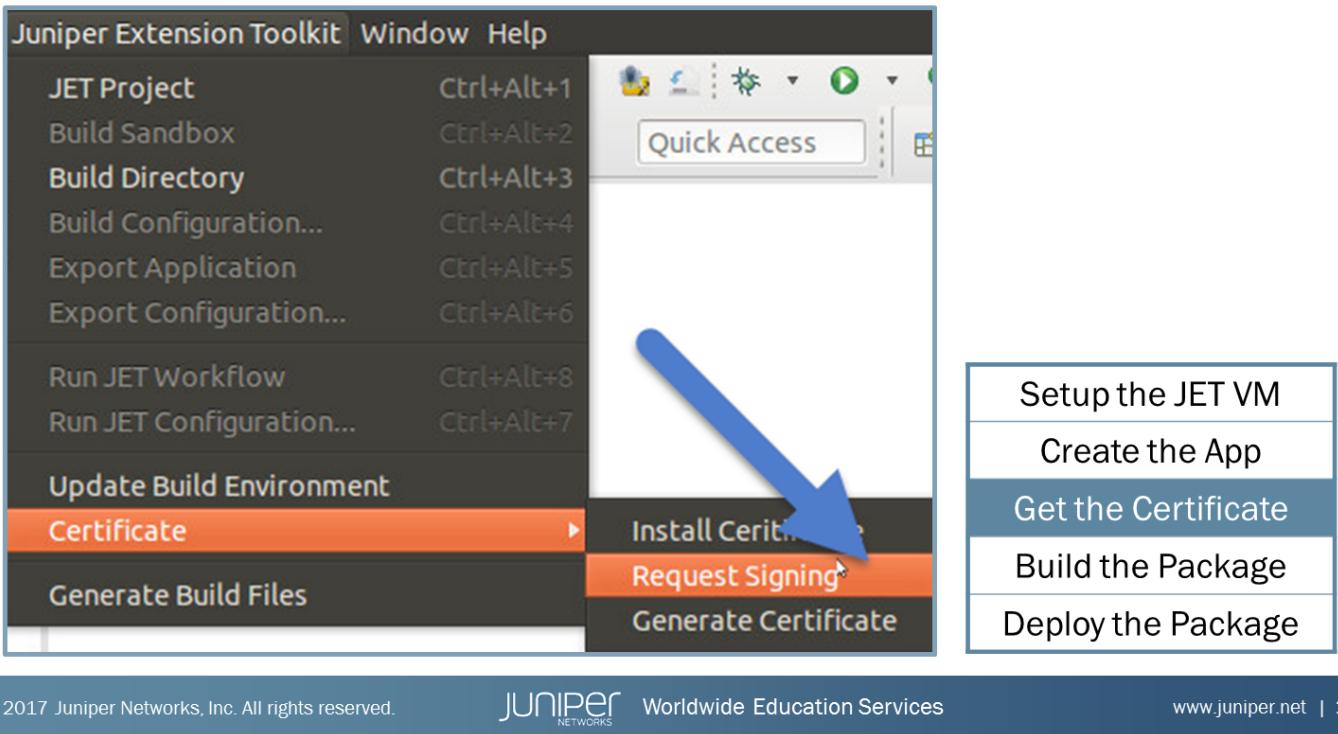
After selecting generate request from the navigation menu, you will see a Generate Certificate dialogue box. Complete the fields in the Generate Certificate Request pane as seen in the slide.

The script prompts for the following data:

- ISO Code – two letter country codes. A list of SSL Certificate ISO codes can be found at: <https://www.digicert.com/ssl-certificate-country-codes.htm>
- State
- Municipality
- Organization Name
- Organization Provider Prefix - This is a prefix your organization can request from Juniper if needed.
- User String - This is an additional specification of your choosing. It could be a string specifying the development team or project name. The user string can consist of a lowercase letter followed by one or more lowercase letters or numbers.
- Deployment Scope - Deployment scopes are either Commercial, Private, or Internal.
- Index number - This number is known as a certificate generations number. It is 1 for your initial certificate. When a certificate expires and a new one is requested, you must increment the number.
- E-mail address - We recommend against using a personal e-mail address for the certificate contact.

Send a Certificate Request to Juniper Networks

- Select Juniper Extension Toolkit > Certificate > Request Signing.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 25

Sent Certificate Request to Juniper

Once the certificate request has been generated, it needs to be sent to Juniper for signing.

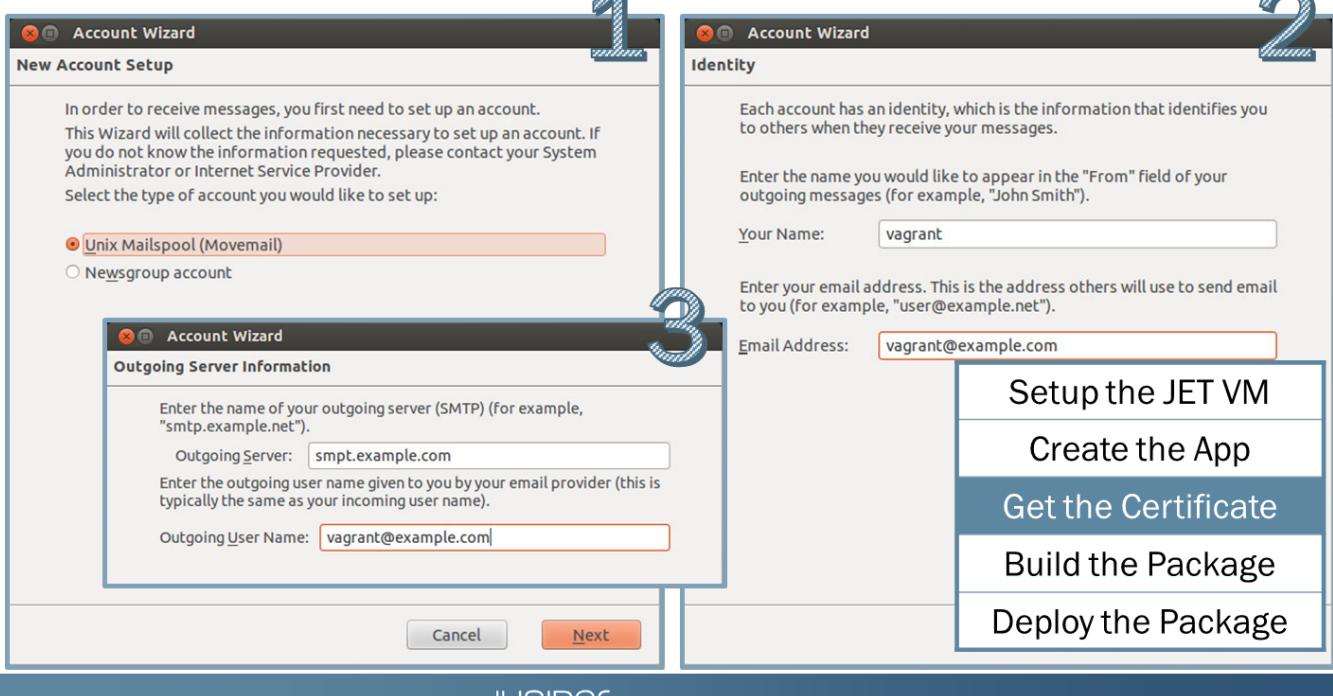
Caution: Never send your signing key to anyone, including Juniper Networks. The key enables anyone to sign applications that your router will trust. Therefore, it should be treated with the same level of security as the root password for the routers. Once you obtain your signing key, save it in a file outside of the VM.

Before you can create a certificate request, you must have the provider prefix—a uniquely identifying prefix that represents the name of your organization. This prefix should have been provided to a contact at your organization. If you do not know this prefix, you must request it before running the jet-certificate-request command. Contact JET Certificate Processing at jet-cert@juniper.net.

From the navigation menu Select Juniper Extension Toolkit > Certificate > Request Signing.

Fill Out Account Wizard Dialogue Boxes (1 of 2)

- Set up local e-mail SMTP server details and your e-mail details screens 1-3



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 26

Fill Out Account Wizard Dialogue Boxes, Part 1

To send the certificate request to Juniper you need to configure SNMP settings. the Account Wizard walks you through this process. The slide shows the dialogue boxes you will see as you request a Juniper signed certificate.

Fill Out Account Wizard Dialogue Boxes (2 of 2)

- Set up local e-mail SMTP server details and your e-mail details screens 4-5

Dialogue Box 4 (Account Name):

Account Name

Enter the name by which you would like to refer to this account (for example, "Work Account", "Home Account" or "News Account").

Account Name:

Dialogue Box 5 (Congratulations!):

Congratulations!

Please verify that the information below is correct.

Account Name:	vagrant@example.com
Email Address:	vagrant@example.com
Incoming Server Type:	MOVEMAIL
Outgoing User Name:	vagrant@example.com
Outgoing Server Name (SMTP):	smpt.example.com

Buttons:

- Cancel
- Back
- Next

Callout 5:

- Setup the JET VM
- Create the App
- Get the Certificate
- Build the Package
- Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

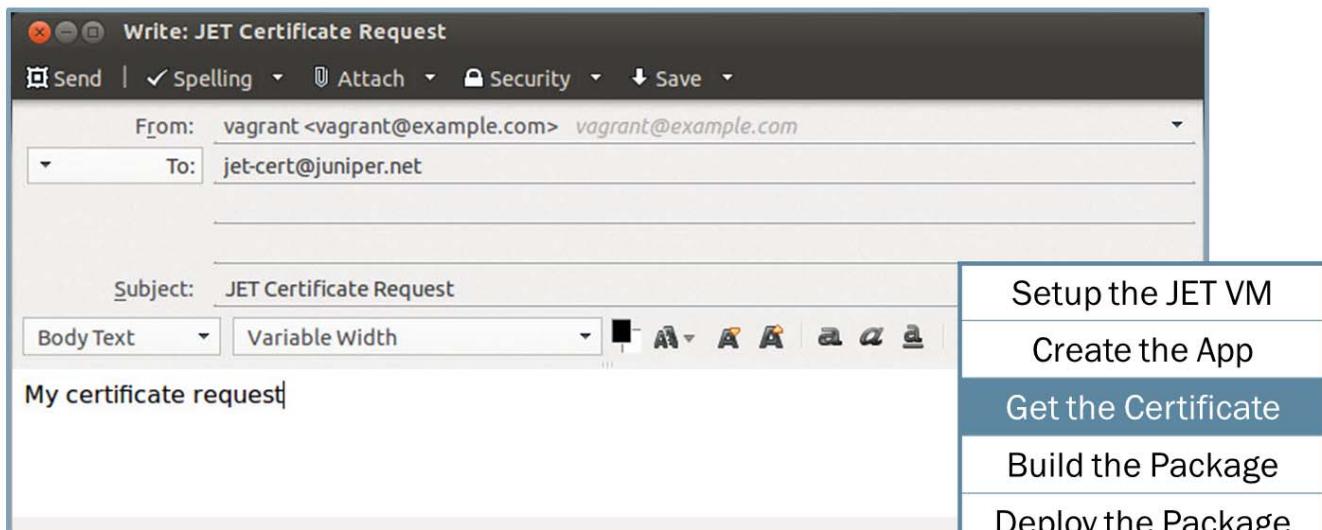
www.juniper.net | 27

Fill Out Account Wizard Dialogue Boxes, Part 2

The slide shows the last two dialogue boxes you will see in the Account Wizard.

Write JET Certificate Request E-mail

- Fill in the subject of the e-mail.
- Attach the certificate request file to the e-mail.
- Send the e-mail.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

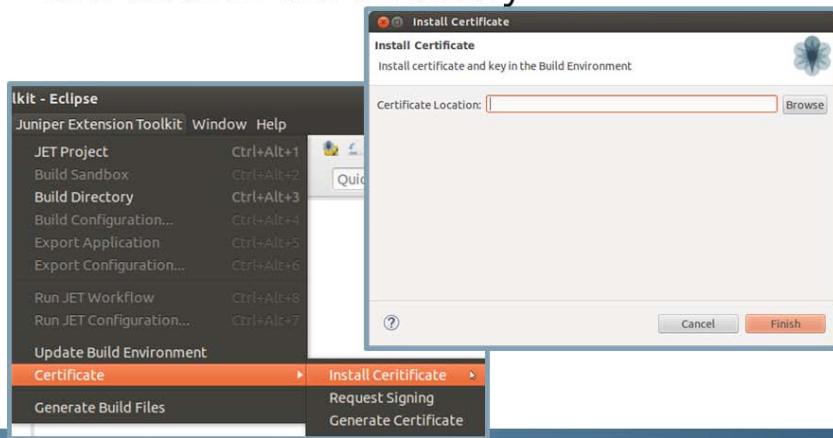
www.juniper.net | 28

Write JET Certificate Request E-mail

Once you have completed the Account Wizard you will be presented with the Write: JET Certificate Request Dialogue. Here you create the email request as seen in the slide. You will also need to attach the certificate request

Install the Certificate on the VM

- Select Juniper Extension Toolkit > Certificate > Install Certificate.
 - Navigate to the location where the certificate resides and click Finish.
 - Check that the certificates `foo_key.pem` and `foo.pem` are both in the directory



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER Worldwide Education Services

www.juniper.net | 29

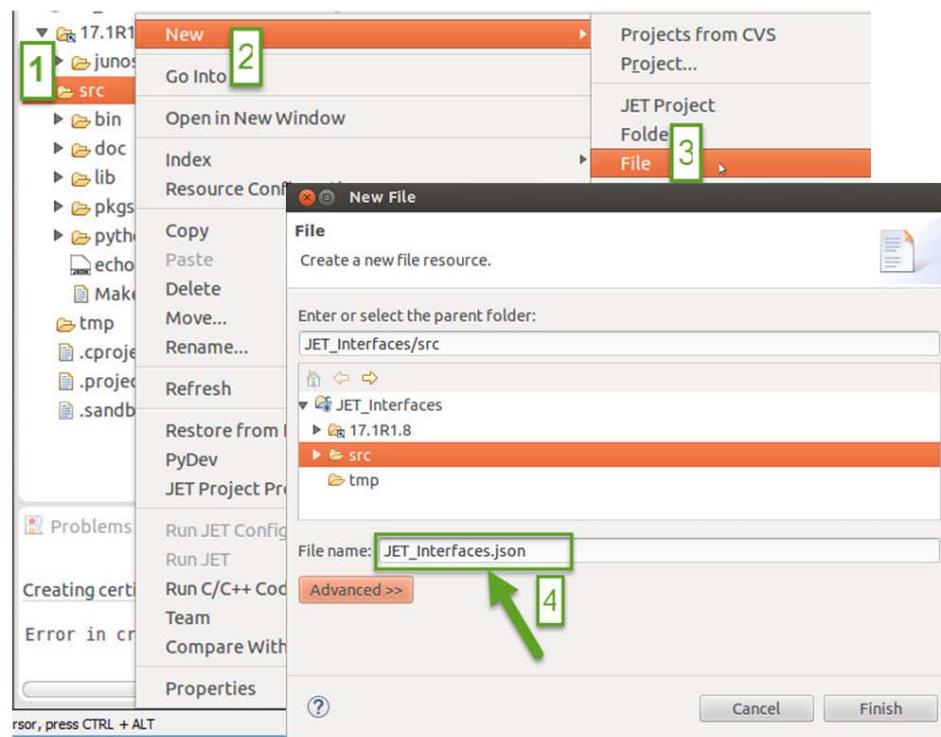
Install the Certificate on the JET VM

After you have sent your certificate signing request you will shortly receive a reply email with the signed certificate from Juniper. You will need to install this certificate.

To install the certificate Select Juniper Extension Toolkit > Certificate > Install Certificate. Navigate to the location where the certificate resides and click Finish. Check that the certificates `foo_key.pem` and `foo.pem` are both in the directory

Create a JSON Application Properties File

- To create the JSON file and the JET package:



Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 30

Create a JSON Application Properties File

Now that the certificate has been installed you can build the JET package. The first step in doing that is to create a JSON file that contains the application information needed to generate the package. The JET Extension toolkit also uses this information to autogenerated the makefiles and manifest file.

Create a JSON file by selecting the `src` directory in the Sandbox Explorer pane and either selecting `File > New > File` from the navigation menu or right-click and select `New > File`, type the filename (use the `.json` extension), and click `Finish`.

Fill Out Application Properties Dialogue

*Jet_Interfaces.json

Application Name	Jet_Interfaces
Application Path	bin/echoclientd
Language	python
Main scripts	Add Remove
Application type	standalone
Signed application	yes
Target OS	BSD 10
Target architecture	i386
Application description	This application configures Junos interfaces via JET.
C-Compiler-Flags	
CPP-Compiler-Flags	
Linker-Flags	
Select from the below list to manage source files and modules:	

- Provide the necessary information about the application and save the file.
- Save the file

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 31

Fill Out the Application Properties Dialogue Box

Once you have created the JSON file, you will need to add information to it. The JET Extension Toolkit provides a form to make this easy. When you open the JSON file, you will see the following form. Provide the necessary information about the application and save the file.

- **Application Name**—Specify the path to the application's implementation directory. If the application is a binary, the application name is treated as a binary name. The package name created is based on the value in this field.
- **Application Path**—Specify the path to the application directory.
- **Language**—Select the language used for developing the application.
- **Main scripts**—This is a list attribute and is generally applicable for Python applications. For binary applications, this item is optional. Main scripts are searched under the application path. Specify the filename or filenames of the main script or scripts that run on the device. Click **Add** to add a filename and path. Select a file and click **Remove** to remove a file.
- **Application type**—Select whether an application is to be a standalone program or a daemon.
- **Signed application**—Select yes or no.
- **Target OS**—Select BSD 6 for legacy Junos OS or BSD 10 for Junos OS with updated FreeBSD.
- **Target architecture**—Specify the target architecture on which the application is to be deployed.

Continued on the next page.

Fill Out the Application Properties Dialogue Box (contd.)

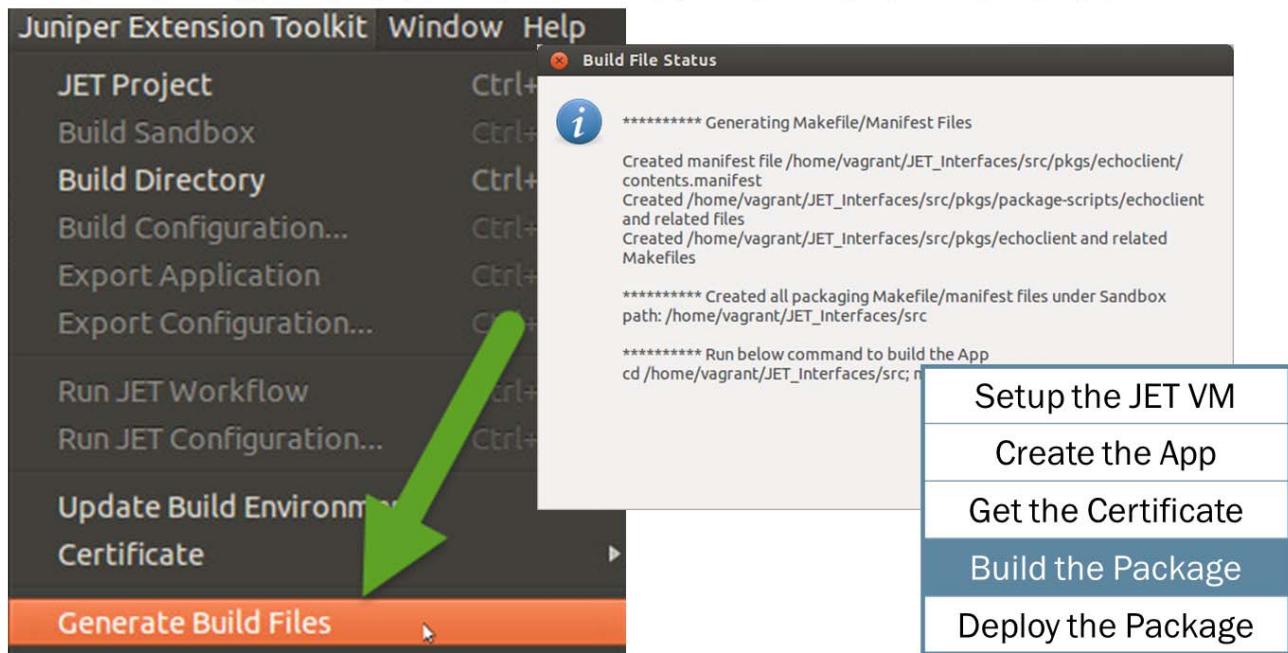
- **Application description**—Specify a brief (one-line) description about the application. This description will be displayed in the CLI show version operational command.
Specify the list of C compiler flags, the list of C++ compiler flags, and the list of linker flags, if any. Use linker flags to specify additional libraries to link with or additional link-specific flags that are required during linking.
- **Source files and modules**—Select a category and click Add to add a filename and path.

Click Preview at any point to see the JSON file format in code.

Do one of the following to save the JSON file: select File > Save, press Ctrl+s, or click the diskette icon.

Generate Build Files

- Select the JSON file and click Juniper Extension Toolkit > Generate Build Files from the menu bar.



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 33

Generate Build Files

Once the JSON file is complete you can Generate the build files. Next select the JSON file and click Juniper Extension Toolkit > Generate Build Files from the menu bar. The makefiles and manifest file are generated automatically.. and you will see the Build File Status Dialogue box as shown on the slide

Build and Create a Package

- Do one of the following:

- Juniper Extension Toolkit > Build Sandbox
 - Builds the applications using the configuration provided in Build Configuration.
- Juniper Extension Toolkit > Build Configuration
 - Selects which applications to build, which targets to build for, and build once you click Finish.
- Juniper Extension Toolkit > Run JET Configuration
 - Specifies which device to load the application on and what commands and configurations to include.

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

Build and Create Application Package

Next Select the project directory in the left pane and then select one of the following to build the package:

Juniper Extension Toolkit > Build Sandbox—Build the applications using the configuration provided in Build Configuration.

Juniper Extension Toolkit > Build Configuration—Select which applications to build, which targets to build for, and build once you click Finish.

Juniper Extension Toolkit > Run JET Configuration—Specify which device to load the application on and what commands and configurations to include.

Once the build is successful, the package is created in the project ship directory. The ship directory is in the project directory under the junos-jet-sb-obj directory. You can see the ship directory in the left pane once the package has been built.

Deploying the Application Package

- Copy the application-name.tar.gz file to the device running Junos OS, for example:

```
% scp application-name.tar.gz device-hostname:/var/tmp.
```

- Deploy the package.

```
user@host> request system software add application-  
name.tar.gz
```

- Verify the version of your application.

```
user@host# show version
```

```
..
```

```
..
```

```
JET echoserver example Application  
[16.1I20150115_1409_root]
```

Setup the JET VM
Create the App
Get the Certificate
Build the Package
Deploy the Package

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 35

Deploying the Application Package

Once the build is successful, the package is created and stored in the project ship directory. The ship directory is in the project directory under the junos-jet-sb-obj directory. You can see the ship directory in the left pane once the package has been built.

After locating the application package, copy it to the device running Junos OS. The slide show how to SCP the file, but you can use other methods as well.

To install the package use the request system software add command. After the package is installed you can use the show version command to verify the installation.

Agenda: Junos Extension Toolkit (JET)

- Overview of JET
- Creating Signed JET Apps
- Creating Unsigned JET Apps
- Creating JET Notification Apps

Creating Unsigned JET Apps

The slide highlights the topic we discuss next.

Create JET Off-Box App

- Install Python 2.7 and virtualenv
- Install gRPC tools
- Download, extract, and compile the Junos JET Client IDL Library for Python
- Write the application
- Configure permissions on Junos OS

Create JET Off-Box App

We just looked at creating an on-box signed application, now let's take a look at creating an off-box unsigned application. We will also take a look at generating or compiling the gRPC .proto files.

Creating an unsigned application does not require a signed cert which makes it simpler. It also will not require the use of the JET VM. The slide lists the steps we will go through.

Install gRPC Tools

- Pip install grpcio-tools

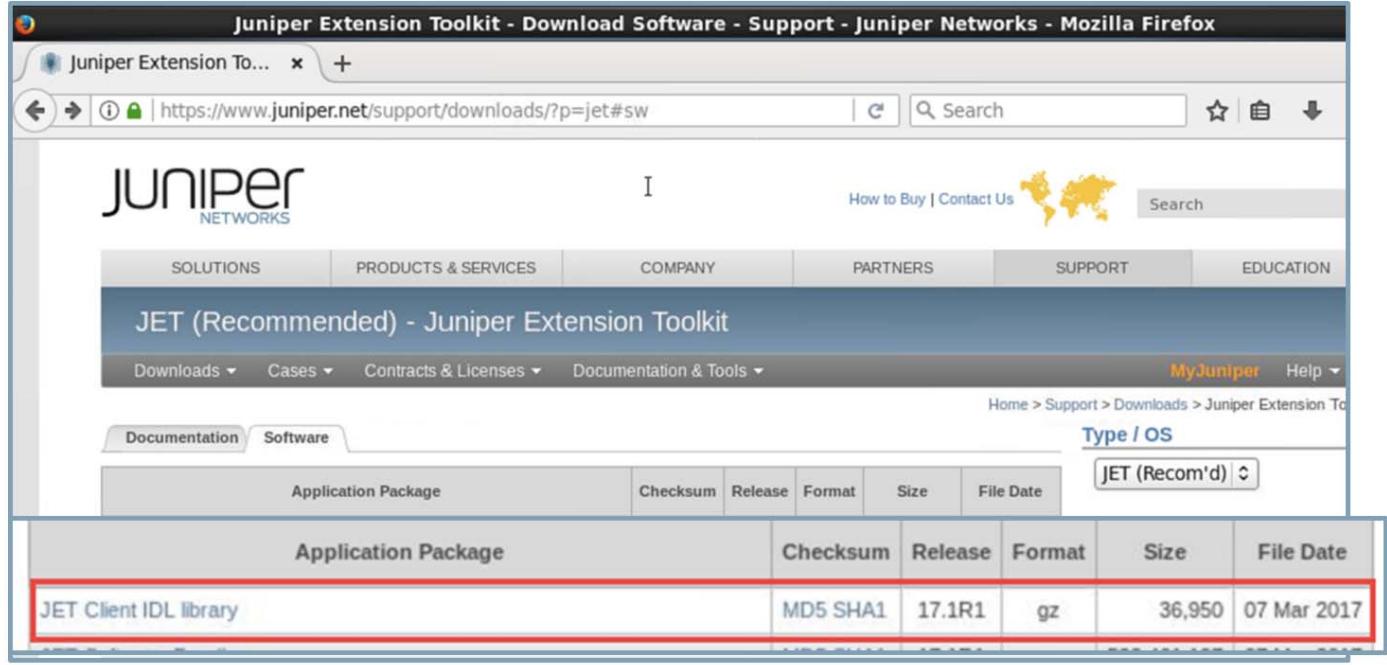
```
(jaut_env) [lab@jaut-desktop jaut_env]$ pip install  
grpcio-tools  
Collecting grpcio-tools  
  Extract files  
    ...Trimmed...
```

Install gRPC Tools

Installing the gRPC tools can be done using pip. If you are going to use virtualenv, you will want to create and activate it before installing the gRPC tools. Information on how to install and activate a virtual environment was presented earlier in the course.

Download the JET Client IDL Library

- <https://www.juniper.net/support/downloads/?p=jet#sw>



The screenshot shows a Mozilla Firefox browser window displaying the Juniper Extension Toolkit download page. The URL in the address bar is <https://www.juniper.net/support/downloads/?p=jet#sw>. The page header includes the Juniper Networks logo and navigation links for Solutions, Products & Services, Company, Partners, Support, and Education. A search bar and a globe icon are also present. The main content area is titled "JET (Recommended) - Juniper Extension Toolkit". Below this, a navigation bar offers links to Downloads, Cases, Contracts & Licenses, Documentation & Tools, MyJuniper, and Help. The "Downloads" tab is selected. On the right, there are filters for "Type / OS" set to "JET (Recom'd)" and a "Software" tab selected. A table lists available packages, with the first row, "JET Client IDL library", highlighted with a red border. The table columns are Application Package, Checksum, Release, Format, Size, and File Date. The highlighted row shows MD5 SHA1, 17.1R1, gz, 36,950, and 07 Mar 2017 respectively.

Application Package	Checksum	Release	Format	Size	File Date
JET Client IDL library	MD5 SHA1	17.1R1	gz	36,950	07 Mar 2017

© 2017 Juniper Networks, Inc. All rights reserved. **JUNIPER** Worldwide Education Services www.juniper.net | 39

Download the JET Client IDL Library

The JET Client IDL library are .proto files written in JSON. You can download the files from the Juniper download site. The URL is shown on the slide.

Extract the JET Client IDL Library Files

```
(jaut_env) [lab@jaut-desktop jaut_env]$ tar -xvf jet-
idl-17.1R1.8.tar.gz
proto
proto/authentication_service.proto
proto/bgp_route_service.proto
proto/cosd_service.proto
proto/dcd_service.proto
proto/firewall_service.proto
proto/jnx_addr.proto
proto/license_service_internal.proto
proto/mgd_service.proto
proto/mpls_api_service.proto
proto/openconfig_service.proto
proto/prpd_common.proto
proto/prpd_service.proto
proto/registration_service.proto
proto/rib_service.proto
(jaut_env) [lab@jaut-desktop jaut_env]$
```

Extract the JET Client IDL Library Files

Before you compile the .proto files you need to first unpack them. The slide shows the command to do that. The slide also shows the .proto files included as part of the IDL library. This list of files will be added to over time. Be sure to download the IDL files that correspond to the version of Junos you are using.

Compile the .proto Files

- Compile the .proto files into Python by issuing the
`python -m grpc_tools.protoc -Iproto/ -`
`python_out=$DST_DIR $SRC_DIR/class.proto` command
- Compile each of the files in the proto directory from the previous slide

```
(jaut_env) [lab@jaut-desktop jaut_env]$ python -m
grpc_tools.protoc -Iproto/ --python_out=jet_py/ --
grpc_python_out=jet_py/ proto/authentication_service.proto
(jaut_env) [lab@jaut-desktop jaut_env]$ python -m
grpc_tools.protoc -Iproto/ --python_out=jet_py/ --
grpc_python_out=jet_py/ proto/bgp_route_service.proto
```

- For more information see:
 - <https://developers.google.com/protocol-buffers/docs/pythontutorial>

Compile the .proto Files

The Python `grpcio` tools you installed earlier contains a compiler that will compile the `.proto` files into a Python compatible format. The slide shows the syntax for compiling the `.proto` files. You only need to compile the `.proto` files you will need. It is recommended that you compile them all so that you don't have to remember which ones have been compiled for later projects.

The slide also shows the URL where you can find more information on how to compile `.proto` files in Python and other languages.

Verify the Files Compiled

```
(jaut_env) [lab@jaut-desktop authentication_service_pb2_grpc.py  
authentication_service_pb2.py  
bgp_route_service_pb2_grpc.py  
bgp_route_service_pb2.py  
cosd_service_pb2_grpc.py  
cosd_service_pb2.py  
dcd_service_pb2_grpc.py  
dcd_service_pb2.py  
firewall_service_pb2_grpc.py  
firewall_service_pb2.py  
jnx_addr_pb2_grpc.py  
registration_service_pb2_grpc.py  
jnx_addr_pb2.py  
license_service_internal_pb2_grpc.py  
license_service_internal_pb2.py  
(jaut_env) [lab@jaut-desktop jaut_env]$ jaut_env]$ ls jet_py  
mgd_service_pb2_grpc.py  
mgd_service_pb2.py  
mpls_api_service_pb2_grpc.py  
mpls_api_service_pb2.py  
openconfig_service_pb2_grpc.py  
openconfig_service_pb2.py  
prpd_common_pb2_grpc.py  
prpd_common_pb2.py  
prpd_service_pb2_grpc.py  
prpd_service_pb2.py  
registration_service_pb2.py  
rib_service_pb2_grpc.py  
rib_service_pb2.py  
(jaut_env) [lab@jaut-desktop jaut_env]$
```

Verify the Files Compiled

Use the ls command to verify which files have been compiled.

Create Python JET Application (1 of 2)

- Create the script in your favorite editor

```
from jnpr.jet.JetHandler import *
try:
    client = JetHandler()
    client.OpenRequestResponseSession(device='127.0.0.1',
port=9090, user='foo',password='bar',client_id='myapp')
    intf_handle = client.GetInterfaceService()
    int_ip = InterfaceAddressConfig('ge-
0/0/1',0,InetFamilyType.INTF_AF_INET,'192.1.1.1/32')

    #Create Interface
    result = intf_handle.InterfaceAddressCreate(int_ip)
    print 'InterfaceAddressCreate : Invoked, return = ', result
```

Create Python JET Application, Part 1

Use your favorite Python editor to create your script. The slide shows a sample Python script that Opens a request and response session, Creates an interface and deletes an interface.

Create Python JET Application (2 of 2)

- Create the script in your favorite editor

```
#Delete Interface
result = intf_handle.InterfaceAddressDelete(int_ip)
print 'InterfaceAddressDelete : Invoked, return = ', result

client.CloseRequestResponseSession()
except Exception as tx:
    print '%s' % (tx.message)
```

Create Python JET Application, Part 2

The slide shows how the interfaces is deleted and the session is closed.

More information about the Junos JET service APIs can be found at:

http://www.juniper.net/documentation/en_US/jet17.1/information-products/pathway-pages/product/17.1/index.html

Configure Junos to Run the Application

- Save the file as myArgs.py

- Copy the Python script to

/var/db/scripts/jet

- Set the script language to Python

[edit]

```
lab@vMX-1# set system scripts language python
```

- Authorize the script

[edit system extensions extension-service application]

```
user@device# set file myInterfaces.py
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 45

Configure Junos to Run the Unsecured Python Application

Once the script is written, you will want to copy it to the device running the Junos OS. Because the application is a JET application, the location to store the file is different than for a Junos commit, op, event, or SNMP script. The script needs to be stored in the /var/db/scripts/jet directory.

Because the application is an unsigned application you will still need to issue the `set system scripts language python` command.

The script will also need to be authorized to execute. At the `[edit system extensions extension-service application]` hierarchy issue the `set file myInterfaces.py` command for the `myInterfaces.py` file.

Agenda: Junos Extension Toolkit (JET)

- Overview of JET
- Creating Signed JET Apps
- Creating Unsigned JET Apps
- Creating JET Notification Apps

Creating JET Notification Apps

The slide highlights the topic we discuss next.

JET Notification Applications

- Typically ran off-box
- Created using Python script
- Can subscribe to multiple events simultaneously

JET Notification Applications

Jet Notification apps are similar to JET services application, but instead of issuing requests to the Junos OS, it subscribes to the Junos JET message service and the listens for streaming messages.

JET Notification Message Format

- JET Notification Messages use JSON

```
"jet-event": {  
    "event-id": "KERNEL_EVENT_IFD_ADD"  
    "hostname": "mydevice",  
    "time": "2016-01-07",  
    "severity": "info",  
    "facility": "KERNEL",  
    "attributes": {  
        "name": "ge-0/0/0",  
        "snmp-id": 520,  
        "flags": 8  
    }  
}
```

JET Notification Message Format

The slide shows a sample JET Notification message for the KERNEL_EVENT_IFD_ADD event. JET Notification messages are in JSON.

Subscription Events, Topics, and Return Values

- The JET Notification API allows applications to subscribe to events

Physical Interface (IFD)

- Each event has multiple topics

/junos/events/kernel/interfaces/ifd/add//ifdname
/junos/events/kernel/interfaces/ifd/change/ifdname
/junos/events/kernel/interfaces/ifd/delete/ifdname

- Each event returns specific information

name, snmp-id, flags

Subscription Events, Topics, and Return Values

The Junos Notification API has many different Events that an application can subscribe to. When an application subscribes to an event, it subscribes to a specific event topic. The slide shows the three topics belonging to the IFD event.

Each event also has specific information that it returns. Below is a table containing the Junos Notification events, topics, and return values. This information comes from: https://www.juniper.net/documentation/en_US/jet16.2/topics/concept/jet-notification-api-overview.html.

Continued on the next page.

Subscription Events, Topics, and Return Values (contd.)

Events	Topic	Event Information Returned
Physical Interface (IFD)	/junos/events/kernel/interfaces/ifd/add//ifdname /junos/events/kernel/interfaces/ifd/change/ifdname /junos/events/kernel/interfaces/ifd/delete/ifdname	name, snmp-id, flags
Logical Interface (IFL)	/junos/events/kernel/interfaces/ifl/add/iflname /junos/events/kernel/interfaces/ifl/change/iflname /junos/events/kernel/interfaces/ifl/delete/iflname	name, subunit, snmp-id, flags
Family (IFF)	/junos/events/kernel/interfaces/iff/add/iflname/family-type /junos/events/kernel/interfaces/iff/change/iflname/family-type /junos/events/kernel/interfaces/iff/delete/iflname/family-type	name, subunit, family, table-name, flags
Address	/junos/events/kernel/interfaces/ifa/add/iflname/family-type/address /junos/events/kernel/interfaces/ifa/change/iflname/family-type/address /junos/events/kernel/interfaces/ifa/delete/iflname/family-type/address	name, subunit, family, local-address, destination-address, broadcast-address, flags
Firewall	/junos/events/kernel/firewall/filter/add/filtername /junos/events/kernel/firewall/filter/change/filtername /junos/events/kernel/firewall/filter/delete/filtername	name, version, client-id, filter-type, protocol, interface-name, flags
Route	/junos/events/kernel/route/add/family/prefix-with-length /junos/events/kernel/route/change/family/prefix-with-length /junos/events/kernel/route/delete/family/prefix-with-length	table-name, logical-router-name, address-family, route-type, route-prefix, arrayof(nexthop-address), flags
Route-table	/junos/events/kernel/route-table/add/tablename/lrname /junos/events/kernel/route-table/change/tablename/lrname /junos/events/kernel/route-table/delete/tablename/lrname	name, logical-router-name, address-family, flags
Syslog	/junos/events/syslog/event-id	arrayof(attribute-value pairs)

Sample JET Notification Script (1 of 7)

```

#!/usr/bin/env python
import argparse
import os
import time
import logging
import sys

# JET shim layer imports
from jnpr.jet.JetHandler import *

# Default Notification topic parameters
DEFAULT_NOTIFICATION_IFD_NAME = "ge-1/0/4"
DEFAULT_NOTIFICATION_IFL_NAME = "ge-1/2/3"
DEFAULT_NOTIFICATION_IFL_UNIT = "35"

# Logging Parameters
DEFAULT_LOG_FILE_NAME =
os.path.basename(__file__).split('.')[0] + '.log'
DEFAULT_LOG_LEVEL = logging.DEBUG

```

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 51

Sample JET Notification Script, Part 1

This and the following 6 slides show a sample JET Notification script. The script comes from: https://www.juniper.net/documentation/en_US/jet1.0/topics/reference/jet-notification-app.html

Notice the `jnpr.jet.JetHandler` module. This is needed to help subscribe to the JET event streams. The rest of the code in the slide assigns values to various variable used in the script.

Sample JET Notification Script (2 of 7)

```
# Enable Logging to a file
logging.basicConfig(filename=DEFAULT_LOG_FILE_NAME,
level=DEFAULT_LOG_LEVEL)

"""
# To display the messages from junos-jet-api package on the
screen uncomment the below line
myLogHandler = logging.getLogger()
myLogHandler.setLevel(logging.INFO)
logChoice = logging.StreamHandler(sys.stdout)
logChoice.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(name)s -
%(levelname)s - %(message)s')
logChoice.setFormatter(formatter)
myLogHandler.addHandler(logChoice)

"""

```

Sample JET Notification Script, Part 2

This section shows how to enable logging and how to configure the script to show output to the screen.

Sample JET Notification Script (3 of 7)

```
def handleEvents1(message):
    print "I am in first handle"
    print "Event Received : " + message['jet-event']['event-id']
    print "Attributes : ", message['jet-event']['attributes']
    return

def handleEvents2(message):
    print "I am in second handle"
    print "Event Received : " + message['jet-event']['event-id']
    print "Attributes : ", message['jet-event']['attributes']
    return
```

Sample JET Notification Script, Part 3

This code shows two functions that show you can have multiple event handlers in the same application.

Sample JET Notification Script (4 of 7)

```
def Main():
    parser =
    argparse.ArgumentParser(prog=os.path.basename(__file__),
                           description='Sample JET application')

    parser.add_argument("--address", nargs='?', help="Address
of the JSD server. (default: %(default)s)", type=str,
default='localhost')

    parser.add_argument("--port", nargs='?', help="Port
number of the JSD notification server. default: %(default)s",
type=int, default=1883)

    parser.add_argument("--bind_address", nargs='?', help="Client source address to bind.",
type=str, default="")

args = parser.parse_args()
```

Sample JET Notification Script, Part 4

On this slide we can see the start of code execution with the `main()` function. You can also see how the arguments are passed in and added to the `args` object so they can be easily reference later in the script.

Sample JET Notification Script (5 of 7)

```

try:
    # Create a client handler for connecting to server
    client = JetHandler()

    # open session with MQTT server
    evHandle =
client.OpenNotificationSession(device=args.address,
port=args.port, bind_address=args.bind_address)

    # different ways of creating topic
    ifdtopic = evHandle.CreateIFDTopic(op=evHandle.ALL,
ifd_name=DEFAULT_NOTIFICATION_IFD_NAME)
    ifltopic = evHandle.CreateIFLTopic(evHandle.ALL,
DEFAULT_NOTIFICATION_IFL_NAME, DEFAULT_NOTIFICATION_IFL_UNIT)
    ifatopic = evHandle.CreateIFATopic(evHandle.DELETE)
    ifftopic = evHandle.CreateIFFTopic(evHandle.CHANGE)
    rtmtopic =
evHandle.CreateRouteTableTopic(evHandle.ALL)
    rttopic = evHandle.CreateRouteTopic(evHandle.ALL)

```

Sample JET Notification Script, Part 5

This section of code shows how the `JetHandler()` method and the `args` object are used to create a connection to the device running Junos OS.

The slide also shows several ways to subscribe to different events. There are six separate topic subscription topics created.

Sample JET Notification Script (6 of 7)

```
# Subscribe for events
print "Subscribing to IFD notifications"
evHandle.Subscribe(ifdtopic, handleEvents1)
evHandle.Subscribe(ifltopic, handleEvents2)
evHandle.Subscribe(ifatopic, handleEvents1)
evHandle.Subscribe(rtmtopic, handleEvents2)
evHandle.Subscribe(rttopic, handleEvents2)

time.sleep(5)

# Unsubscribe events
print "Unsubscribe from all the event notifications"
evHandle.Unsubscribe()
```

Sample JET Notification Script, Part 6

On this slide five of the six topics are subscribed to using the `evHandle.Subscribe()` method. Notice how three of the topics use `handleEvents1` and the other two use `handleEvents2` to process the events. This allows you to gather and print different information depending upon the event.

After the subscriptions are created the script listens for 5 seconds and then unsubscribes from all events.

Depending upon your needs, you may want to the script to sleep for a much longer time than 5 seconds.

Sample JET Notification Script (7 of 7)

```
# Close session
print "Closing the Client"
client.CloseNotificationSession()

except Exception, tx:
    print '%s' % (tx.message)

return

if __name__ == '__main__':
    Main()
```

Sample JET Notification Script, Part 7

Once the script has unsubscribed from all the topics, you then close the session and exit the program.

Summary

- In this content, we:

- Setup a JET VM
- Created JET Packages
- Used the JET API

We Discussed:

- How to setup a JET VM;
- How to Create JET Packages; and
- How to use the JET API.

Review Questions

1. What is the MQTT protocol used for?
2. Do you need to use the JET VM to create signed packages?
3. In what language are .proto files written?

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 59

Review Questions

- 1.
- 2.
- 3.

Answers to Review Questions

1.

The MQTT protocol is used for JET notification API.

2.

Yes, you need to use the JET VM to create signed packages.

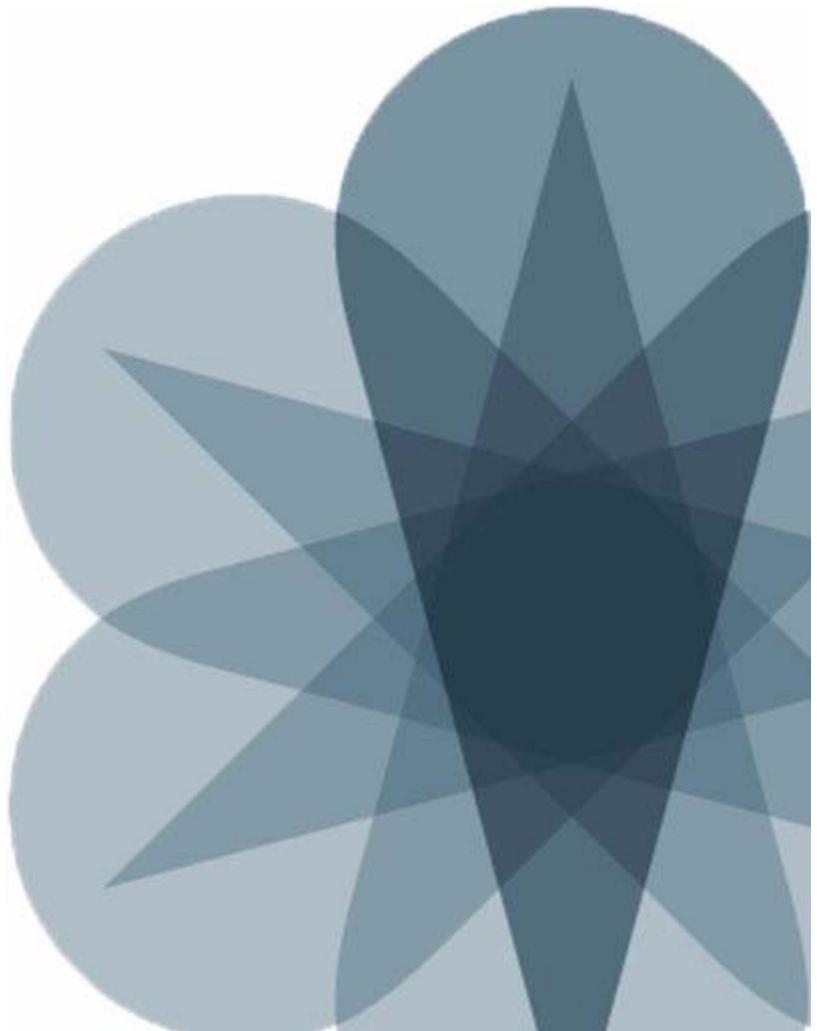
3.

The .proto files are written in JSON.



Junos Platform Automation and DevOps

Chapter 14: The Junos OS REST API



Objectives

- After successfully completing this content, you will be able to:
 - Describe the purpose of the Junos OS REST API
 - Create REST API RPC queries
 - Use the REST API Explorer

We Will Discuss:

- Describing the purpose of the Junos OS REST API,
- Creating REST API RPC queries, and
- Understanding the use of the REST API Explorer.

Agenda: The Junos OS REST API

- REST API Overview
 - Installing the REST API
 - Using the REST API
 - REST API Explorer

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 3

The Junos OS REST API Overview

The slide lists the topics we will discuss. We discuss the highlighted topic first.

What is the Junos REST API?

- Representational State Transfer (REST) is a method of locating, transferring, and manipulating data using URIs and the HTTP protocol
- Enables you to:
 - Securely connect to Junos OS devices
 - Execute RPCs (remote procedure calls)
- Includes a GUI-based API explorer
- Includes the ability retrieve data in text, JSON and XML formats

What Is the Junos REST API?

The Junos REST API is a Representational State Transfer (REST) interface that enables you to securely connect to Juniper Networks Junos operating system (Junos OS) devices, execute remote procedure calls (rpc commands), use a REST API Explorer GUI, which enables you to conveniently experiment with any of the REST APIs and use a variety of formatting and display options including JavaScript Object Notation (JSON).

In the context of web applications, an API is typically defined as a set of HTTP request messages along with a definition of the structure of response messages. APIs give flexibility to extend the basic service-provided functionality; you can build upon it to suit your needs.

Benefits of Junos REST API

- Enables Junos OS devices to participate in REST management system environments
- NETCONF and SSH not necessary to execute RPCs
- Easy to set up and configure
- Provides GUI API explorer for non-programmers

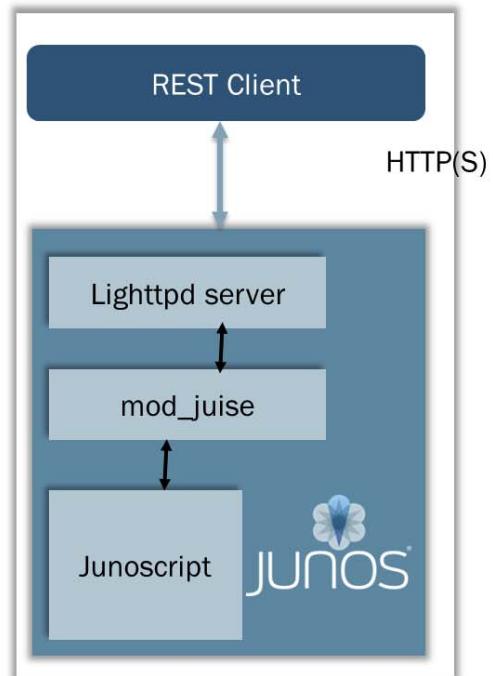
Benefits of Junos REST API

Representational state transfer (REST) is an architectural style that consists of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. Networks now not only include network gear, but host a variety of devices performing multiple tasks. Each device provides REST interfaces for the other devices to communicate with, and the ability to program on top of it.

Junos exposes NETCONF and Junoscript APIs, which enables developers to program devices and build network applications that can manage Juniper devices. In order to use these APIs, a client application must understand NETCONF protocol and be able to use SSH to communicate with Junos box. In a data center style setup, most devices communicate over HTTP, using REST APIs provided by target machines. In order to manage Junos devices in such cases, one needs a NETCONF adapter/translator that can communicate with the network gear from client machine. By opening up a REST API to Junos devices, it will be easier to program Juniper devices in an enterprise setup.

REST API Architecture

- REST Client makes HTTP Request
- Lighttpd server makes HTTP or HTTPS connection and passes request to mod_juise
- Mod_juise authenticates connection and passes request to Junoscript and the MGD



REST API Architecture

After REST is configured in the Junos OS CLI and committed, the Lighttpd server starts listening for incoming connections. If a request is made, the Lighttpd web server handles the get or post request and passes the data onto the mod_juise plugin. The mod_juise plugin then handles the authentication of the user. Once the user is authenticated, the RPC is then handed over to the MGD process. The MGD returns the data back to mod_juise, which reformats the data for the Lighttpd web server, which then passes the response back to the REST client.

Agenda: The Junos OS REST API

- REST API Overview
- Installing the REST API
- Using the REST API
- REST API Explorer

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 7

Installing the Junos OS REST API

The slide lists the topic we will discuss next.

How to Enable REST using HTTP

- Use HTTP for testing

[edit]

```
lab@vMX-1# set system services rest http
```

- Customize HTTP port #

[edit system services]

```
lab@vMX-1# set rest http port 3030
```

HTTP default port#
3000

- Customize HTTPS port #

[edit system services]

```
lab@vMX-1# set rest https port 4040
```

HTTPS default port#
3443

- The Junos REST API works with IPv4 only

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 8

How to Enable REST using HTTP

To configure HTTP access:

Specify `set system services rest http addresses [addresses]` to set the addresses on which the server will listen for incoming HTTP connections.

Specify `set system services rest http port port-number` to set the TCP port for incoming HTTP connections. You can assign a value from 1024 through 65535 (the default is 3000).

Note: The Junos OS REST API only works with IPv4.

How to Enable REST using HTTPS

- Use HTTPS to secure your REST connection

```
[edit]
lab@vMX-1> request security pki generate-key-pair
certificate-id REST_CERT
Generated key pair REST_CERT, key size 1024 bits

[edit]
lab@vMX-1> request security pki local-certificate generate-
self-signed certificate-id REST_CERT subject CN=my-test
domain-name vMX-1.my-test.org email lab@my-test.org
Self-signed certificate generated and loaded successfully

[edit]
lab@vMX-1> configure
Entering configuration mode

[edit]
lab@vMX-1> set system services rest https server-certificate
REST_CERT
```

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 9

How to Enable REST using HTTPS

To configure HTTP access:

Specify `set system services rest http addresses [addresses]` to set the addresses on which the server listens for incoming HTTP connections.

Specify `set system services rest http port port-number` to set the TCP port for incoming HTTP connections. You can assign a value from 1024 through 65535 (the default is 3000).

Note: The Junos OS REST API only works with IPv4.

Controlling Access to the REST API

- You can control the number of concurrent connections and specify a list of IP addresses permitted to connect

```
[edit system services]
```

```
lab@vMX-1# set rest control ?
```

Possible completions:

+ allowed-sources	List of allowed source IP addresses
+ apply-groups	Groups from which to inherit configuration data
+ apply-groups-except	Don't inherit configuration data from these groups
connection-limit	Maximum number of simultaneous connections (1..1024)

Controlling Access to the REST API

You can restrict access to the REST API server by limiting the IP address to those on a list you specify. You can also set a limit to the number of concurrent connections to the REST API server.

Agenda: The Junos OS REST API

- REST API Overview
- Installing the REST API
- Using the REST API
- REST API Explorer

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 11

Using The Junos OS REST API

The slide lists the topic we will discuss next.

Making RESTful Web Browser Queries

- Query Format:

scheme://device-name:port/rpc/method

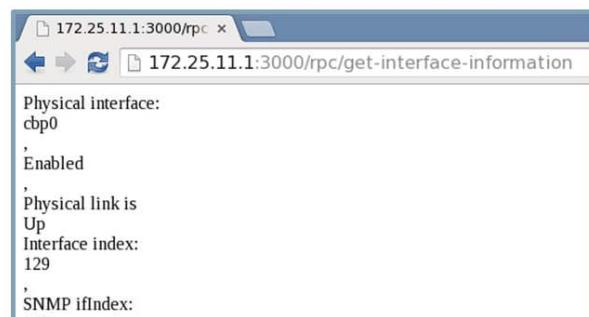
- Scheme = HTTP or HTTPS

- Device-name = URL or IP address

- Port = 3000 for HTTP or 3443 for HTTPS unless you configured it differently

- Method = RPC i.e. show-interface-information

http://172.25.11.1:3000/rpc/get-interface-information



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 12

Making RESTful Web Browser Queries

Creating queries in a web browser can be useful to quickly gather information from a remote device or create a series of bookmarks that can be opened to check the status of a device. It can also be useful in developing web based tools that have integrated web browsers.

The slide shows basic information about browser based queries. The lab accompanying this chapter allows you to create and test browser based queries.

Making RESTful curl Queries

- curl is a command line program that can be used to make RESTful queries
- Query Format:

```
curl -u "username:password" http://device-name:port/rpc/method
```

- -u = username and password
- Device-name = URL or IP address
- Port = 3000 for HTTP or 3443 for HTTPS unless you configured it differently
- Method = RPC i.e. show-interface-information

```
Curl -u "lab:lab123" http://172.25.11.1:3000/rpc/get-interface-information
```

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 13

Making RESTful cURL Queries

cURL is a command line application designed to transport data using many different formats including HTTP. The slide shows the basics of creating cURL queries.

cURL is used by many web based applications to issue REST queries and parse the returned data.

In the lab you will use cURL to query the Junos REST API server

Making RESTful curl Queries with Attributes and Parameters

```
[lab@jaut-desktop ~]$ curl -u "lab:lab123"
"http://172.25.11.1:3000/rpc/get-interface-
information@format=text?interface-name=ge-0/0/0&terse="
Interface          Admin Link Proto Local
Remote
ge-0/0/0           up    up
ge-0/0/0.0         up    up    inet   172.17.1.1/24
                           multiservice
```

■ Notes:

- Quotes – Include URL in quotes so that special characters do not cause problems
- Attributes – Attributes like `format=text` are prefaced with an @ sign
- The `terse` statement – Even though the `terse` statement does not take a value, the equals (=) sign is required

Making RESTful curl Queries with Attributes and Parameters

The slide shows the curl equivalent of the `show interfaces ge-0/0/0 terse` command. By default, the output format is in XML. You can use the `format=text` attribute to specify the return format. Because `format=text` is an attribute, it needs to be preceded with the @ sign. Other possible formats include JSON and XML.

The RPC and any of the RPC parameters are separated with a ?.

The `terse` parameter doesn't take a value, so you can either state that `terse=""` or include the `terse=` statement without the quotes.

If you are having difficulty coming up with a working statement, you may want to experiment using the Junos REST API Explorer, which we will talk about next.

Agenda: The Junos OS REST API

- REST API Overview
 - Installing the REST API
 - Using the REST API
- REST API Explorer

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 15

REST API Explorer

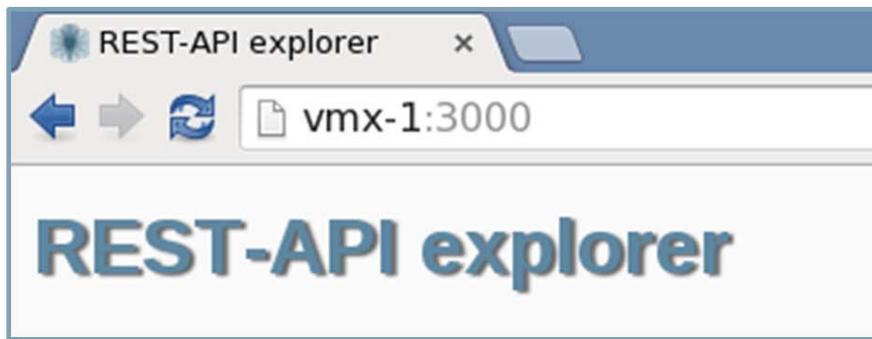
The slide lists the topic we will discuss next.

Enabling the Junos REST API Explorer

- Enable the REST API Explorer

```
[edit]
user@R1# set system services rest enable-explorer
[edit]
lab@vMX-1# commit
commit complete
```

- Open the REST API Explorer in a browser



© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 16

Enabling the Junos REST API Explorer

To enable the REST API Explorer, issue the `set system services rest enable-explorer` command at the `[edit]` hierarchy. After you commit the configuration, navigate in a browser to the URL or IP address of your Junos device, then append the configured port number at the end. The default port number is 3000 for https and 2443 for http. You will be prompted for a user name and password.

Executing a Single RPC (1 of 3)

Choose between GET and PUT

Choose between submitting a single RPC or multiple RPCs

Choose your output format:
XML, JSON, plain text

The screenshot shows the REST API Explorer interface. On the left, there's a sidebar with 'Request Headers' containing a curl command. The main area has fields for 'HTTP method' (set to 'GET'), 'Required output format' (set to 'XML'), 'RPC URL' ('/rpc/get-interface-information'), 'Username' ('lab'), and 'Password' ('*****'). A 'Submit' button is at the bottom. To the right, the 'Response' pane shows XML output for the RPC call.

```

Request Headers
GET /rpc/get-interface-information HTTP/1.1
Authorization: Basic bGF1OmXhYjEyMw==
Accept: application/xml
Content-Type: application/xml

curl http://vmx-1:3000/rpc/get-interface-information -u "lab:lab123" -H "Accept: application/xml"
  
```

```

<interface-information>
<physical-interfaces>
<name>ge-0/0</name>
<admin-status>oper-status</admin-status>
<link-index>518</link-index>
<link-level-type>Ethernet</link-level-type>
<mtu>1514</mtu>
<sonet-mode>LAN-PHY</sonet-mode>
<mru>1522</mru>
<source-filtering>disabled</source-filtering>
<speed>1000Mbps</speed>
<bpdus-error>none</bpdus-error>
<ld-pdu-error>none</ld-pdu-error>
<l2pt-error>none</l2pt-error>
<loopback>disabled</loopback>
<if-flow-control>enabled</if-flow-control>
<pad-to-minimum-frame-size>Disabled</pad-to-minimum-frame-size>
<if-device-flags>
<ifdf-present/>
<ifdf-running/>
  
```

Single RPC Multiple RPCs

GET

XML

/rpc/get-interface-information

lab

.....

Submit

Enter the RPC and all necessary parameters

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 17

Executing a Single RPC, Part 1

Executing a single is the default mode and selected when opening the REST API Explorer. Queries to the REST API can be submitted as either an HTTP GET or HTTP POST requests.

The Junos REST API supports output methods of XML, JSON, and plain text.

The RPC URL is where you input the requested RPC.

The Username and Password fields are where you supply credentials for the Junos device.

Executing a Single RPC (2 of 3)

The diagram illustrates the Junos REST API interface for executing a single RPC. It shows the following components:

- Request Headers:** A box containing the HTTP headers sent to the Junos REST API server. A red callout points to it from the top right, labeled "Data included in the HTTP GET or POST request headers".
- cURL request:** A box containing the generated cURL command used to make the request. A red callout points to it from the middle right, labeled "The curl request".
- Response:** A box showing the XML response received from the Junos REST API server. The XML content includes details about an interface, such as operational status, link-level type (Ethernet), speed (1514kbit/s), and various error and configuration parameters.

```

Request Headers
GET /rpc/get-interface-information HTTP/1.1
Authorization: Basic bGF1OmjhYjEyMw==
Accept: application/xml
Content-Type: application/xml

curl http://vmx-1:3000/rpc/get-interface-information -u "lab:lab123" -H
"Content-Type: application/xml" -H "Accept: application/xml"

<oper-status>up</oper-status>
<local-index>148</local-index>
<smp-index>518</smp-index>
<link-level-type>Ethernet</link-level-type>
<speed>1514kbit/s</speed>
<sonet-mode>LAN-PHY</sonet-mode>
<error>152</error>
<source-filtering>disabled</source-filtering>
<speed>1000mbps</speed>
<bpdu-error>none</bpdu-error>
<ld-pdu-error>none</ld-pdu-error>
<l2pt-error>none</l2pt-error>
<loopback>disabled</loopback>
<if-flow-control>enabled</if-flow-control>
<pad-to-minimum-frame-size>disabled</pad-to-minimum-frame-size>
<if-device-flags>
<ifdf-present/>
<ifdf-running/>

```

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 18

Executing a Single RPC, Part 2

The Request Headers field contains a copy of the data included in the HTTP header field that is sent to the Junos REST API server.

The cURL request is a generated request that you can take and use in your cURL application.

Executing a Single RPC (3 of 3)

Response Headers

```
Content-Type: application/xml; charset=utf-8
Transfer-Encoding: chunked
Date: Sat, 20 May 2017 01:29:36 GMT
Server: lighttpd/1.4.32
```

Response Body

```
<interface-information xmlns="http://xml.juniper.net/junos/17.1R1/junos-ini...
<physical-interface>
<name>ge-0/0/0</name>
<admin-status junos:format="Enabled">up</admin-status>
<oper-status>up</oper-status>
<local-index>140</local-index>
<snmp-index>518</snmp-index>
<link-level-type>Ethernet</link-level-type>
<mtu>1514</mtu>
<sonet-mode>LAN-PHY</sonet-mode>
<mru>1522</mru>
<source-filtering>disabled</source-filtering>
<speed>1000mbps</speed>
<bpdu-error>none</bpdu-error>
<ld-pdu-error>none</ld-pdu-error>
<l2pt-error>none</l2pt-error>
<loopback>disabled</loopback>
<if-flow-control>enabled</if-flow-control>
<pad-to-minimum-frame-size>Disabled</pad-to-minimum-frame-size>
```

Data returned in the response header

RPC Response

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 19

Executing a Single RPC, Part 3

The Response Header contains the content of the HTTP header returned from the Junos REST API web server.

The Response body contains the RPC response from the Junos REST API server.

Executing Multiple RPCs at Once

The screenshot shows a configuration interface for executing multiple RPCs. On the left, there are several input fields:

- RPC Type:** A radio button group with "Single RPC" (unchecked) and "Multiple RPCs" (checked). A red callout box labeled "Stop on error" points to the "Multiple RPCs" option.
- HTTP method:** POST
- Input data type:** XML
- Required output format:** Plain text
- RPC URL:** /rpc?stop-on-error
- Username:** lab
- Password:**
- Request body:** <get-interface-information><interface-name>ge-0/0/1</interface-name><terse></get-interface-information><get-alarm-information />

Below the request body is a "Submit" button.

The right side shows the **Response Body** containing the output of two RPCs:

```
--nwlrbbmqbhcdarz
Content-Type: text/plain; charset=utf-8

Interface          Admin Link Proto Local      Remote
ge-0/0/1           up    up      up       up

--nwlrbbmqbhcdarz
Content-Type: text/plain; charset=utf-8

No alarms currently active
--nwlrbbmqbhcdarz--
```

A red callout box labeled "RPC response" points to the first part of the response body, and another red callout box labeled "Multiple RPCs" points to the second part.

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER Worldwide Education Services

www.juniper.net | 20

Executing Multiple RPCs at Once

Executing multiple RPCs at once gives you the additional option of stopping an execution if it has an error. If the `stop_on_error` check-box is not selected, the script will report the error and continue executing.

Notice that you can still chose an HTTP method. To perform a Multiple RPC lookup, you must use POST. Using Get will return an error.

On the slide, you can see the `get-interface-information` and the `get-alarm-information` RPCs. You can also encode the RPCs as plain text.

The response body shows the output of both RPCs. The string "`--nwlrbbmqbhcdarz--`" is the string used to separate one RPC from the next.

Troubleshooting with Trace Options

- To trace the lighttpd web server

[edit system services]

```
lab@vMX-1# set rest traceoptions flag lighttpd
```

- To trace the juise plug-in

[edit system services]

```
lab@vMX-1# set rest traceoptions flag juise
```

- To trace both

[edit system services]

```
lab@vMX-1# set rest traceoptions flag all
```

- You can only have one flag set at a time

© 2017 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 21

Troubleshooting with Trace Options

To configure trace options for lighttpd, juise, or both, specify `set system services rest traceoptions flag`. Set flag to lighttpd, juise, or all. When you specify `traceoptions`, the command overwrites any previous trace option settings.

Summary

- In this content, we:

- Described the purpose of the Junos OS REST API
- Created REST API RPC queries
- Described the use of the REST API Explorer

We Discussed:

- The purpose of the Junos OS REST API,
- Creation of REST API RPC queries, and
- Described the use of the REST API Explorer.

Review Questions

1. What port does the REST API service listen on by default?
2. Is it permissible to use HTTP GET requests for a multiple RPC call?
3. What traceoption flag would you use to trace juice process errors?

© 2017 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

www.juniper.net | 23

Review Questions

- 1.
- 2.
- 3.

Lab: Junos REST API

- Use curl to create query the Junos REST API
- Use the REST API explorer to query the REST API

© 2017 Juniper Networks, Inc. All rights reserved.



Worldwide Education Services

www.juniper.net | 24

Lab: Implementing the Junos OS REST API

The slide lists the objectives for this lab.

Answers to Review Questions

1.

The REST API service listens on port 3000 by default.

2.

You must you POST requests for multiple RPC queries.

3.

You would want to use the `juise` flag to trace issue with the `juise` process.

Resources to Help You Learn More

Resource	URL
Pathfinder	http://pathfinder.juniper.net
Content Explorer	http://www.juniper.net/techpubs/content-applications/content-explorer
Feature Explorer	http://pathfinder.juniper.net/feature-explorer
Learning Bytes	www.juniper.net/learningbytes
Installation and configuration courses	www.juniper.net/courses
J-Net Forum	http://forums.juniper.net/t5/Training-Certification-and/bd-p/Training_and_Certification
Certification program	www.juniper.net/certification
Courses	http://www.juniper.net/training/technical_education
Translation tools	http://www.juniper.net/customers/support/#task

© 2017 Juniper Networks, Inc. All rights reserved.



www.juniper.net

Resources to Help You Learn More

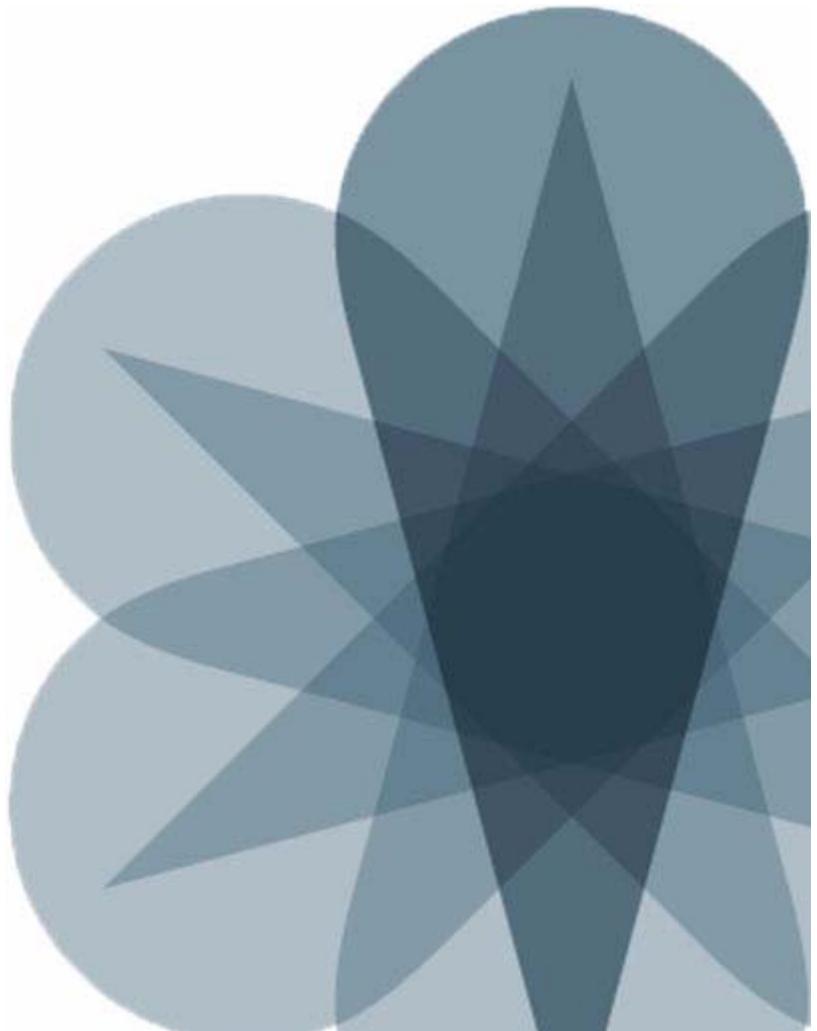
The slide lists online resources available to learn more about Juniper Networks and technology. These resources include the following sites:

- *Pathfinder*: An information experience hub that provides centralized product details.
- *Content Explorer*: Junos OS and ScreenOS software feature information to find the right software release and hardware platform for your network.
- *Feature Explorer*: Technical documentation for Junos OS-based products by product, task, and software release, and downloadable documentation PDFs.
- *Learning Bytes*: Concise tips and instructions on specific features and functions of Juniper technologies.
- *Installation and configuration courses*: Over 60 free Web-based training courses on product installation and configuration (just choose eLearning under Delivery Modality).
- *J-Net Forum*: Training, certification, and career topics to discuss with your peers.
- *Juniper Networks Certification Program*: Complete details on the certification program, including tracks, exam details, promotions, and how to get started.
- *Technical courses*: A complete list of instructor-led, hands-on courses and self-paced, eLearning courses.
- *Translation tools*: Several online translation tools to help simplify migration tasks.



Junos Platform Automation and DevOps

Appendix A: Zero Touch Provisioning (ZTP)



Objectives

- After successfully completing this content, you will be able to:
 - Explain the purpose and value of ZTP
 - Describe the components and operations of ZTP
 - Deploy a QFX5100 Series switch using ZTP

We Will Discuss:

- The purpose and value of ZTP;
- The components and operations of ZTP; and
- How to deploy a QFX5100 Series switch using ZTP.

Agenda: Zero Touch Provisioning

- Understanding Zero Touch Provisioning
- ZTP in Action: A Working Example

© 2014 Juniper Networks, Inc. All rights reserved.

 Worldwide Education Services

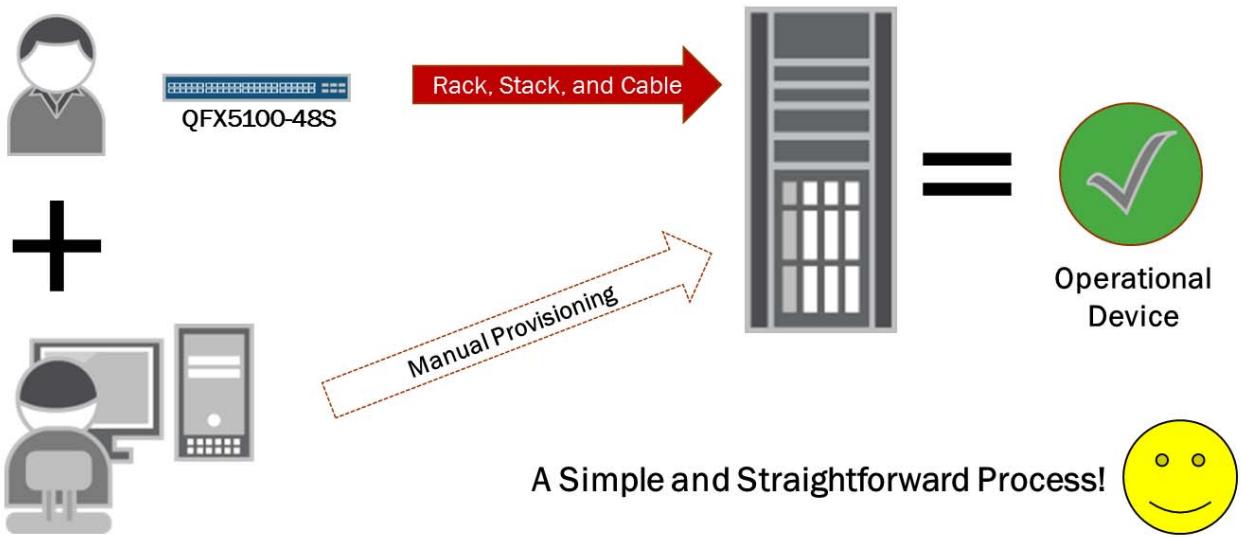
www.juniper.net | 3

Understanding Zero Touch Provisioning

The slide lists the topics we will discuss. We discuss the highlighted topic first.

Deploying a Switch

- When deploying a switch, you typically rack and cable the physical equipment and then perform the initial provisioning tasks to make the switch operational



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 4

Deploying a Switch

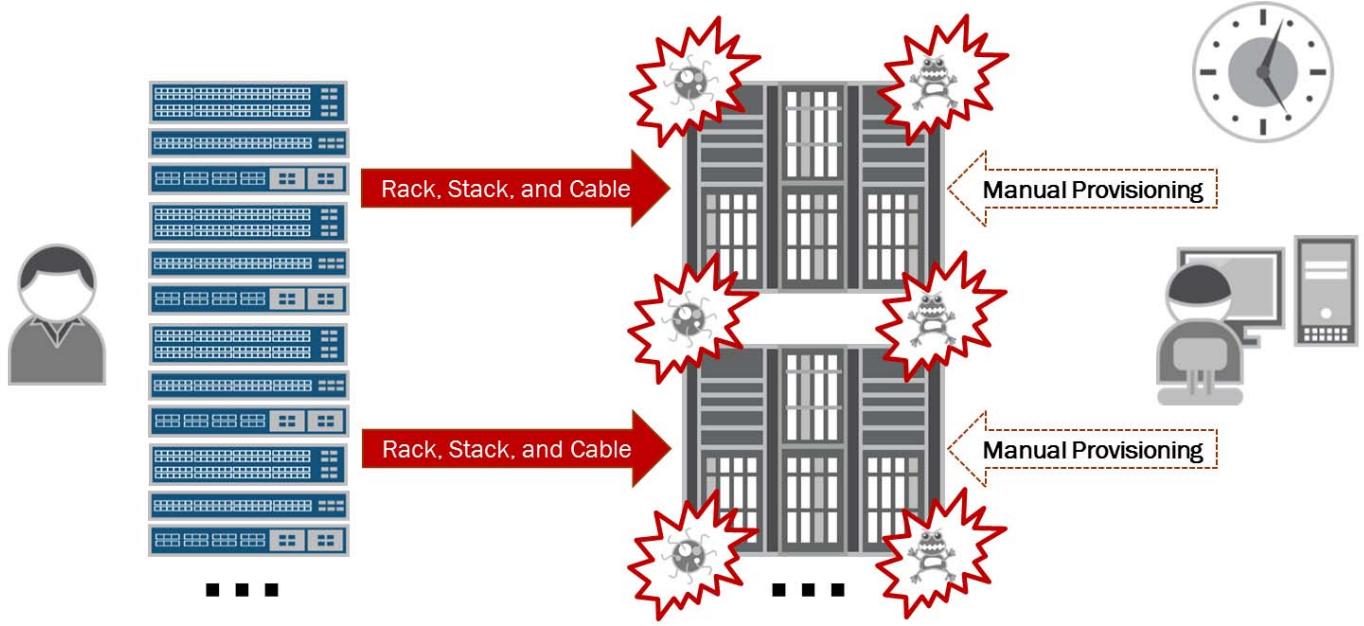
As you deploy a switch in your network, there are a few basic requirements you must meet before that switch provides any operational value. As illustrated on the slide you must physically rack each switch in its designated location and then connect it, using the required cables, with other devices on the network. Once the switch is racked and cabled, you then perform the provisioning tasks required to make the switch ready for network operations.

Some example provisioning tasks include adding the switch to its out-of-band (OOB) management network so it can be remotely managed using telnet or SSH as well as adding other configuration parameters to ensure the switch is a secure and functional participant on the network. Once the switch can communicate with other devices on the network, you may need upgrade its software to ensure it is running the desired version for your specific environment.

Once the basic installation and provisioning tasks are complete, the switch can become a value-added element on the network. At a quick glance this seems like a simple and straightforward process, right?

Complicating Things

- Deploying and manually provisioning dozens of switches can be time consuming and introduce issues



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 5

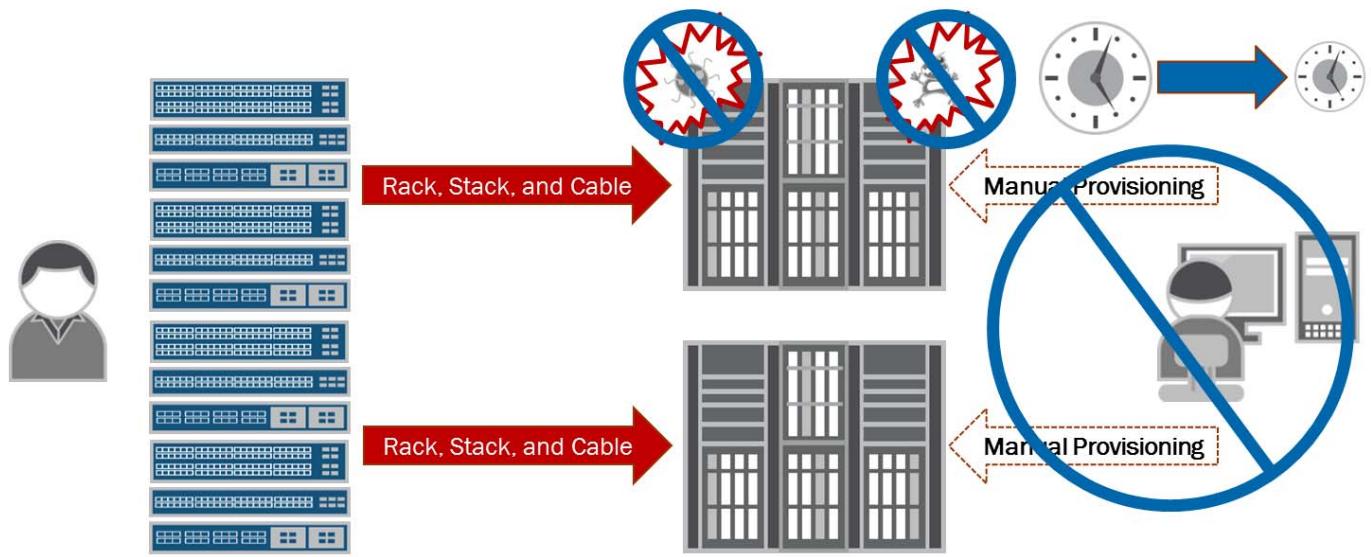
Complicating Things

To install and provision a single switch may not be too difficult for a smooth operator like yourself. However, if you are tasked with installing and provisioning dozens or even hundreds of switches things can get a bit more complicated. With an increased load of switches that must be properly racked, cabled, and provisioned, you will also see that the time required for such deployments also increases. You may also find that new and challenging issues are introduced because of improperly provisioned switches on your network.

Fortunately, as discussed on forthcoming slides, this situation can be simplified in some ways!

Making Things Easier

- Zero Touch Provisioning automates provisioning which can increase speed of deployment and provisioning and reduce configuration errors



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 6

Making Things Easier

In large scale deployments, where dozens or hundreds of switches are deployed, Zero Touch Provisioning (ZTP) can be used to automate and simplify some of the provisioning requirements. Using ZTP can save you time and, if done correctly, reduce many of the errors and issues often introduced when performing the same provisioning tasks manually on individual switches.

Note that while ZTP can offload some of the manual administrative load, it is really only part of an emerging approach to managing infrastructure resources through automation in the data center. In addition to the initial provisioning efforts, subsequent and ongoing modifications are required as network requirements change. The subsequent and ongoing modifications required on the switches can be consolidated and simplified through the use of PyEZ, Junos Space and the Network Director application. These subsequent modifications can also be automated using Puppet or Ansible.

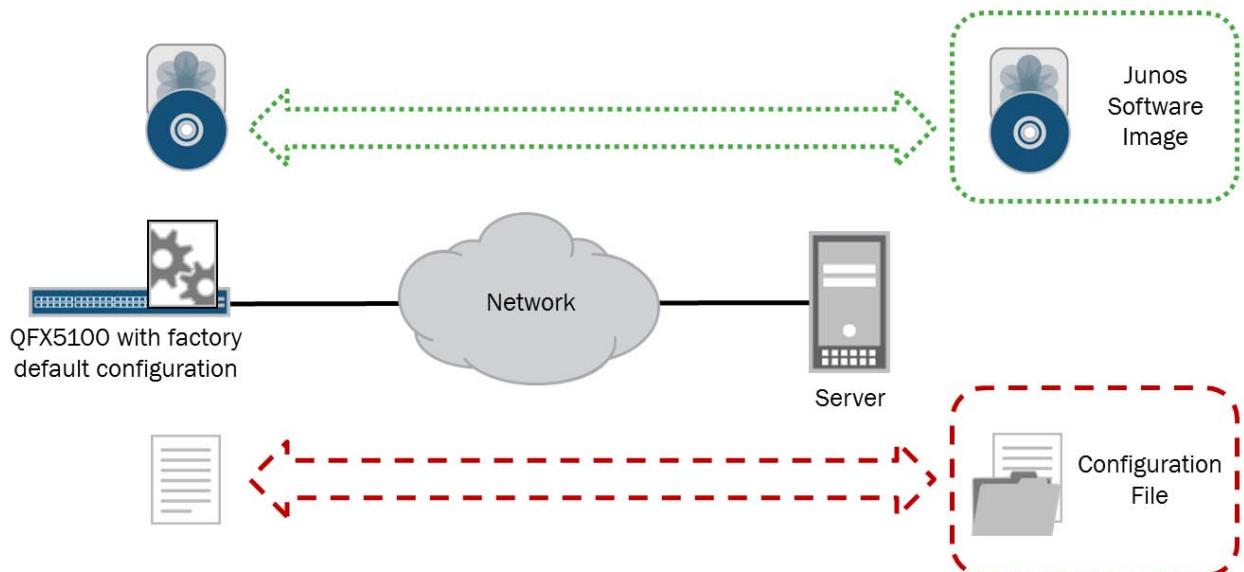
In addition to simplifying ongoing maintenance efforts through management applications, such as Network Director, or automation tools, such as Ansible, it is also becoming increasingly more common to orchestrate operations between infrastructure devices in data centers. OpenStack is emerging as the leading orchestration solution of all infrastructure devices.

Note that Puppet, Ansible, Salt, and OpenStack are supported on many Junos OS devices positioned for the data center, including the QFX5100 Series switches. For support details of your specific products, check the support documentation. Note that Puppet and OpenStack are outside the scope of this class.

We cover ZTP in more detail on the next slide and throughout the remainder of this chapter.

What Does It Do?

- ZTP automates day one provisioning tasks such as:
 - Loading the correct software image on all switches
 - Applying the desired configuration file on each switch



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

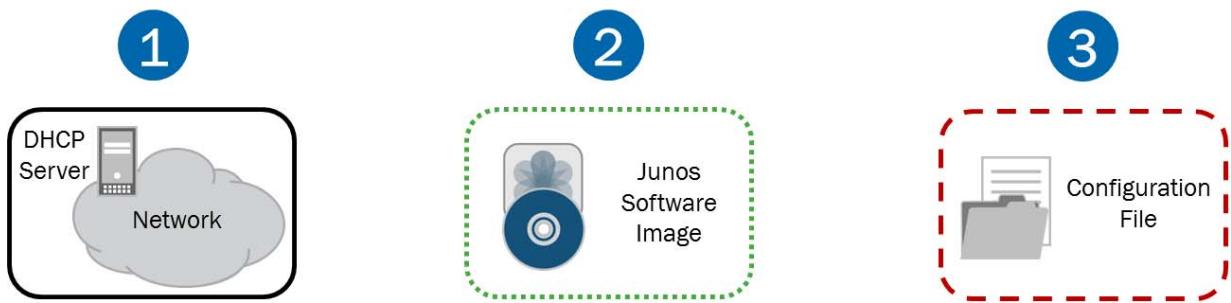
www.juniper.net | 7

What Does It Do?

As previously mentioned, ZTP automates day one provisioning tasks. The specific provisioning tasks automated through ZTP include loading and installing the correct software image on the devices running the Junos OS installed in your network and applying the desired configuration to each installed device. We describe the operations of ZTP on the next slides.

How Does It Work?

- When a switch boots with its factory default configuration, it does the following:
 1. Discovers a DHCP server and obtains an IP address and some other key details used to locate resources
 2. Verifies its software version is the desired version for the network and, if needed, retrieves and upgrades image
 3. Locates and applies its unique configuration from server



How Does It Work?

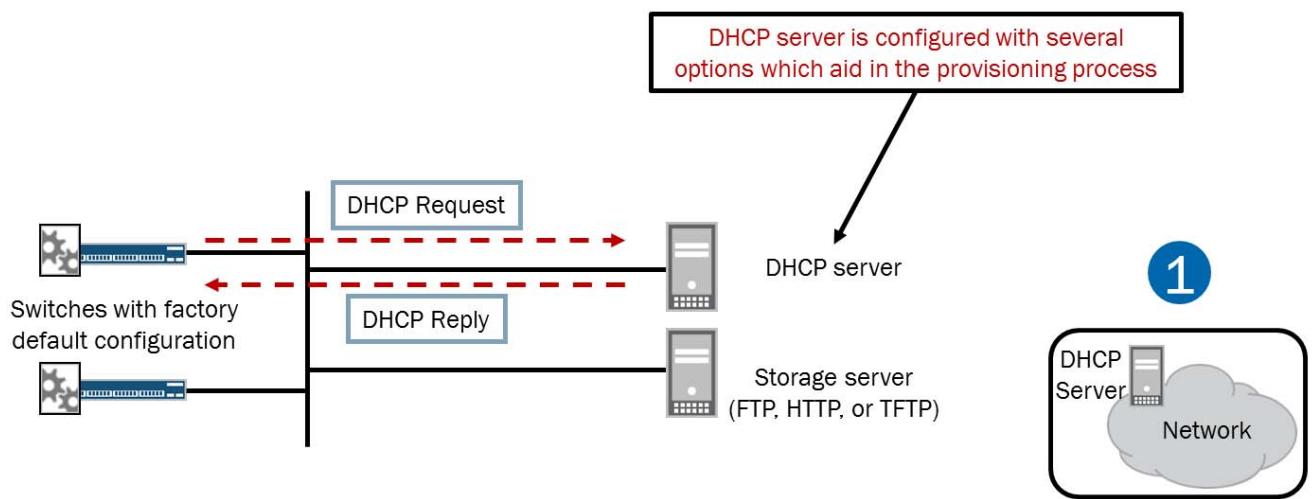
Once a switch, with its factory-default configuration, boots, the ZTP process takes over. The ZTP process does the following:

1. It discovers a DHCP server and obtains an IP address. In the process of obtaining an IP address from the DHCP server, the switch receives key details, by way of the DHCP options configured on the server, that inform the switch where provisioning resources are located and how the required files on those resources can be accessed.
2. Once a switch, undergoing provisioning through ZTP, is on the network and has received sufficient information from the DHCP server and the options it provided, it verifies the software version it is running against the version listed in the corresponding DHCP option. If the versions do not match, the image for the target version is obtained using the specified transfer method and the switch is upgraded.
3. In addition to performing the software verification and, if needed, an upgrade, the switch also locates and applies its designated configuration as provided by the administrator.

We look at some additional details involved in the illustrated ZTP steps on the next slides.

A Closer Look (1 of 3)

- Switches, with the factory default configuration, use DHCP to obtain an IP address and join the network
 - DHCP options are used to inform the switch about key details used during the provisioning process



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 9

A Closer Look, Part 1

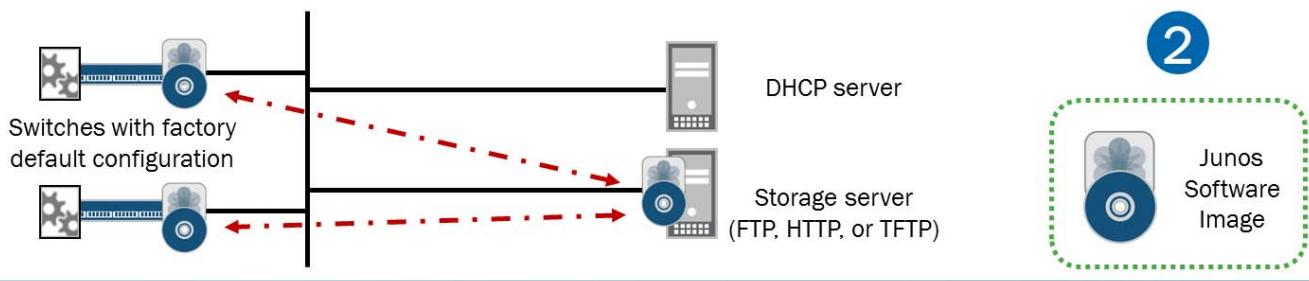
The first step of ZTP is to add the switch being provisioned to the network. The switch, in its factory-default state, functions as a DHCP client and, when booted, sends a DHCP request packet out its management Ethernet interface as well as all attached revenue ports. The management Ethernet interface, me0 on EX Series and most QFX Series and em0 on the QFX5100, is enabled as a DHCP client in the factory-default configuration. The factory-default configuration also includes a single integrated routing and bridging (IRB) interface enabled as a DHCP client. This IRB interface is assigned to the default VLAN as a Layer 3 interface along with all revenue ports, which are configured as Layer 2 access ports in the same default VLAN. The DHCP requests are sent from the switch out all physical interfaces that are connected and operational.

As long as a DHCP server is configured and accessible on either the management network or through a network accessible by one or more of the revenue ports, the DHCP request sent from the switch should make its way to the DHCP server. Once the DHCP server receives the request packet, it will respond with a DHCP reply offering its DHCP services. For the target DHCP server to facilitate a successful ZTP event, it must include DHCP options, which we will discuss on later slides, in its DHCP reply and offer packets.

The end objective of this step in the ZTP process is that the switch be added to the network and have sufficient information, through the DHCP options, to advance to the next steps discussed on the subsequent slides.

A Closer Look (2 of 3)

- Using DHCP Option 43 sub option 00 (or 04), DHCP server informs switches of software image name
 - If switch is already running the referenced image, no action is performed and the switch moves to next ZTP step
 - If switch is not running the referenced image, it downloads the image from the Storage server and is then upgraded
 - DHCP Option 43, sub option 03 informs switch of transfer mode
 - DHCP Option 150 (or 66) informs switch of Storage server's IP



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 10

A Closer Look, Part 2

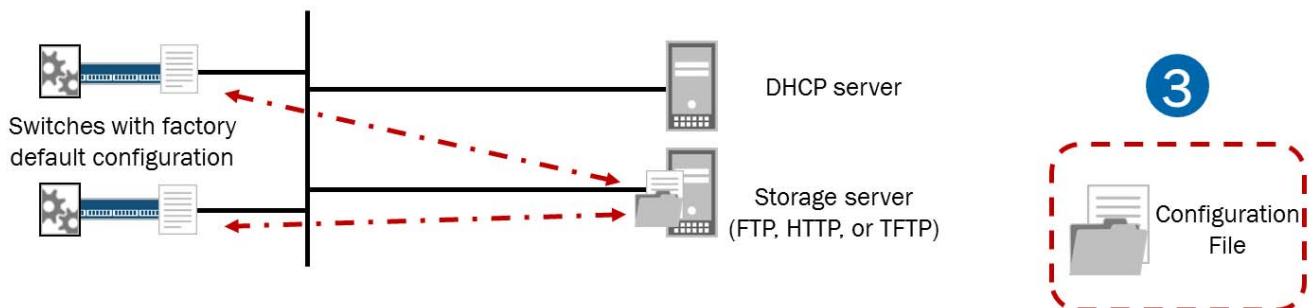
Once the switch has been added to the network and has received instructions through the DHCP options provided from the DHCP server, it can then move on to the next step in the ZTP process. The second step in the ZTP process is for the switch to verify, through the appropriate DHCP options, which software image it should be running. The switch determines the software image required for the network using DHCP Option 43 and sub option 00 (or alternatively sub option 04). If both sub options are defined on the DHCP server and a different image name is listed by each, the switch prefers the details included in sub option 00.

If the switch is already running the specified image, no immediate action is taken and the switch moves on to the next step in the ZTP process. If the switch is not running the specified image, the switch begins the process of downloading and installing the specified image. The switch evaluates other DHCP options or sub options to determine how and from where the specified image should be obtained. Specifically, DHCP option 43 sub option is used to inform the switch of the transfer mode (FTP, DHCP, or HTTP) and DHCP option 150 (or 66) is used to inform the switch of the storage server's IP address. If both DHCP option 150 and 66 are specified with different server IP addresses, DHCP option 150 is preferred.

Note that before the image is actually retrieved and the system is upgraded, step three, which is outlined on the next slide, is performed. Once the switch evaluates all details included in the DHCP options and determines which files, if any, are required, it then downloads the identified configuration file, which is part of step three, and then the image file. Once the required files are downloaded, it first performs the upgrade referenced on this slide and then applies the configuration file retrieved from the storage server. This download and operations sequence accommodates scenarios where the user wants to update the configuration file on the switch without also upgrading the software image on the switch.

A Closer Look (3 of 3)

- Using DHCP Option 43 sub option 01, DHCP server informs switch of configuration file name
 - Switch then retrieves and applies configuration file
 - DHCP Option 43, sub option 03 informs switch of transfer mode
 - DHCP Option 150 (or 66) informs switch of Storage server's IP
 - If no configuration file is referenced in DHCP Options, switch retains factory default settings



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS Worldwide Education Services

www.juniper.net | 11

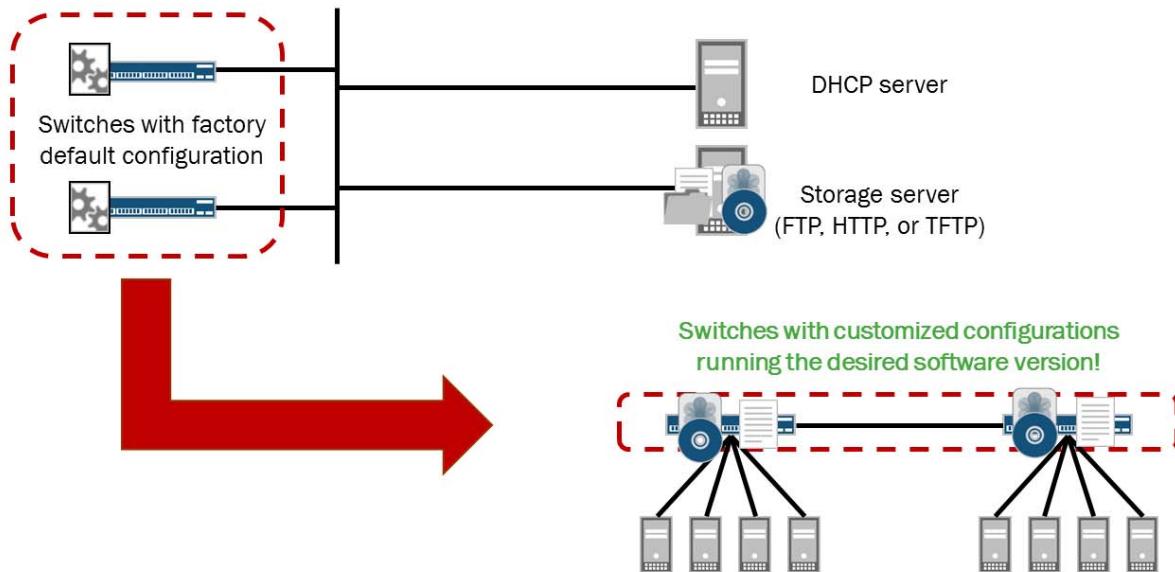
A Closer Look, Part 3

The last step, before the aforementioned download and application process takes place, is to identify if the switch should retrieve and apply any unique configuration file. As shown on the slide, DHCP option 43 sub option 01 is used to inform the switch of the target configuration file it should retrieve and apply. If present, the switch uses the information in DHCP option 43 sub option 01 along with the details learned in DHCP option 43, sub option 03 and DHCP option 150 (or 66) to construct a work order to retrieve and apply its designated configuration file. If no configuration file is referenced in the DHCP options, the switch retains its factory default settings.

As previously mentioned, the switch determines which files (software image and configuration file), if any, are required before it attempts to establish any connection with the storage server and download the identified files. If both a software image and configuration file are required, they are both downloaded. Once the files are downloaded, the switch performs the upgrade illustrated as part of step two and then, once the upgrade is complete, applies its newly retrieved configuration file.

The End Result!

- Once ZTP processing finishes, the switches are then provisioned with the desired software image and a functional configuration used for network operations!



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

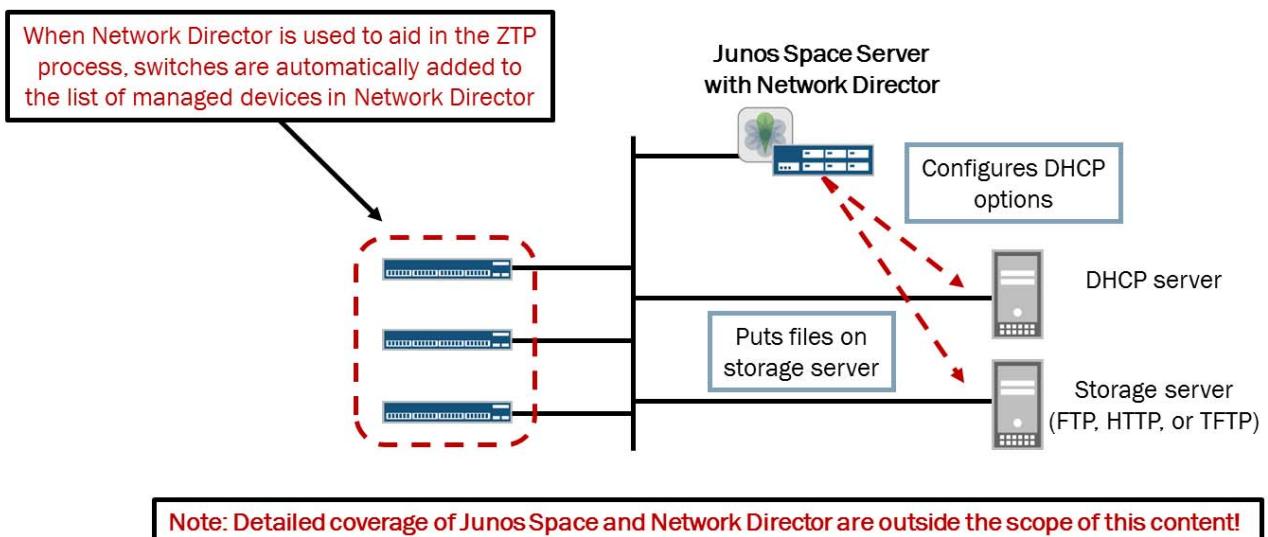
www.juniper.net | 12

The End Result!

Once ZTP processing finishes, the switches being provisioned should be running the desired software image and an updated configuration making them useful and contributing participants on the network. We describe some of the functional considerations of ZTP and look at a working example in the next section of this chapter.

Network Director and ZTP

- Network Director can be used to:
 - Provision the DHCP server with the required DHCP Options
 - Place Junos images and configuration files on storage server



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 13

Network Director and ZTP

In environments that use Junos Space and the Network Director application, certain administrative aspects of ZTP can be simplified. Specifically you can provision the DHCP server with the required DHCP options it uses to inform the switches being provisioned of the provisioning details discussed earlier. You can also use the Network Director application to manage your image and configuration file repository located on the storage server in your network. Note that, depending on the software version used, switches provisioned through ZTP, where the DHCP server's configuration is created with the help of Network Director, are automatically added to the list of managed devices within the Network Director application.

For more details regarding Junos Space and the Network Director application, please refer to the technical documentation available at www.juniper.net or consider attending the training courses that cover Junos Space and Network Director.

Agenda: Zero Touch Provisioning

- Understanding Zero Touch Provisioning
- ZTP in Action: A Working Example

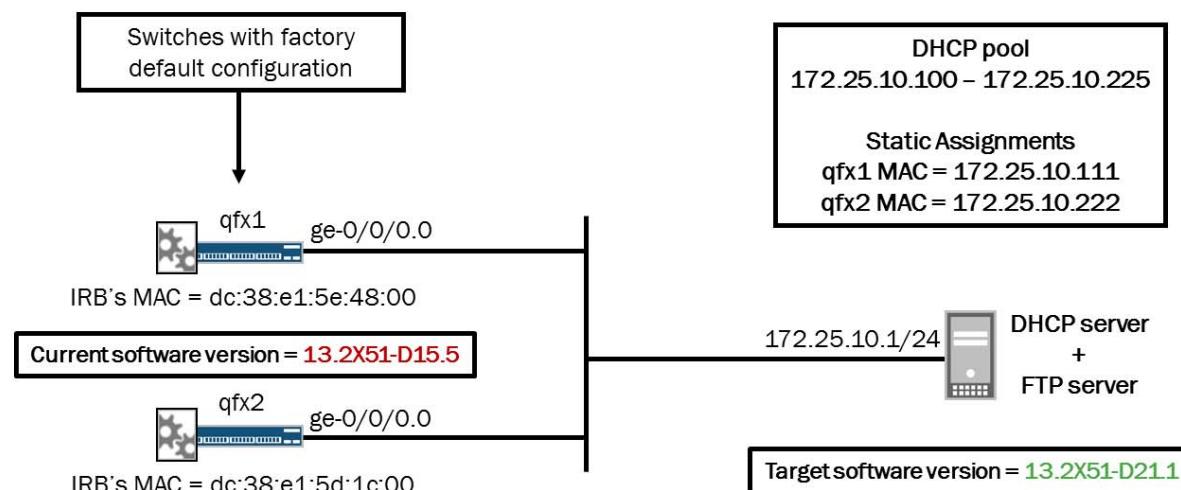
ZTP in Action: A Working Example

The slide highlights the topic we discuss next.

Sample Objectives and Topology

- Case study objectives:

- Use ZTP to provision new QFX5100 Series switches and ensure the new switches are running the desired software version and have their respective configurations applied



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER
NETWORKS

Worldwide Education Services

www.juniper.net | 15

Sample Objectives and Topology

This slide introduces the objectives and topology we use to illustrate a working example of ZTP. In this example we have a pair of QFX5100 Series switches running a different software version than the desired software version for this network. As shown on the slide, we have a single server device functioning as the DHCP server and the FTP server. This server is accessible from the QFX5100 Series switches through their respective ge-0/0/0.0 interfaces. The server is participating on and serving addresses from the 172.25.10.0/24 subnet.

The qfx1 switch is assigned the 172.25.10.111 IP address while qfx2 is assigned the 172.25.10.222 IP address. These IP address assignments are statically defined on the DHCP server and are based on the switches' IRB MAC addresses. Predefined configurations for each switch, again based on the target switches' IRB MAC address, are loaded on the server and are accessible through FTP, which is the selected transfer method used in this example. Note that, once the device-specific configuration files are retrieved and applied, the IP addresses assigned to the switches through DHCP will eventually change to addresses outside the DHCP address pool during the ZTP process.

Note

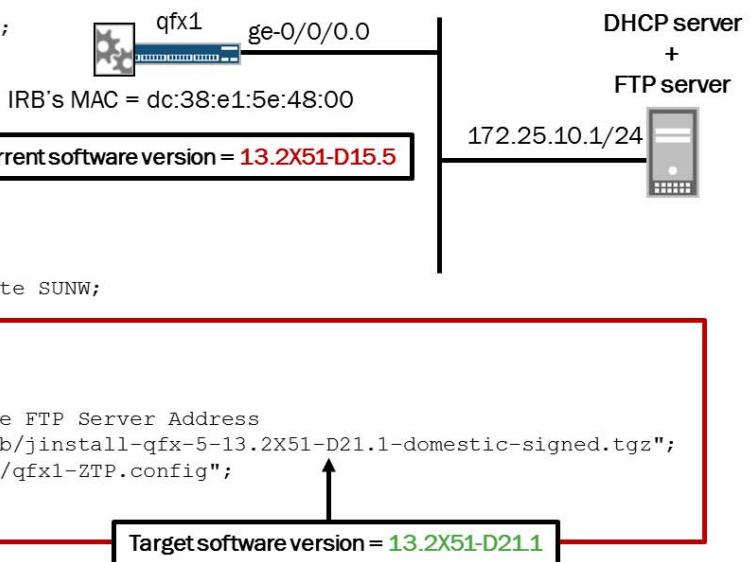
The illustrated example is not, as you may have realized, truly a zero touch provisioning approach. In this example, the administrator must retrieve the MAC address of each switch being provisioned, which requires a touch. There are other ways to work through ZTP that remove the need for this specific touch. You can, for example, automate the learning of the MAC addresses and subsequently automate the update of the DHCP configuration used during the provisioning operation. You could also create a provisioning infrastructure where each switch, requiring automated provisioning, is isolated and therefore can be predictably provisioned based on its position in the provisioning infrastructure network. Regardless of your selected approach, there is some administrative trade-off involved.

Sample DHCP Server Configuration

```
[root@K01-LP ~]# cat /etc/dhcpd.conf
...
option option-150 code 150 = ip-address;
subnet 172.25.10.0 netmask 255.255.255.0 {
    option routers             172.25.10.1;
    option subnet-mask         255.255.255.0;
    default-lease-time 21600;
    max-lease-time 43200;
    range 172.25.10.100 172.25.10.225;
}
}

option space SUNW;
option SUNW.server-image code 4 = text;
option SUNW.server-image code 0 = text;
option SUNW.server-file code 1 = text;
option SUNW.image-type code 2 = text;
option SUNW.transfer-mode code 3 = text;
option SUNW-encapsulation code 43 = encapsulate SUNW;
```

```
host qfx1 {
    hardware ethernet dc:38:e1:5e:48:00;
    fixed-address 172.25.10.111;
    option option-150 172.25.10.1;# Define FTP Server Address
    option SUNW.server-image "/var/ftp/pub/jinstall-qfx-5-13.2X51-D21.1-domestic-signed.tgz";
    option SUNW.server-file "/var/ftp/pub/qfx1-ZTP.config";
}
```

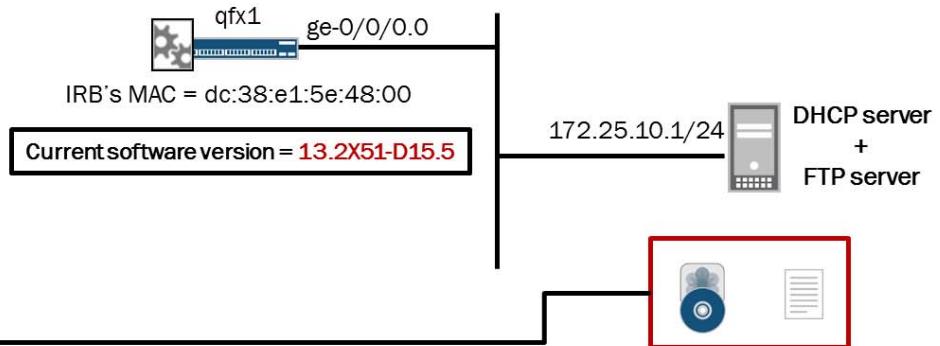


Sample DHCP Server Configuration

This slide illustrates a sample DHCP server configuration used to assign the desired IP addresses to the QFX Series switches and to properly distribute the ZTP provisioning details through the appropriate DHCP options and sub options.

Sample FTP Server Configuration

```
[root@K01-LP ~]# cat /etc/vsftpd/vsftpd.conf
#local_enable=YES
tcp_wrappers=YES
pam_service_name=vsftpd
dirlist_enable=YES
force_dot_files=YES
listen=YES
listen_port=21
chmod_enable=YES
dirmessage_enable=YES
ftp_username=ftp
anon_max_rate=0
anonymous_enable=YES
anon_mkdir_write_enable=YES
anon_other_write_enable=YES
anon_root=/var/ftp/pub
anon_upload_enable=YES
write_enable=YES
xferlog_enable=YES
syslog_enable=YES
data_connection_timeout=120
idle_session_timeout=120
pasv_min_port=5920
pasv_max_port=5920
ascii_upload_enable=YES
ascii_download_enable=YES
```



Basic Checklist:



- Service must be enabled and listening on port 21
- Anonymous FTP access must be enabled
- Files must be stored in the root directory

Sample FTP Server Configuration

This slide illustrates a sample FTP server configuration used to accept incoming sessions on port 21 and permit anonymous FTP access, which is required for ZTP. In this sample configuration you can also see where the files are stored that will be retrieved by FTP clients, which in our case are the QFX Series switches. You must make sure the software image and configuration files referenced in the associated DHCP options and sub options are stored in the referenced directory. If the required files are not stored in the directory, the operation will fail and an error message will be shown on the console. You must also ensure that the files have sufficient read/write permissions so they can be retrieved.

You can verify the files are in the correct directory and have sufficient read/write permissions as shown in the following output:

```
[root@server ~]# ls -l /var/ftp/pub
total 807468
-rwxr-xr-x 1 root root 414176908 Aug 22 08:38
jinstall-qfx-5-13.2X51-D15.5-domestic-signed.tgz
-rw-rw-rw- 1 root root 411839279 Aug 20 12:35
jinstall-qfx-5-13.2X51-D21.1-domestic-signed.tgz
-rwxrwxrwx 1 root root 1772 Aug 22 12:52 qfx1-ZTP.config
-rwxrwxrwx 1 root root 1772 Aug 22 12:51 qfx2-ZTP.config
```

Verifying Services and Communications

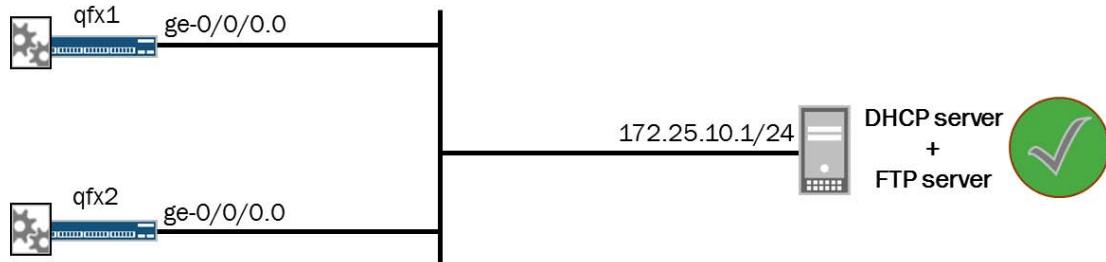
```
[root@K01-LP ~]# service dhcpcd status  
dhcpcd (pid 14847) is running...
```

```
[root@K01-LP ~]# service vsftpd status  
vsftpd (pid 3094) is running...
```

```
[root@K01-LP ~]# ftp 172.25.10.1
Connected to 172.25.10.1.
530 Please login with USER and PASS.
KERBEROS V4 rejected as an authentication type
Name (172.25.10.1:root): anonymous
331 Please specify the password. ←
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

- Verify that both services are running

FTP Server should allow incoming FTP sessions from the anonymous user and no password.



© 2014 Juniper Networks, Inc. All rights reserved.

JUNIPER Worldwide Education Services

www.juniper.net | 18

Verifying Services and Communications

Once the DHCP and FTP server configurations are properly applied, you must ensure that the associated services are running on the server. The sample output on the slide illustrates this verification step along with a simple test to ensure the FTP server accepts incoming FTP sessions from the anonymous user.

Zeroizing the Configuration

```

{master:0}
root> request system zeroize
warning: System will be rebooted and may not boot without configuration
Erase all data, including configuration and log files? [yes,no] (no) yes
...
unloading fpga driver
unloading host-dev
Shutting down ACPI
Rebooting...
...
Amnesiac (ttyd0)
login: root
System boots in Amnesiac mode and
expects root login without a password.

--- JUNOS 13.2X51-D15.5 built 2014-03-06 10:05:33 UTC
root@RE:0% cli
{master:0}
root>

```

**Current configuration and log files
are erased when system is zeroized**

**System boots in Amnesiac mode and
expects root login without a password.**

The diagram consists of two red boxes. The top box contains the command 'request system zeroize' and its warning message. An arrow points upwards from this box to another red box containing the text 'Current configuration and log files are erased when system is zeroized'. Another arrow points downwards from this second box to a third red box containing the text 'System boots in Amnesiac mode and expects root login without a password.' A final arrow points leftwards from this third box to the terminal session where the user logs in as 'root'.

Zeroizing the Configuration

By default, EX and QFX Series switches should support ZTP with their factory-default configurations and settings. If the configuration file has been changed because of a previous deployment, for example, you can zeroize the system, which essentially makes it able to participate in the ZTP process. Using the **request system zeroize** command, illustrated on the slide, deletes the current configuration along with the stored rollback configurations. This command also removes all log files on the system and restores some other default settings. Because this operation removes stored information, you must confirm the operation by typing in **yes**, as illustrated on the slide.

Note that if the configuration files have been modified and other default settings have been changed, you must perform the zeroize operation and cannot simply load the factory-default configuration using the **load factory-default** command with a **commit** while in configuration mode.

Once you zeroize a switch, the system boots up in the Amnesiac mode and is accessible using the root login without a password. Note that this default state should change by the end of the ZTP process, at which time any further login attempts will require a known username and password as specified in the configuration file loaded during the ZTP process.

The Zeroized Configuration (1 of 4)

```
{master:0}
root> show configuration | no-more
system {
...
    commit {
        factory-settings {
            reset-virtual-chassis-configuration;
            reset-chassis-lcd-menu;
        }
    }
    processes {
        dhcp-service {
            traceoptions {
                file dhcp_logfile size 10m;
                level all;
                flag all;
            }
        }
    }
...
## Warning: missing mandatory statement(s): 'root-authentication'
}
chassis {
    auto-image-upgrade;
}
```



An abundance of information logged that may be useful for troubleshooting issues with the initial ZTP phase.

Triggers ZTP process. Note that if this statement is deleted, the ZTP process stops.

The Zeroized Configuration, Part 1

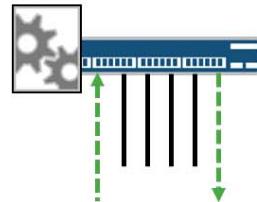
This and the next several slides illustrates some key parts of the zeroized configuration.

The Zeroized Configuration (2 of 4)

```

interfaces {
    ge-0/0/0 {
        unit 0 {
            family ethernet-switching {
                vlan {
                    members default;
                }
                storm-control default;
            }
        }
    }
    xe-0/0/0 {
        unit 0 {
            family ethernet-switching {
                vlan {
                    members default;
                }
                storm-control default;
            }
        }
    }
...
}

```



All interface combinations are defined as Layer 2 access ports and associated with the default VLAN

The Zeroized Configuration, Part 2

This slide illustrates some key parts of the zeroized configuration.

The Zeroized Configuration (3 of 4)

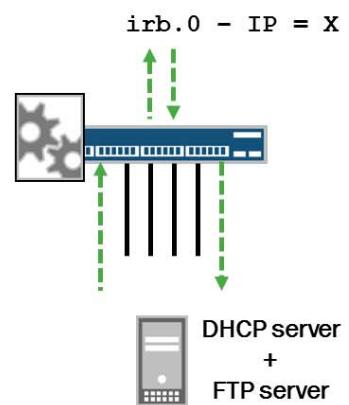
```

interfaces {
...
    irb {
        unit 0 {
            family inet {
                dhcp {
                    vendor-id Juniper-qfx5100-48s-6q;
                }
            }
        }
    vme {
        unit 0 {
            family inet {
                dhcp {
                    vendor-id Juniper-qfx5100-48s-6q;
                }
            }
        }
    }
}

```

Note: The vendor-id differs between switch models and is used to provide the DHCP server vendor and model specific details.

The **irb** and **vme** interfaces are configured for Layer 3 operations and as DHCP clients



The Zeroized Configuration, Part 3

This slide illustrates some key parts of the zeroized configuration.

The Zeroized Configuration (4 of 4)

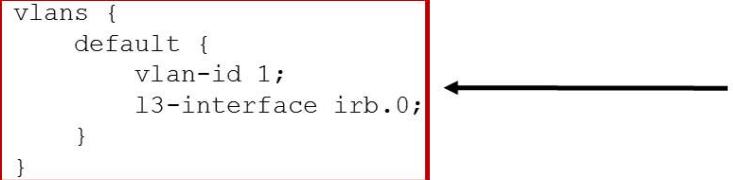
```

...
protocols {
    lldp {
        interface all;
    }
    lldp-med {
        interface all;
    }
    igmp-snooping {
        vlan default;
    }
    rstp {
        interface ge-0/0/0;
        interface xe-0/0/0;
    ...
}
}

vlans {
    default {
        vlan-id 1;
        13-interface irb.0;
    }
}

```

The **irb.0** interface and all revenue ports are associated with the default VLAN.



The Zeroized Configuration, Part 4

This slide illustrates some key parts of the zeroized configuration.

Monitoring Operations (1 of 3)

```
{master:0}
root>
Auto Image Upgrade: DHCP Client Bound interfaces:
Auto Image Upgrade: DHCP Client Unbound interfaces: irb.0 vme.0

Auto Image Upgrade: No DHCP Client in bound state, reset all enabled DHCP clients
Auto Image Upgrade: DHCP Client State Reset: irb.0 vme.0

Auto Image Upgrade: DHCP Client Bound interfaces:
Auto Image Upgrade: DHCP Client Unbound interfaces: irb.0 vme.0
Auto Image Upgrade: No DHCP Client in bound state, reset all enabled DHCP clients
Auto Image Upgrade: DHCP Client State Reset: irb.0 vme.0
Auto Image Upgrade: DHCP Options for client interface irb.0:
ConfigFile: /var/ftp/pub/qfx1-ZTP.config ImageFile: /var/ftp/pub/jinstall-qfx-5-
13.2X51-D21.1-domestic-signed.tgz Gateway: 172.25.10.1 File Server: 172.25.10.1
Options state: All options set
Auto Image Upgrade: DHCP Client Bound interfaces: irb.0
Auto Image Upgrade: DHCP Client Unbound interfaces: vme.0 ge-0/0/0.0 et-0/0/48.0 ...
Auto Image Upgrade: Active on client interface: irb.0
```

...

Note: ZTP operations can be monitored through the console of the device being provisioned.

Monitoring Operations, Part 1

Once the switch boots with the zeroized configuration you can monitor the ZTP operations using the console connection. This and the next slides illustrate some of the highlights of the ZTP process monitored through the console.

Monitoring Operations (2 of 3)

```
...
Auto Image Upgrade: Interface:: "irb"
Auto Image Upgrade: Server:: "172.25.10.1"
Auto Image Upgrade: Image File:: "jinstall-qfx-5-13.2X51-D21.1-domestic-signed.tgz"
Auto Image Upgrade: Config File:: "qfx1-ZTP.config"
Auto Image Upgrade: Gateway:: "172.25.10.1"
Auto Image Upgrade: Protocol:: "ftp"
Auto Image Upgrade: Start fetching qfx1-ZTP.config file from server 172.25.10.1
through irb using ftp
Auto Image Upgrade: File qfx1-ZTP.config fetched from server 172.25.10.1 through irb
Auto Image Upgrade: Start fetching jinstall-qfx-5-13.2X51-D21.1-domestic-signed.tgz
file from server 172.25.10.1 through irb using ftp

Auto Image Upgrade: File jinstall-qfx-5-13.2X51-D21.1-domestic-signed.tgz fetched
from server 172.25.10.1 through irb
Auto Image Upgrade: To install /var/tmp/jinstall-qfx-5-13.2X51-D21.1-domestic-
signed.tgz image fetched from server 172.25.10.1 through irb
```

WARNING!!! On successful image installation, system will reboot automatically

Auto Image Upgrade: Installation of /var/tmp/jinstall-qfx-5-13.2X51-D21.1-domestic-
signed.tgz image fetched from server 172.25.10.1 through irb is done, proceeding for
reboot of system

...

Monitoring Operations, Part 2

This slide illustrates some of the highlights of the ZTP process monitored through the console.

Monitoring Operations (3 of 3)

```
...  
Broadcast Message from root@  
    (no tty) at 20:41 UTC...  
Auto image Upgrade: Stopped  
*** FINAL System shutdown message from root@ ***
```

```
System going down IMMEDIATELY
```

```
...TRIMMED...
```

```
Amnesiac (ttyd0)  
  
login: lab  
Password:  
  
--- JUNOS 13.2X51-D21.1 built 2014-05-29 11:41:16 UTC  
{master:0}  
lab@qfx1-ZTP> show version  
fpc0:  
-----  
Hostname: qfx1-ZTP  
Model: qfx5100-48s-6q  
JUNOS Base OS Software Suite [13.2X51-D21.1]
```

Evidence of a successful ZTP event:
Login is now required, hostname has changed, and switch is running target software version.



Monitoring Operations, Part 3

This slide illustrates some of the highlights of the ZTP process monitored through the console.

Summary

- In this content, we:

- Explained the purpose and value of ZTP
- Described the components and operations of ZTP
- Deployed a QFX5100 Series switch using ZTP

We Discussed:

- The purpose and value of ZTP;
- The components and operations of ZTP; and
- How to deploy a QFX5100 Series switch using ZTP.

Review Questions

1. Which two provisioning tasks does ZTP automate?
2. What is the role of DHCP in the ZTP processing?
3. Which three transfer methods does ZTP support?

Review Questions

- 1.
- 2.
- 3.

Answers to Review Questions

1.

ZTP loads a predefined configuration and performs a software upgrade, if needed. The predefined configurations along with the software image used for upgrade operations are stored on a server that is reachable from the switch being provisioned.

2.

Before a switch being provisioned through ZTP can retrieve its configuration or a software image from a storage server on the network, the switch must be able to communicate on the network, which requires an IP address. Along with being able to communicate on the network, the switch must receive instructions on how to retrieve the designated files. The instructions indicating how the switch is to retrieve the designated files and from where they should be retrieved are relayed to the switch through DHCP options during the IP acquisition process.

3.

One of the DHCP options communicated to the switch during the IP acquisition process informs the switch of the file transfer method. The three file transfer methods that can be used by ZTP are TFTP, FTP, and HTTP. If no transfer method is specified, TFTP is used by default.

Acronym List

API	application programming interface
AS	autonomous system
BGP	Border Gateway Protocol
CLI	command-line interface
CNAME	canonical name
DNS	Domain Name System
DSL	Domain Specific Language
FQDN	fully qualified domain name
GUI	graphical user interface
iBGP	internal BGP
IGP	interior gateway protocol
IPv6	IP version 6
ISO	International Organization for Standardization
ISP	Internet service provider
lo0	loopback interface
NOC	network operations center
op	operation script
OSI	Open Systems Interconnection
pvm	Python virtual machine
pypi	Python Package Index
RPC	remote procedure call
scp	secure copy
SLAX	Stylesheet Language Alternative Syntax
UTC	Coordinated Universal Time
VRRP	Virtual Router Redundancy Protocol
XML	Extensible Markup Language
XPath	Extensible Markup Path Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

Corporate and Sales Headquarters

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or 408.745.2000
Fax: 408.745.2100
www.juniper.net

APAC and EMEA Headquarters

Juniper Networks International B.V.
Boeing Avenue 240
1110 PZ SCHIPHOL-RIJK
Amsterdam, The Netherlands
Phone: 31.0.207.125.700
Fax: 31.0.207.125.701

Copyright 2017

Juniper Networks, Inc. All rights reserved.
Juniper Networks, the Juniper Networks logo, Junos, NetScreen, and ScreenOS
are registered trademarks of Juniper Networks, Inc. in the United States and
other countries. All other trademarks, services marks, registered marks, or
registered services marks are the property of their respective owners. Juniper
Networks assumes no responsibility for any inaccuracies in this document.
Juniper Networks reserves the right to change, modify, transfer, or otherwise
revise this publication without notice.

❖ Printed on recycled paper