JNCIA

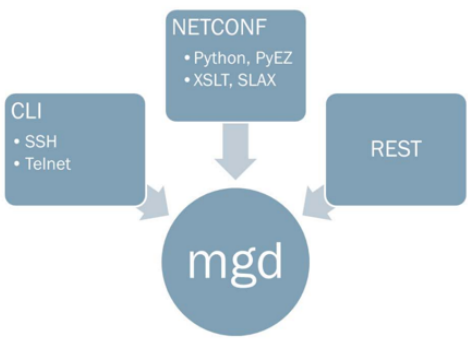
Course Introduction

- Introduction to Junos Platform Automation and Devops(IJAUT)
  - Course
    - Basic Devops Practices
    - Junos APIs
    - NETCONF
    - Python
    - PyEZ
    - Ansible
    - REST API
    - XML
    - JSON
    - YAML
  - Versions
    - Junos OS Release 18.1R1
    - Junos PyEZ 2.1
    - Ansible 2.5
  - Objectives(Mục tiêu)
    - Get to know another
    - Identify the objectives(xđ mục tiêu), prerequisites(đk tiên quyết), facilities(cơ sở), materials used during this course (tài liệu sd)
    - Identify additional education services courses at juniper networks(xđ các khoá học bổ sung)
    - Describe the Juniper Networks Certification Program
  - Prerequisites
    - Intermediate-level networking knowledge and an understanding of the Open System Interconnection (OSI) model and the TCP/IP protocol suite
    - Familiarity XML basics and have introductory knowledge the Python programing language

- Attend the Introduction to the Junos Operating System (IJOS) course prior (trước) to attending this class
- High-level understanding of object-oriented programing is a plus bust not a requirement

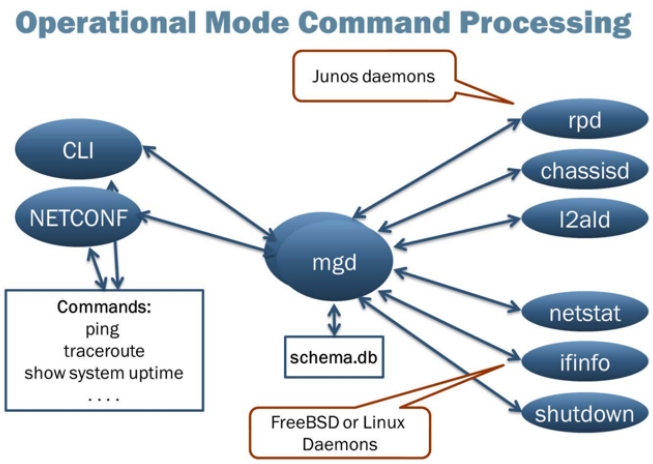
Junos Automation Architecture and Overview

- Objectives
  - Describe the Junos Architecture and Automation UI
  - Explain the role gRPC, NETCONF and REST in Junos Automation
  - Identify the languages, frameworks, management suites and tools commonly used in automating Junos
- Why Automate?
  - Configuration Management
  - Cross Platform Configuration(Cấu hình nền tảng chéo)
  - Enforce Consistency(Thi hành nhất quán)
  - Improve Security
  - Improve Efficiency
  - Increase Visibility(Tăng khả năng hiển thị)
  - Cut Costs
- Junos mgd-Based Automation
  - The Junos Processes
    - Junos supports many forms of automation
    - Junos automation ultimately goes through the mgd or jsd processes
  - The mgd(Management process)
    - manages CLI, NETCONF and REST connection

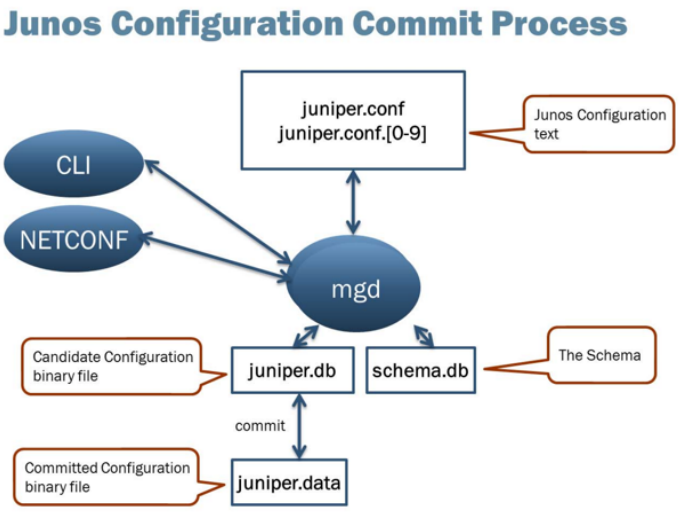


    - The Junos UI (User Infrastructure)

- Summary
  - Database: contains all configuration data
  - Schema: Defines the layout of database, as well as all JUNOS commands
  - mgd: Manages user input, command validation and configuration changes
  - CLI: Forwards terminal input to mgd, handles command completion and formats command output
- The Junos DDL(data definition language) and ODL(output definition language) Languages and their Historical Context
  - It's language the core of the Junos XML API and the UI
  - The Junos schema is implemented using the DDL and ODL languages
- The Junos OS Operational Command Processing

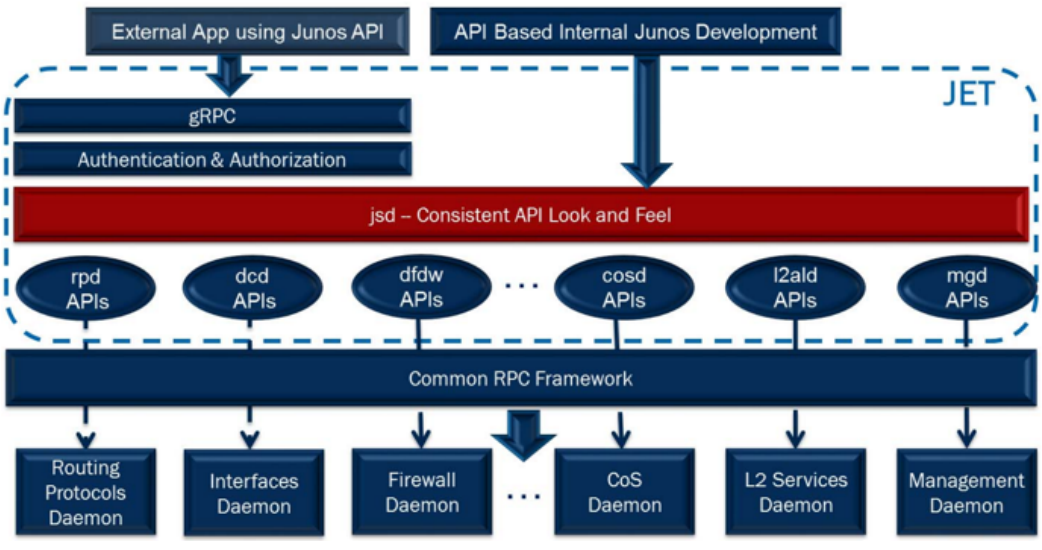


- The Junos OS Configuration Commit Process



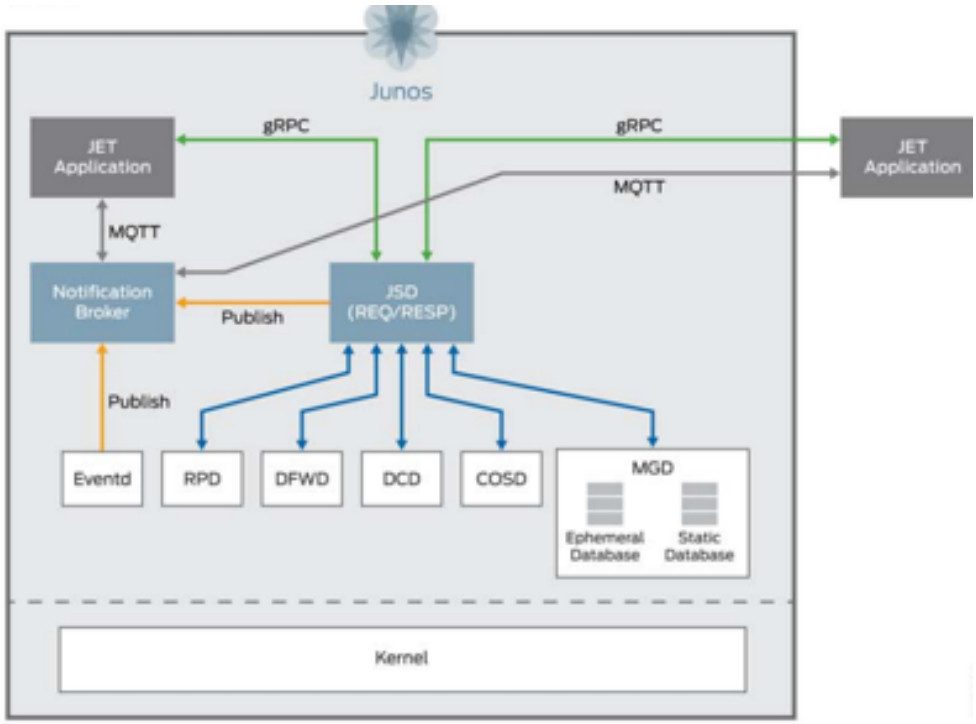
- Junos CLI

        - Console Connections
        - Telnet
        - SSH
    - Junos NETCONF(network configuration protocol) Connections
        - JUNOScript came first
        - NETCONF is the IETF standard
        - NETCONF sessions carry automation data for: XSLT, SLAX, Python, PyEZ, Other languages
    - Junos REST(representational state transfer-dịch chuyển trạng thái tượng trưng) API
        - REST is popular non-network specific Web protocol
        - REST API calls initially received by micro web server
        - Functionality limited due to limitation of REST and HTTP
- Junos jsd-Based Automation
    - The JET(Juniper Extension Toolkit) Service Process
        - The jsd exposes(hiển thị) internal APIs for automation



    - Jet Architecture

- The jsd and MQTT Notification Broker are separate(riêng biệt) services that run on different TCP port numbers



- Automation Languages, Libraries, and Frameworks
  - Automation Architecture Review
    - Calls to JET API go through gRPC connection to jsd
    - Calls to XML API and YANG go through NETCONF to mgd
    - REST API calls go through REST indirectly(gián tiếp) to mgd
  - Programming Languages
    - Connection type dictates(ra lệnh) programming language

NETCONF	gRPC	REST
• Python / PyEZ • Perl • Ruby • Java • SLAX / XSLT • Others …	• Python • Others Possible – Need to Compile IDL	• Languages that can do HTTP GET and POST

      - NETCONF Libraries

- make programming easier

API Language	Distribution Mode	Maturity	Support	Additional Notes
Ruby	Open Source	Most popular, 3200+ downloads	Open Source	Best ease of installation, packed with features, but has limited dependencies and active support
Java	Juniper website and Github	Already in use by enterprise customers	JTAC	Easy installation, simple start-up, single .jar file to use with zero dependencies
Python	Open Source	Based on a popular open source client	Open Source	Most popular scripting language
Perl	Juniper website	Oldest API; more difficult installation	JTAC	API installation needs simplification

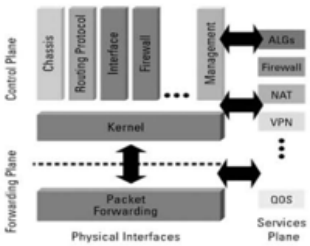
  - Python On-Box
    - Python 2.7 on the box in Junos 16.1 and later
    - Everything that is possible with SLAX can be done with Python
    - Part of core Junos image
  - Python Off-Box
    - Also uses Python 2.7 and PyEZ
    - Python with PyEZ used with 11.4 and later
  - PyEZ
    - A powerful, easy to learn microframework
      - Python framework with easy learning curve(đường cong)
      - Works with any Junos device running 11.4 or later
    - Run shell scripts
    - Junos automation
  - Ruby and RubyEZ
    - Ruby
      - True object-orient language
      - Like Python, scripts are interpreted(diễn giải), not complied
      - No on-box implementation
    - RubyEZ
      - provides classes and methods to allow Ruby programs to control Junos devices

- Data is formatted in XML, transported via NETCONF and sent securely over SSH
- Junos 11.4+ is recommended
- Ruby 1.9.3+ is required

- Junos Extension Toolkit(JET)

Enable 3rd party apps to run on Junos

1



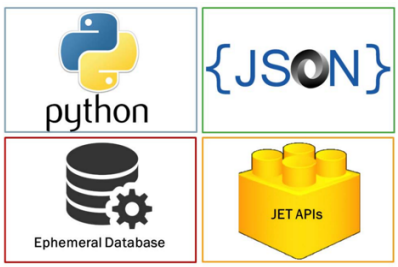
Program Junos control plane

2

Enable developers to create customized controls

3

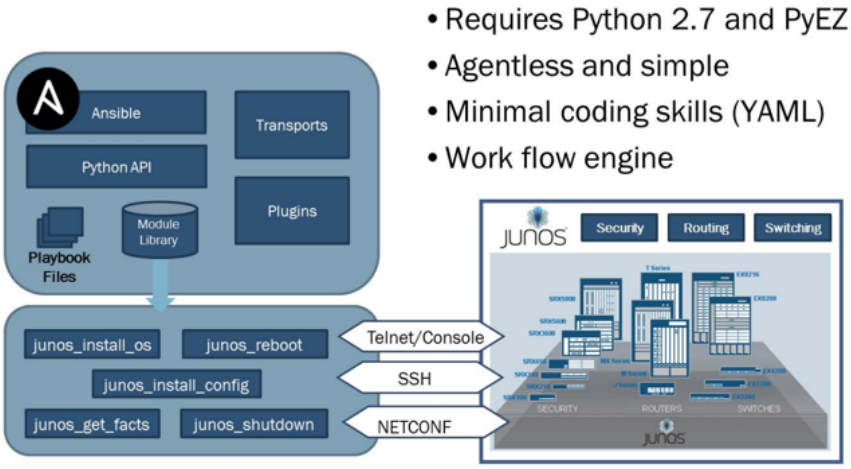
  - JET Component



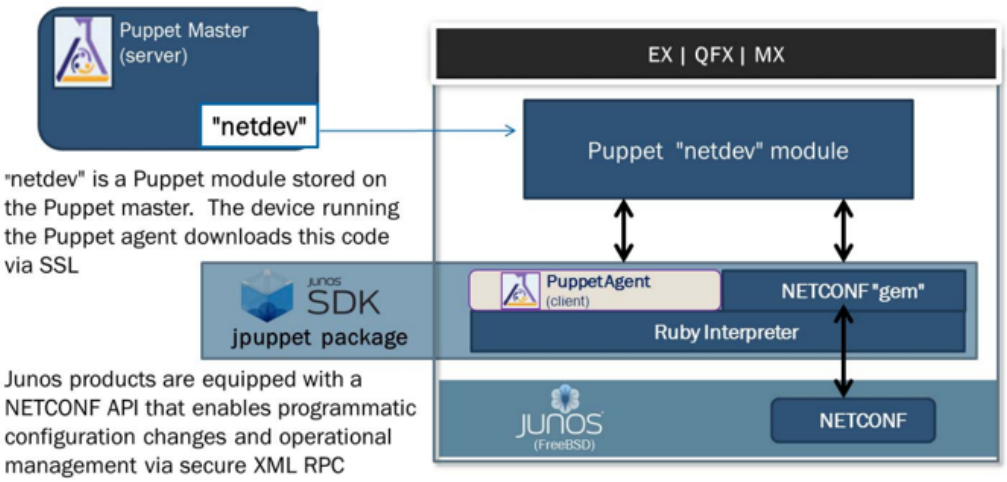
  - JET Development Environment
    - Installed as a Vagrant VM
    - Hosted on ubuntu
    - Required for applications with C or C++ dependencies
    - Needed to sign applications

- Automation Management Systems

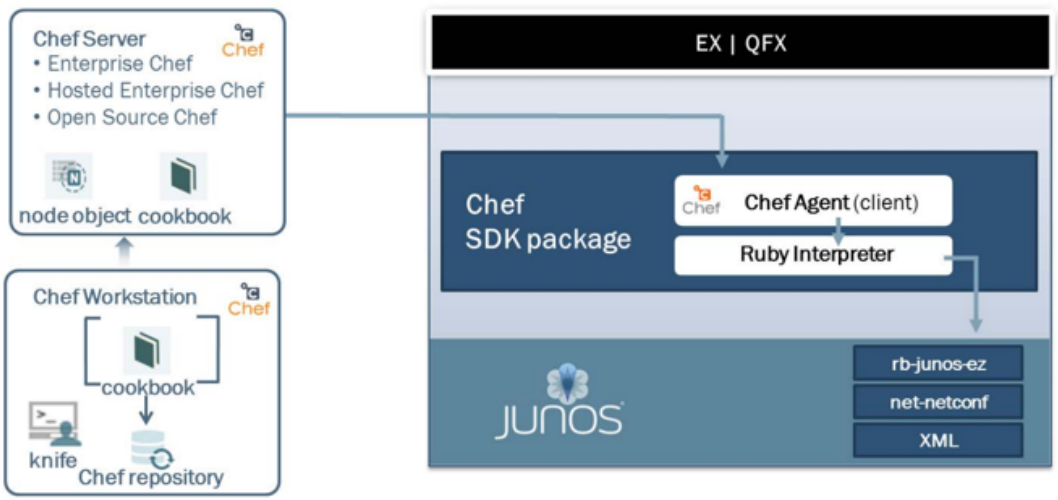
- Ansible



- Puppet
  - Requires Ruby and Jpuppet module



- Chef
  - Requires Chef client and Ruby interpreter



- SaltStack

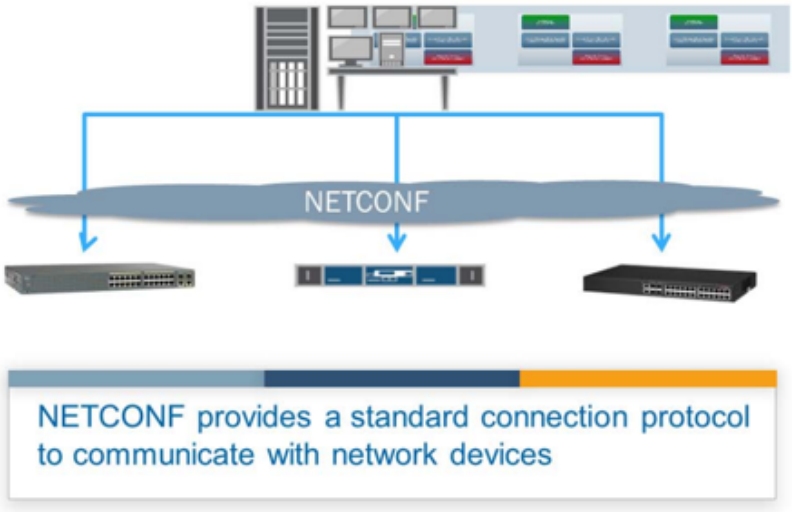
- Feature: Facts gathering(thu thập), managing configuration, RPC execution, CLI support, install software, and file copy
- Closed loop automation

- Junos Automation Tools
  - JSNAPy
    - Takes snapshots and compares Snapshots
    - Used to audit(đánh giá) environment against pre-defined criteria(tiêu chuẩn)
    - Python version of JSNAP
  - Junos ZTP
    - ZTP is called Auto Installation on some devices
    - Allow for zero-touch provisioning(cung cấp) of Junos devices
- Some questions
  - What are the two main daemons in the Junos OS that handle automation?
    - The two main processes in the Junos OS used for automation are the mgd and the jsd.
  - Why was the JET API added to the Junos OS?
    - The JET API was added to the Junos OS to improve commit times and adapt to the growing modularity of the Junos OS.
  - What does JSNAPy do?
    - JSNAPy is a Juniper tool written in Python that allows administrators to perform configuration management through the use of configuration snapshots.

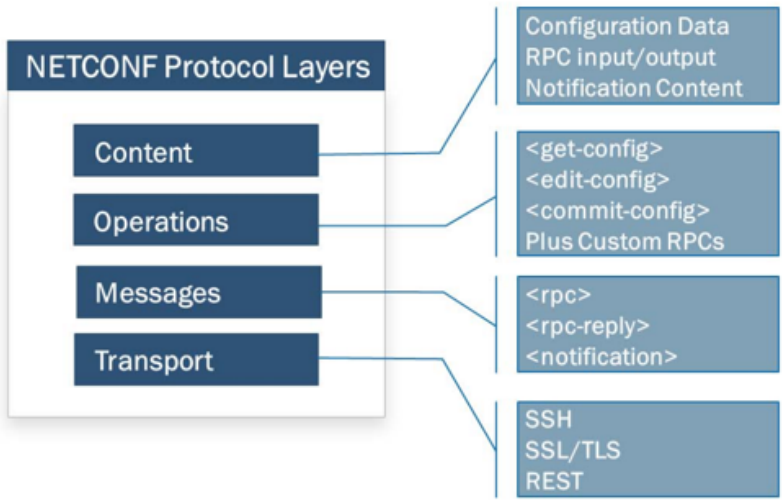
NETCONF and the XML API

- Objectives
  - Describe the NETCONF Protocol
  - Explain the capabilities(các khả năng) of the XML API
  - Describe the use of XSLT, SLAX, and XPath in XML API development
- NETCONF

- Why NETCONF?



- The NETCONF Protocol Layers

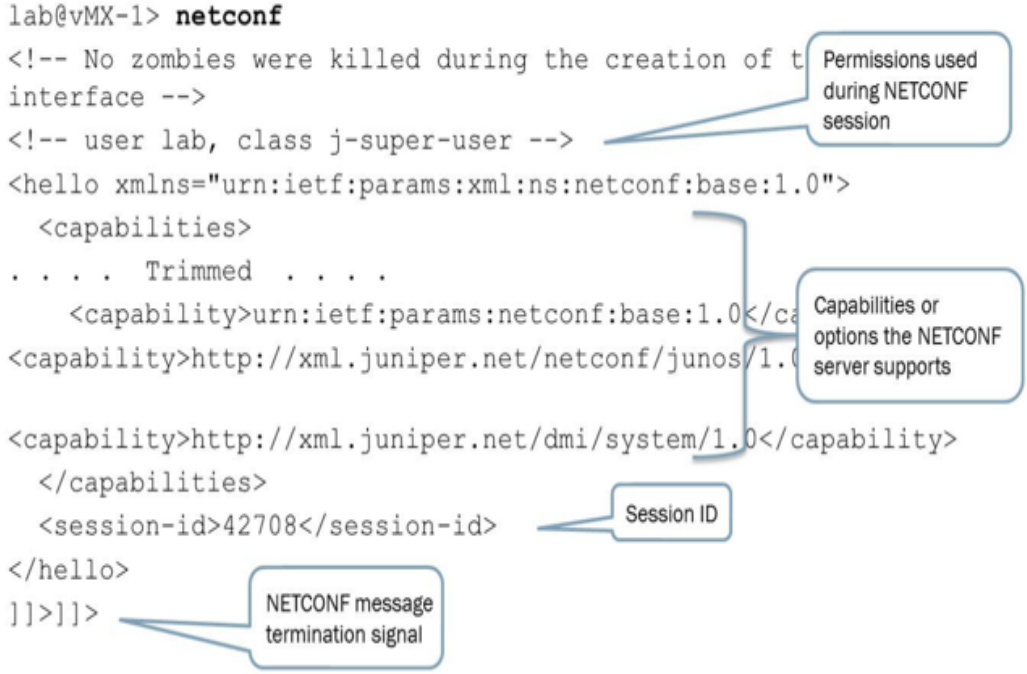


  - Messages Layer
    - <rpc>: encapsulates(tóm lược) all remote procedure(thủ tục) calls to the NETCONF server. This includes both operational mode and configuration RPCs
    - <rpc-reply>: encapsulates all remote procedure call replies from the NETCONF server. This includes data returned from the NETCONF server and any OK, error, or warning messages
    - <notification>: is a one-way message and is used to subscribe to a stream of data that the NETCONF server publishes
  - Operations Layer
    - <lock>: used to lock the configuration file before modifying it so that other users or
      RPCs cannot modify the config on a NETCONF device

- <unlock>: used to unlock the configuration after modifying it so that other users may access it
- <get>: retrieves data(gọi dữ liệu ra) from the running configuration database or device statistics. The get operation is used to retrieve the active configuration and device state information.
- <get-config>: used when retrieving the configuration off of a NETCONF device
- <edit-config>: used when changing the configuration on a NETCONF device
- <copy-config>: used to make a copy of the configuration on a NETCONF devices
- <commit>: used to commit the candidate configuration to the running configuration
  on a NETCONF device
- <delete-config>: used to delete a section or all of a NETCONF device
- <create-subscription>: used to create a NETCONF subscription
- <close-session>: issued to close a NETCONF session
- <kill-session>: used to close a different NETCONF session that the one you are currently using

- Content Layer
  - Contains RPCs(remote procedure call)
  - Contains configuration data
- Transport Layer
  - encrypted using SSH
- Example...
  - Configure Junos OS for NETCONF
    - Configure Junos to accept NETCONF SSH session over port #830

```
[edit system services]
lab@vMX-1# set netconf ssh

[edit system services]
lab@vMX-1# show
ssh;
netconf {
    ssh;
}
```

- Start a NETCONF Session
  - 

- XML API
  - The Junos OS XML API
    - XML API Uses
      - Issue operational mode commands
      - Change the device configuration
    - XML based encoding
      - All commands and configuration encapsulated in XML element tags
    - The Display XML RPC Pipe Command

```
lab@vMX-1> show chassis alarms | display xml rpc
<rpc-reply>
<rpcxmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
    <rpc>
        <get-alarm-information>
        </get-alarm-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

  - The Display XML Pipe Command

- CLI

```
lab@vMX-1> show chassis alarms
No alarms currently active
```

- | display xml

```
lab@vMX-1> show chassis alarms | display xml
<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
    <alarm-information
xmlns="http://xml.juniper.net/junos/17.1R1/junos-alarm">
        <alarm-summary>
            <no-active-alarms/>
        </alarm-summary>
    </alarm-information>
    . . . Trimmed . . .
```

- The Display JSON Pipe Command

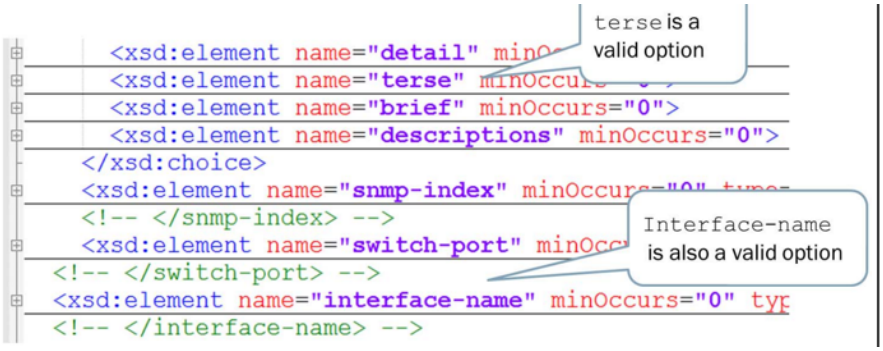
```
lab@vMX-1> show chassis alarms | display json
{
    "alarm-information" : [
    {
        "attributes" : {"xmlns" :
"http://xml.juniper.net/junos/17.1R1/junos-alarm"},
        "alarm-summary" : [
        {
            "no-active-alarms" : [
            {
                "data" : [null]
            }
            ]
        }]}]}
```

- Find the Correct RPC
  - Part 1

```
lab@vMX-1> show interfaces ge-0/0/0 terse | display xml rpc
<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
    <rpc>
        <get-interface-information>
                <terse/>
                <interface-name>ge-0/0/0</interface-name>
        </get-interface-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

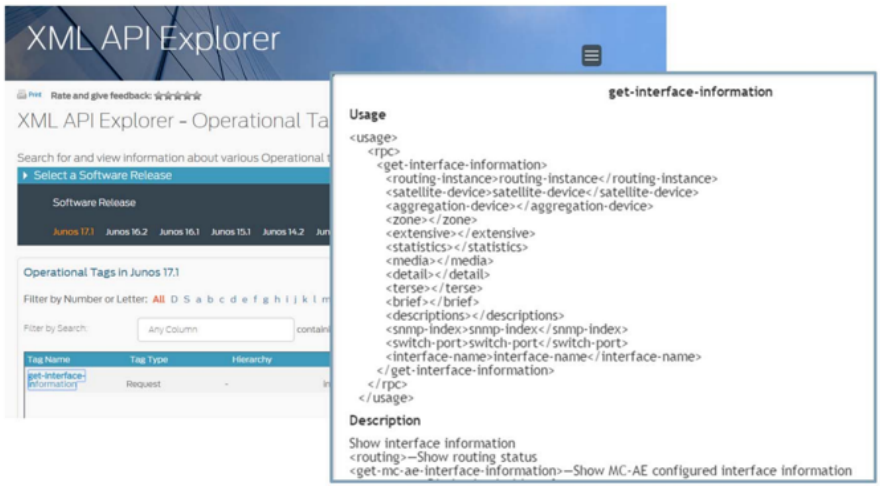
    - Fix: show interfaces terse ge-0/0/0

- Part 2

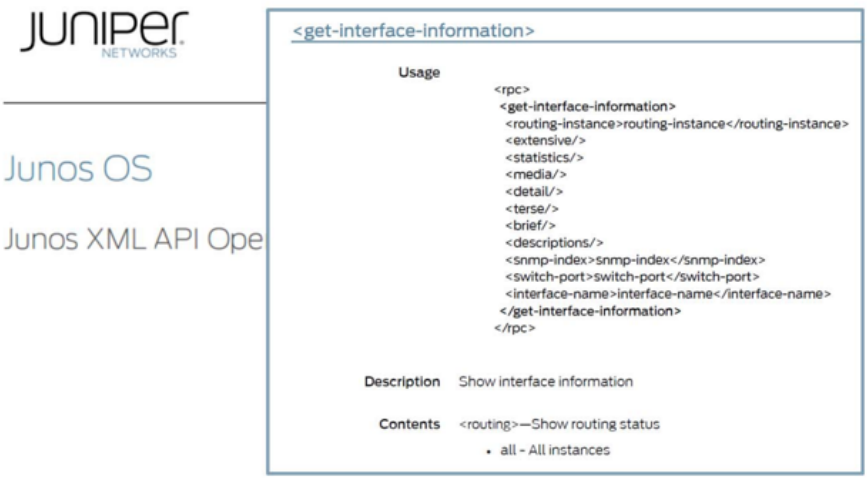


  - Use the Junos schema documents
  - The schema (.XSD) document can be downloaded or generated from Junos device

- Part 3



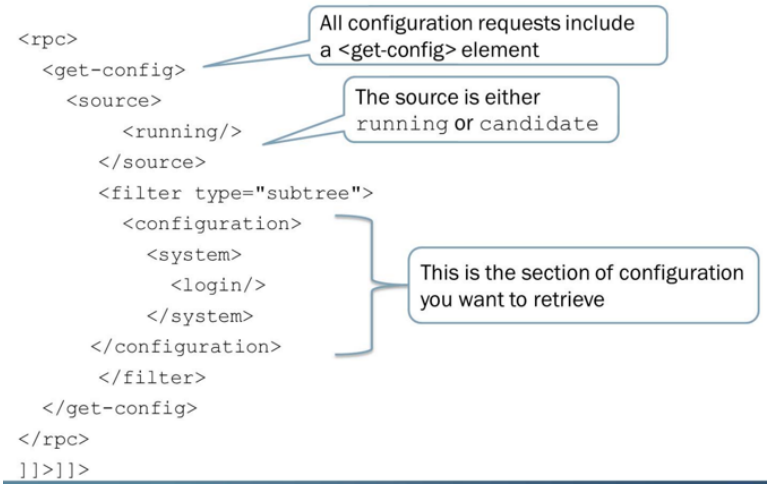
- Part 4



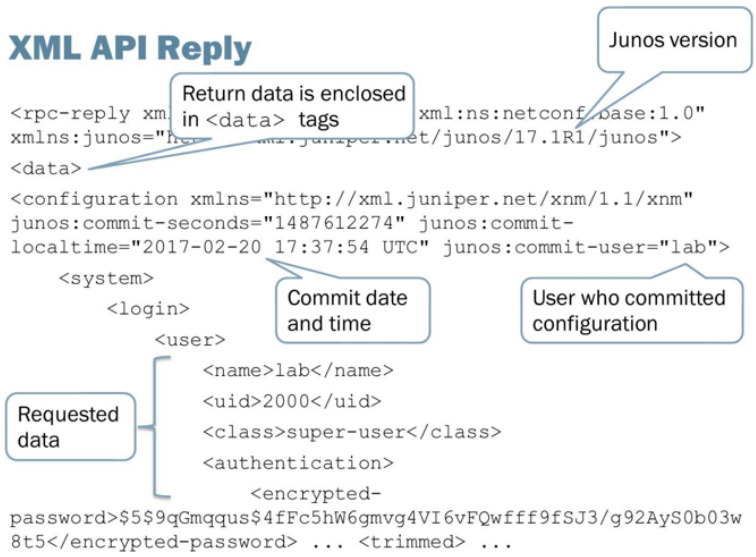
  - Junos XML API Operational Developer Reference

- The XML Schemas
  - Consists of two files

        - XML Schema for Configuration Data
        - XML Schema for Operational Data
    - Downloadable from Junos XML API download site
    - Unpacked files become
        - Config-16.2.xsd
        - operational-command-16.2.xsd
    - Config schema retrievable(có thể phục hồi) from Junos device using NETCONF
- Requesting Configuration Data Using the XML API



- XML API Reply



- XML API Programming Languages
    - Using XSLT To Automate the Junos XML API
        - Designed for XML to XML transformations
        - Find specific nodes in the configuration hierarch(phân cấp)

- Perform if-then type logic
- Perform loops

- XSLT Conditional Logic

```
<xsl:choose>
<xsl:when test="system/host-name">
     <change>
          <system>
               <host-name>M320</host-name>
          </system>
     </change>
</xsl:when>
<xsl:otherwise>
     <xnm:error>
          <message>
              Missing [edit system host-name] M320.
          </message>
     </xnm:error>
</xsl:otherwise>
</xsl:choose>
```

When the `host-name` statement is included at the `[edit system]` hierarchy level, change the hostname to `M320`. Otherwise, issue the warning message: `Missing [edit system host-name] M320.`

- The SLAX Programming Language
  - Open source language
  - Direct translates to XSLT
  - Easier to learn and program than XSLT
- The XML Path Language (XPath)
  - XPath Nodes
    - Element
    - Text
    - Attribute
  - XPath Axes(hệ trục)
    - Ancestor(Tổ tiên) or Parent
    - Child
    - Sibing
  - XPath Predicates and Operators
    - Predicates
      -

```
server[name = '10.1.1.1']
*[@inactive]
route[starts-with(next-hop, '10.10.')]
```

- Operators
  -

```
Logical Operators
      AND, OR
Comparison Operators
      =, !=, <, and >
Numerical Operators
      +, -, and *.
```

- Adding Selection Criteria(tiêu chuẩn)
  - Get the user id for user "lab"
    - Using system/login/user/uid
      -

```
user@R1> show configuration system | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
  <configuration … Trimmed …>
     <version>17.1R1.8</version>
        <system>
           <login>
              <user>
                 <name>lab</name>
                 <uid>2000</uid>
                 <class>super-user</class>
                 <authentication>
                   <encrypted-password>$1$8…Trimmed…</encrypted-password>
                 </authentication>
              </user>
              <user>
                 <name>user</name>
                <uid>2001</uid>[...]
```

    - Use square brackets after the user element to specify that you want the user whose name is "lab"
      -

System/login/user[name=='lab']/uid

```
user@R1> show configuration system | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
  <configuration … Trimmed …>
     <version>17.1R1.8</version>
        <system>
           <login>
              <user>
                 <name>lab</name>
                 <uid>2000</uid>
                 <class>super-user</class>
                 <authentication>
                   <encrypted-password>$1$8…Trimmed…</encrypted-password>
                 </authentication>
              </user>
              <user>
                 <name>user</name>
                <uid>2001</uid>[...]
```

- Elements with Attributes

- Elements

```
<wrapper>
    <element>
        <name>Bob</name>
    </element>
</wrapper>
```

  - Xpath: element/name
- Elements with and attribute

```
<wrapper>
    <element name="Bob" />
</wrapper>
```

  - Xpath: element/@name
- Command Attributes
  - junos:changed is attached to nodes that have change
  - junos:group is attached to nodes that have been inherited(thừa kế) from a configuration group

- XML API Tools
  - Other Language Options
    - Language Possibilities

    - Language Libraries make it easier
  - Junos XML Protocol Perl Client
    - Perl module-- JUNOS:DEVICE
      - Execute XML API Operational Commands
      - Retrieve, modify, and commit configurations
    - Include sample scripts
    - Download

      Downloads
  - NETCONF Java Toolkit
    - Makes working with NETCONF in Java easier

- Includes four main classes

Class	Description
Device	Defines the device on which the NETCONF server runs, and represents the SSHv2 connection and default NETCONF session with that device
NetconfSession	Represents a NETCONF session established with the device on which the NETCONF server runs.
XMLBuilder	Creates XML-encoded data.
XML	XML-encoded data that represents an operational or configuration request or configuration data.

- Download

  Releases · Juniper/netconf-java · Git...

- Questions...
  - List three NETCONF operation level command
    - There are 11 NETCONF Operational commands including :<lock>, <unlock>, <get>, <get-config>, <edit-config>, <copy-config>, commit> , <discard-changes>, <delete-config>, <validate>,<create subscription>, <close-session>, and <kill-session>
  - What W3C language is designed to manipulate(vận dụng/thao tác) XML?
    - XSLT is the W3C language designed to manipulate XML.
  - What is the purpose of Xpath?
    - The purpose of XPath is to navigate an XML tree and identify elements for examination or manipulation
  - Name at least one language library simplifies working with NETCONF and the XML API
    - The Junos Perl Module and NETCONF Java classes are libraries that make working with NETCONF and the XML API easier

JSON and YAML

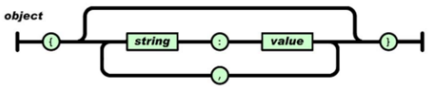
- Objectives
  - Identify where JSON and YAML are used in Junos OS Automation
  - Read JSON and YAML documents
- JSON and YAML Overview
  - What are JSON and YAML?
    - Javascript Object Notion(JSON)
      - Easy to read
      - Language independent
      - Familiar syntax for programmers

- YAML Ain't Markup Language(YAML)
  - Easier to read
  - Language independent
- JSON and YAML Similarities

- Where are JSON and YAML used?
  - JSON
    - Junos OS Configuration
    - Junos REST API
  - YAML
    - Junos PyEZ Tables
    - Ansible Playbooks
    - JSNAPy

- JSON
  - JSON Basics
    - JSON is case sensitive(nhạy cảm)
    - JSON uses curly braces {} for structure
    - Whitespace is ignored in JSON
    - JSON does not offer a way to comment the code
    - JSON structures
      - Objects
      - Arrays
  - JSON objects

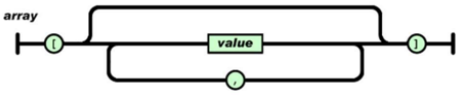
- contain key-value pairs



- Example

```
{
 "interface" : "ge-0/0/0.0",
 "address" : "172.17.1.1/24"
}
```

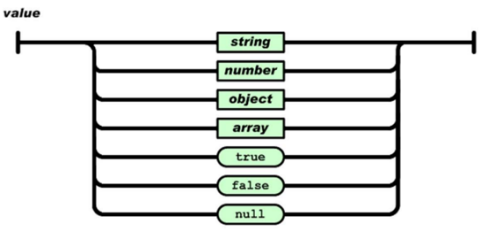
- JSON arrays
  - encapsulate data in [ ]



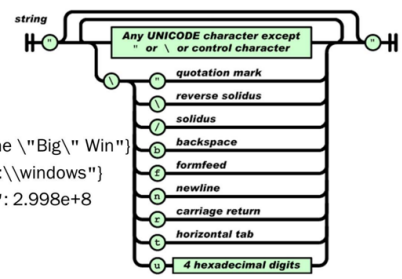
  - Example

```
{
  "physical-interface": [{
    "name": [{
      "data": "ge-0/0/0"
    }],
    "admin-status": [{
       "data": "up"
    }]
  }]
}
```

- JSON values
  - Objects and array values enable nesting



- JSON Strings and Numbers



  - Examples
    {"title": "The \"Big\" Win"}
    {"path": "c:\\windows"}
    {"light m/s": 2.998e+8

- Configuration in JSON

```
[edit]
lab@vMX-1# show protocols bgp | display json
{
    "configuration" : {
        "@" : {
            "junos:changed-seconds" : "1489765454",
            "junos:changed-localtime" : "2017-03-17 15:44:14 UTC"
        },
        "protocols" : {
            "bgp" : {
                "group" : [
                {
                    "name" : "ext-peers",
                    "type" : "external",
                    "peer-as" : "65513",
                    "neighbor" : [
                    {
                        "name" : "172.17.1.2"
                    }
                    ]
                }]}}}
```

- Operation Mode Command Output in JSON

```
lab@vMX-1> show chassis alarms | display json
{
    "alarm-information" : [
    {
        "attributes" : {"xmlns" :
"http://xml.juniper.net/junos/16.2R1/junos-alarm"},
        "alarm-summary" : [
        {
            "no-active-alarms" : [
            {
                "data" : [null]
            }
            ]
        }
        ]
    }
    ]
}
```

- YAML
  - YAML Basics
    - YAML is case-sensitive
    - YAML uses indents for structure
    - YAML uses spaces, not tabs
    - YAML document start with three dashes ---
    - Comment begin with a #
    - String do not need quotes unless they include special characters
    - YAML Structures
      - Mappings
      - Sequences
  - YAML Mappings
    - Mappings are key-value pairs
    - Mappings are similar to JSON objects

- Example

```
---
name : Bob
height : 6 foot
```

- YAML Sequences
  - Sequences are lists of arrays of data
  - Sequences are similar to JSON arrays
  - Sequences items start with a dash -
  - Example

```
---
-apple
-orange
-banana
```

- Nested Mappings and Sequences
  - Mappings and sequences can be nested

```
---
Fruit:
- apple
- orange
- banana
Vegetable:
  cruciferous:
  - radish
  - wasabi
  gourd:
  - cucumber
  - pumpkin
```

- Junos PyEZ Example

```
---
bgpTable:
 rpc: get-bgp-neighbor-information
 item: bgp-peer
 view: bgpView
 key: peer-id
bgpView:
 fields:
  local_as: local-as
  peer_as: peer-as
  local_address: local-address
  peer_id: peer-id
  local_id: local-id
  route_received: bgp-rib/received-prefix-count
```

- Ansible Example

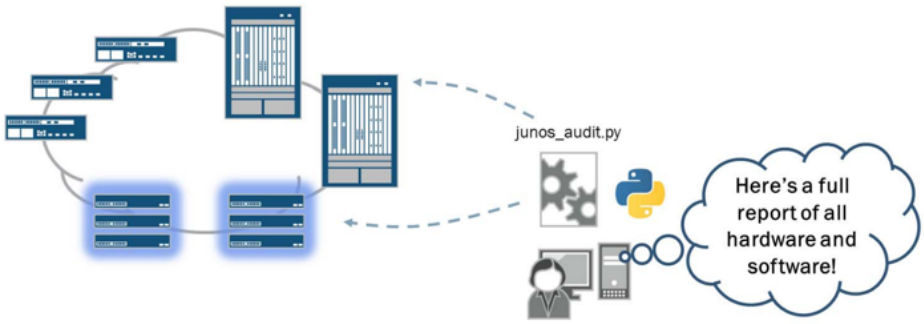
```
---
- name: Install Junos OS
  hosts: dc1
  roles:
    - Juniper.junos
  connection: local
  gather_facts: no
  vars:
    wait_time: 3600
    pkg_dir: /var/tmp/junos-install
    OS_version: 14.1R1.10
    OS_package: jinstall-14.1R1.10-domestic-signed.tgz
    log_dir: /var/log/ansible
. . .Trimmed. . .
```

- Questions...
  - What language are JSON and YAML replacing?
    - XML
  - Is whitespace important in JSON?
    - No
  - How do you create a comment in YAML?
    - #

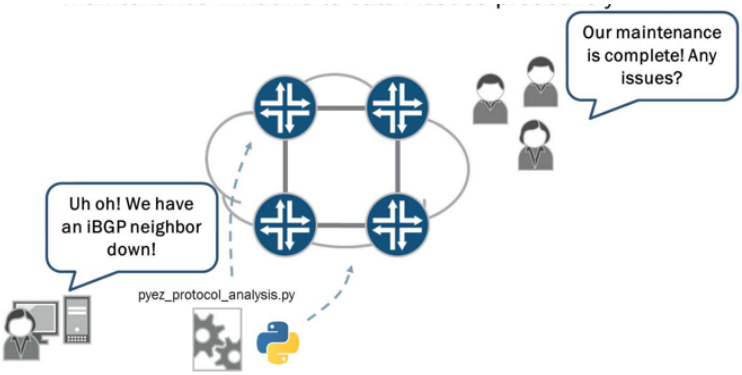
Python and Junos PyEZ

- Objectives
  - Build a Python environment where you can manage devices running the Junos OS
  - Get informations from Junos OS RPCs using PyEZ scripts
  - Modify the Junos OS configuration using PyEZ
  - Use PyEZ tables and views to views RPCs and modify the Junos OS configuration
  - Use the PyEZ exception handling modules
- Introduction to Python and Junos PyEZ

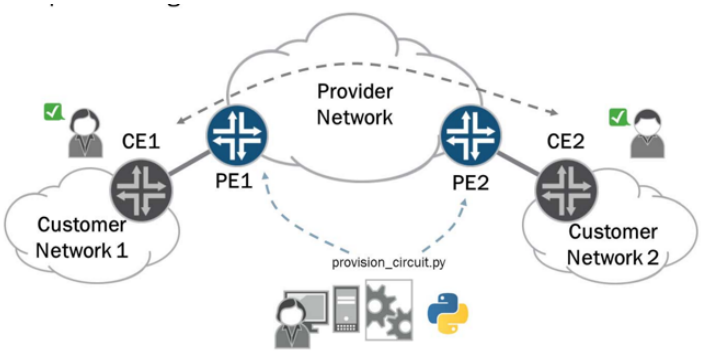
- PyEZ Use Cases
  - Software and hardware audits(kiểm tra)
    - On-demand access(y/c truy cập) to all hardware and software running on Junos platforms on devices running Junos OS



  - Automated data collection
    - Analyze device and protocol state before and after maintenance windows to catch issues proactively(chủ động)



  - Basic automated configuration deployment
    - Deploy basic Junos OS configuration templates for service provisioning(cung cấp) to eliminate(loại bỏ) errors

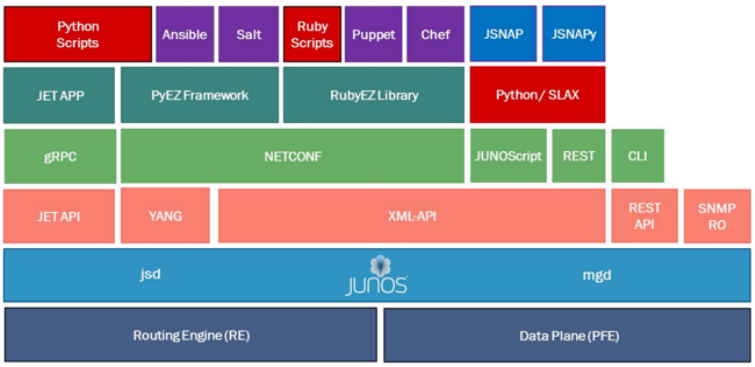


- Junos Automation Now Includes Python On-Box
  - Python extensions(mở rộng) package embedded in Junos OS
    - On-box as of Junos 16.1R3
    - Supports Python 2.7

- Uses PyEZ APIs
  - Easy to execute RPCs
  - Easy to perform operational and configuration tasks
- Able to perform tasks not possible in XSLT and SLAX

- Junos OS Python Modules
  - Junos OS includes many different open-source Python modules to facilitate(thuận tiện) automation

- The PyEZ Package
  - PyEZ is a "microframework" allowing for "Pythonic"access to Junos



- PyEZ Modules

Modules	Description
device	Defines the Device class, which represents the device running Junos OS and enables you to connect to and retrieve facts from the device.
exception	Defines exceptions encountered when accessing, configuring, and managing devices running Junos OS.
factory	Contains code pertaining to Tables and Views, including the loadyaml() method, which is used to load custom Tables and Views.
op	Includes predefined operational Tables and Views that can be used to filter output for common operational commands.
resources	Includes predefined configuration Tables and Views representing specific configuration resources, which can be used to programmatically configure devices running Junos OS.
transport	Contains code used by the Device class to support different connection types such as telnet and serial console connections.
utils	Includes configuration utilities, file system utilities, shell utilities, software installation utilities, and secure copy utilities.

Python Development Environment

Summary

- Python and Python modules are includes on devices running Junos OS
  - Junos OS release 16.1R3 for MX series and PTX series device
- Included in the JET virtual machine
- Install on personal workstation
  - Install Python
  - Install PyEZ Dependencies
  - Install PyEZ package

Building a Basic PyEZ Environment

- Installation and Setup of PyEZ

Junos PyEZ Developer Guide - Tech...

```
bash$ sudo apt-get install python-pip
.
bash$ sudo apt-get install python-dev
.
bash$ sudo apt-get install libxml2-dev
.
bash$ sudo apt-get install libxslt-dev
.
bash$ sudo pip install junos-eznc
.
```

Python virtualenv Package

- Use to create Python development environments

- Active a virtualenv

```
[lab@jaut-desktop ~]$ source virtualenvs/jaut_env/bin/activate
(jaut_env) [lab@jaut-desktop ~]$
```

- Install software in virtualenv: `pip install junos-eznc`
- Deactivate a virtualenv: `Deactivate`
- Do not store(ko lưu trữ) project files in virtualenv

Prepare Devices to be Managed

Configure Management Network Connections

- Console, telnet, SSH

```
[edit system services]
lab@vMX-1# set netconf ssh
```

- Create a user account with sufficient (đủ) permissions
- Configure public/private keys for authentication(optional)
- Configure the language python statement

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

Executing Unsigned Python Scripts

- Python does not execute unsigned scripts by default

Additional steps to executing unsigned scripts

- Set language statement

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

- 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
```

- Store script in appropriate directory

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

- Adding the following if storing scripts on flash storage

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

- Script must meet permission requirements
  - Executing 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 need read permissions
    - Event and SNMP scripts need to configure the `python-script-user username` statement
- The Python Interpreter
  - Reads and executes commands interactively(tương tác)
  - To start Python Interpreter: python or python2.7
  - To exit the Python Interpreter: Ctrl + d or exit()

Python Interpreter Output

```
(jaut_env) [lab@jaut-desktop ~]$ python
Python 2.7.12 (default, Apr 22 2017, 07:04:30)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-18)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> exit()
```

- Python Script Structure
  - 1. Import libraries

```
from jnpr.junos import Device
from pprint import pprint
```

  - 2. Create device connection string

```
dev = Device(host='vMX-1', user='lab',
password='lab123')
```

  - 3. Open connection to a device: 'dev.open()'
  - 4. Perform task: 'pprint(dev.facts)'
  - 5. Close connection: 'dev.close()'
- Working with RPCs
  - Using PyEZ to Call RPCs

- Find the RPC using the CLI

```
lab@vMX-1> show interfaces terse | display xml rpc
<rpc-reply
xmlns:junos="http://xml.juniper.net/junos/17.1R1/junos">
    <rpc>
        <get-interface-information>
                <terse/>
        </get-interface-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

- Show the XML RPC using PyEZ

```
from jnpr.junos import Device
from lxml import etree
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()
print (dev.display_xml_rpc('show interfaces terse',
format='text'))

dev.close()
```

```
<get-interface-information>
    <terse/>
</get-interface-information>

------------------
(program exited with code: 0)
Press return to continue
```

- The RPC call can be translated by changing hyphens(nối) to underscores and tags to arguments

These XML Tags:

```
<get-interface-information>
    <terse/>
    <interface-name>lo0</interface-name>
</get-interface-information>
```

Translate into:

```
dev.rpc.get_interface_information (interface_name='lo0',
terse=True)
```

- Example RPC calls

```
from jnpr.junos import Device
from lxml import etree
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()
#1. Basic RPC call - Returns XML
op = dev.rpc.get_interface_information(dev_timeout = 60)
#2. Basic RPC call - Returns Text
op = dev.rpc.get_interface_information({'format':'text'})
#3. RPC Call with additional attributes
op = dev.rpc.get_interface_information (interface_name='lo0',
extensive=True)
print (etree.tostring(op))
#4. Display a specific piece of information using xpath
for i in op.xpath('.//link-level-type'):
    print i.text
dev.close()
```

- Using an RPC to Retrieve the Junos OS Configuration

```
from jnpr.junos import Device
from lxml import etree
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()

#1. View the whole configuration
cnf = dev.rpc.get_config(dev_timeout = 60)

#2. View partial configuration using XPath
cnf = dev.rpc.get_config(filter_xml=etree.XML
('<configuration><interfaces/></configuration>'))

print etree.tostring(cnf)
dev.close()
```

- Normalizing the XML RPC Reply
  - Normalizing RPC output removes excess whitespace
  - Normalizing RPC output makes it easier to parse

- Three places to normalize

```
dev = Device(host='vMX-1', user='lab', passwd='lab123',
normalize=True)
dev.open(normalize=True)
op = dev.rpc.get_alarm_information(normalize=True)
```

Default RPC Reply

```
<alarm-information>
<alarm-summary>
<no-active-alarms/>
</alarm-summary>
</alarm-information>

------------------
(program exited with code: 0)
Press return to continue
```

Normalized RPC Reply

```
<alarm-information><alarm-
summary><no-active-alarms/></alarm-
summary></alarm-information>

------------------
(program exited with code: 0)
Press return to continue
```

- Working With and Unstructured Configuration
  - Unstructured Configuration Changes
    - You can use XML, JSON, Junos OS 'set' commands, or ASCII text to make configuration changes
    - Begin by creating a 'config' object and associating(kết hợp) it with the Device object
      - dev = Device(host='vMx-1', user='lab', passwd='lab123')
        cu = Config(dev)
    - Follow the process
      - 1. Lock the configuration using 'lock()'
      - 2. Load the new configuration changes using 'load()', roll back the configuration using 'rollback()', revert to a rescue(cứu hộ) configuration using 'rescue()'
      - 3. Commit the configuration using 'commit()'
      - 4. Unlock the configuration using 'unlock()'
    - Example using text data

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()
cu = Config(dev)
data = """interfaces {
    ge-0/0/1 {
        description "Test interface";
        unit 0 {
            family inet {
                address 172.17.1.150/24;
            }
        }
    }
}
"""
```

```
cu.lock()
cu.load(data, format='text')
cu.pdiff()
if cu.commit_check():
   cu.commit()
else:
   cu.rollback()
cu.unlock()
dev.close()
```

```
Script Output
[edit interfaces ge-0/0/1]
+   description "Test interface";
[edit interfaces ge-0/0/1 unit 0 family inet]
        address 172.17.1.100/24 { ... }
+       address 172.17.1.150/24;

------------------
(program exited with code: 0)
Press return to continue
```

- Using 'set' command to change the configuration

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()

with Config(dev, mode='private') as cu:
    data='set system services netconf traceoptions file
test.log'
    cu.load(data, format='set', merge=False, overwrite=False)
    cu.pdiff()
    cu.commit()

dev.close()
```

```
Script Output
[edit system services netconf]
+     traceoptions {
+         file test.log;
+     }

------------------
(program exited with code: 0)
Press return to continue
```

- Load Configuration From a File

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()

with Config(dev, mode='private') as cu:
    conf_file = "test.conf"
    cu.load(conf_file, merge=True)
    cu.pdiff()
    cu.commit()

dev.close()
```

```
Script Output
[edit routing-options static]
     route 0.0.0.0/0 { ... }
+    route 172.31.2.0/24 next-hop 172.25.11.254;

------------------
(program exited with code: 0)
Press return to continue
```

- Installing Software

```
from jnpr.junos import Device
from jnpr.junos.utils.sw import SW

dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()

def update_progress(dev, report):
    print dev.hostname, '> ', report

sw = SW(dev)
ok = sw.install(package=r'/home/lab/Downloads/jinstall-
lx.lxxxx.tgz', progress=update_progress)
if ok:
    print 'rebooting'
    sw.reboot()
dev.close()
```

- Working With Tables and Views
  - PyEZ Tables and Views
    - PyEZ tables and views provide a structured way to RPC and configuration data
      - Table hold data for an RPC or configuration
      - Views or a subnet of specific fields of the table
    - Tables and views are defined using YAML files
    - PyEZ has many predefined tables and views included with the PyEZ installation
    - You can define your own tables and views

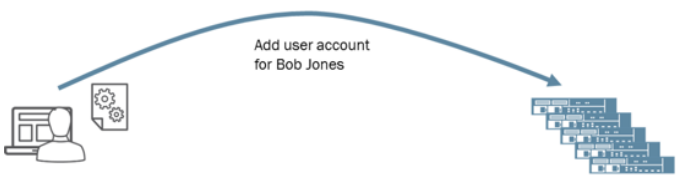
- PyEZ Table and View Files

```
---
ArpTable:
  rpc: get-arp-table-information
  item: arp-table-entry
  key: mac-address
  view: ArpView

ArpView:
  fields:
    mac_address: mac-address
    ip_address: ip-address
    interface_name: interface-name
```

- Loading a Table and View
  - 1. Import the table
    - from jnpr. junos.op.arp import ArpTable
  - 2. Associate ArpTable with Device
    - arp = ArpTable(dev)
  - 3. Retrieve items
    - arp.get()
  - 4. Output results

```
for mac in arp:
    print ("{}: {}  {}".format(mac.mac_address,
mac.ip_address, mac.interface_name))
```

- Tables and Views to Access Junos Configuration
  - PyEZ tables provide access to the Junos OS configuration data
  - Access can be read-only using get property
  - Access can be read-write using set property
- Modifying the Configuration Using Tables
  - Case study scenario
    - Create a Python script using PyEZ that allows a user who doesn't know the Junos OS to add a user account to a device running the Junos OS
      -



- Identify the Junos configuration hierarchy level

```
lab@vMX-1> show configuration system login | display xml
<rpc-reply>
    <configuration>
            <system>
                <login>
                    <user>
                        <name>lab</name>
                        <uid>2000</uid>
                        <class>super-user</class>
                        <authentication>
                            <encrypted-password>$1kj3.</encrypted-password>
                        </authentication>
                    </user>
                </login>
            </system>
    </configuration>
   . . . Trimmed . . .
</rpc-reply>
```

This is the Hierarchy we need to add a new user

- Create table definitions for structured resource

```
UserAccountTable:
    set: system/login/user
    key-field: username
    view: UserAccountView
```

- Create the view definitions

```
UserAccountView:
  groups:
    auth: authentication
  fields:
    username: name
    userclass:
      class:
        default: unauthorized
    uid:
      uid:
        type: int
        default: 1001
        minValue: 100
        maxValue: 64000
    fullname: full-name
  fields_auth:
    password: encrypted-password
```

- Save the Files
  - 1. Create a folder called 'myTables' in your project folder to store your table definitions
  - 2. Save the file with your UserAccountTable and UserAccountView as configTables.yml in myTables folder
  - 3. Create file called configTables.py in myTables folder with following four line of code

```
from jnpr.junos.factory import loadyaml
from os.path import splitext
_YAML_ = splitext(__file__)[0] + '.yml'
globals().update(loadyaml(_YAML_))
```

- 4. Create a blank file call ___init___.py and save it to myTables folder

- Write the script

```
from jnpr.junos import Device
from myTables.configTables import UserAccountTable
dev = Device(host='vMX-1', user='lab', passwd='lab123')
dev.open()

ua = UserAccountTable(dev)
ua.username = 'bob'
ua.userclass = 'super-user'
ua.password = 'lab123'
ua.append()
ua.set(merge=True, comment="Junos PyEZ commit")

#ua.lock()
#ua.load(merge=True
#ua.commit(comment="Junos PyEZ commit")
#ua.unlock()
dev.close()
```

- Test the result

```
from jnpr.junos import Device
from myTables.configTables import UserAccountTable
dev = Device(host='vMX-1', user='lab', passwd='lab123')
ua = UserAccountTable(dev)
dev.open()

ua.get()
for account in ua:
    print("Username is {}\nUser class is
{}".format(account.username, account.userclass))

dev.close()
```

```
Script Output
Username is bob
Userclass is super-user
Username is lab
Userclass is super-user
------------------
(program exited with code: 0)
Press return to continue
```

- PyEZ Exception Handling
  - Introduction to Exception Handling
    - Exception handling - important to protect against these scenarios
      - Host becoming unreachable periodically
      - Users locking the configuration database
      - The SSH user limit being reached

- PyEZ try/except Statement

```
from jnpr.junos import Device
from jnpr.junos.exception import *
from jnpr.junos.utils.config import Config
import sys
try:
    dev = Device(host='192.168.64.51', user='bunk',
password='bunk')
    dev.open()
    dev.close()
    print "Logged in!"
except Exception as error:
    if type(error) is ConnectAuthError:
        print "Authentication error!"
        sys.exit()
    else:
        print "Encountered an exception!"
        print error
        sys.exit()
```

try statement

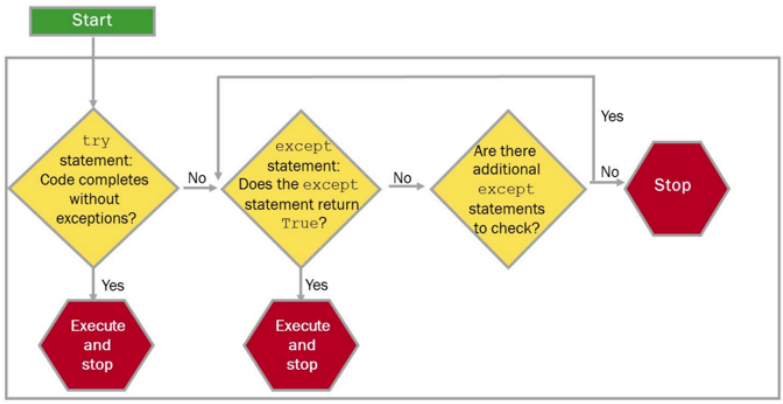
except statement

Bad username/password

- PyEZ try/except Example
  - The Python try/except statement allows the application/script to deal with exceptions

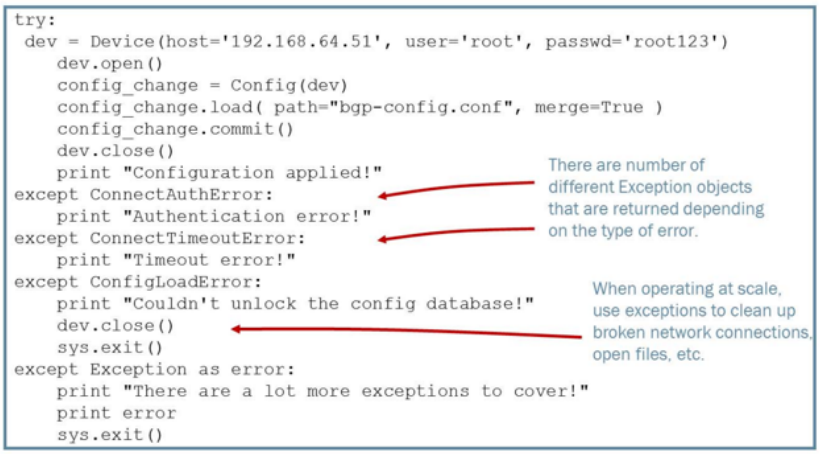
```
bash $ python pyez_exception_handling_1.py
Authentication error!
bash $
```

- PyEZ try/exception Statement Flowchart



- Troubleshooting PyEZ

- There are a wide array of exceptions that PyEZ catches. Consult the PyEZ API Docs for details

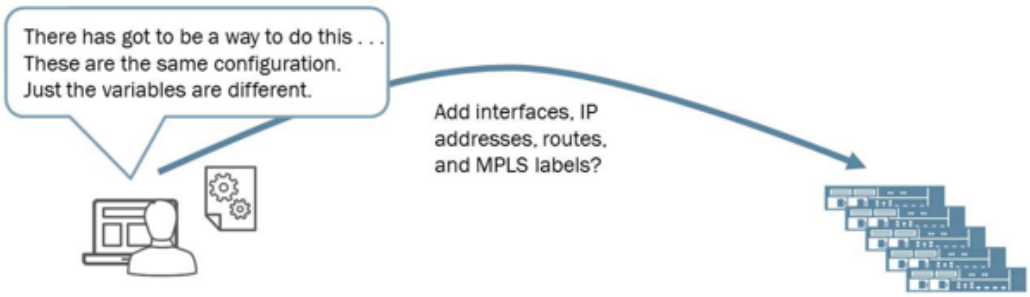


- Questions/Answers
  - 1. What is purpose of a Python virtualenv?
    - The Python Virtualenv is used to create an isolated development environment.
  - 2. What three ways can PyEZ connect to devices running the Junos OS
    - Junos PyEZ can connect to devices using a console connection, a telnet connection, and a NETCONF SSH connection.
  - 3. What is the shebang !# used for?
    - The !# is put at the beginning of a python file and enables a python script to be executed by just calling the name of the file.
  - 4. What dose it mean to normalized output?
    - Normalizing the output removes leading and trailing whitespace.
  - 5. What do you have to do to the RPC get-interface-infomation so that it can be called by PyEZ?
    - In order for a get-interface-information RPC to be called by PyEZ the RPC has to be converted by changing the hyphens to underscores (i.e., get_interface_information).

Jinja2 and Junos PyEZ

- Objectives
  - Understand how Jinja2 can help you automate the Junos OS
  - Understand the syntax of Jinja2
  - Create Jinja2 statements to automate a Junos OS configuration
- Jinja2 Overview

- How do I automate multiple devices?
  - What if the IP addresses, OSPF areas, or MPLS labels are difference on each device?



  - The solution: Jinja2, YAML and Junos PyEZ
    - 1. Turn your Junos OS configuration into a Jinja2 template
    - 2. Put interface IDs, IP addresses, and route information into a YAML data file
    - 3. Use Junos PyEZ to merge the two files and load the configurations
- Jinja2 Syntax
  - Overview
    - Jinja2 can turn any text file into a template
    - Jinja2 is the standard for templating in Python
    - Jinja2 documentation

      Jinja — Jinja Documentation (...
    - To install Jinja2: pip install jinja2
  - Variable Expansion
    - To expand or replace a section of code, use double curly brackets {{}}

```
interfaces {
    {{interface_ID}} {
        unit {{unit_num}} {
            family inet {
                address {{inf_address}};
            }
        }
    }
}
```

Example.yml

```
---
interface_ID: ge-0/0/0
unit_num: 0
inf_address: 172.25.11.1
```

  - Jinja2 if Statement
    - Conditionally importing templates
    - Determining if variables are defined

- Jinja2 for Loops

Jinja2 `for` Loops

- Repeat tasks

```
ethernet-switching-options {
    analyzer multi-session {
      input {
        ingress {
{%- for iface in host_ports %}
            interface {{ iface }}.0;
{%- endfor %}
        }
        egress {
{%- for iface in host_ports %}
            interface {{ iface }}.0;
{%- endfor %}
        }
      }
... Trimmed ...
```

Script Output

```
ethernet-switching-options {
    analyzer multi-session {
      input {
        ingress {
            interface ge-0/0/1.0;
            interface ge-0/0/2.0;
            interface ge-0/0/3.0;
            interface ge-0/0/4.0;
            interface ge-0/0/5.0;
        }
        egress {
            interface ge-0/0/1.0;
            interface ge-0/0/2.0;
            interface ge-0/0/3.0;
            interface ge-0/0/4.0;
            interface ge-0/0/5.0;
```

Example.yml

```
---
host_ports:
    - ge-0/0/1
    - ge-0/0/2
    - ge-0/0/3
    - ge-0/0/4
    - ge-0/0/5
```

- Jinja2 Comments
  - Use {#...#} to enclose comments

```
{# This code not needed, keep for reference
    {% for user in users %}
        ...
    {% endfor %}
#}
```

- Jinja2 include Directive
  - Use include statements to pull in other Jinja2 templates
    - Example 1

```
{% if install == 'new' %}
{% include 'new_install.conf' %}
{% else %}
{% include 'merge_install.conf' %}
{% endif %}
```

    - Example 2

```
{% include 'merge_install.conf' ignore missing %}
```

- Jinja2 Set Directive

- Use the set directive to assign values to variables

```
{% set install = 'new' %}
{% if install == 'new' %}
{% include 'new_install.conf' %}
{% else %}
{% include 'merge_install.conf' %}
{% endif %}
```

- Jinja2 Custom Filters
  - Jinja2 filters modify the content of variables
  - Filters are separated(phân tách) from the content by a pipe (|) symbol
- Jinja2 Math Operators

Operator	Description
+	Adds two objects together. Usually the objects are numbers, but if both are strings or lists, you can concatenate them this way. This, however, is not the preferred way to concatenate strings! For string concatenation, see at the (~) operator. {{ 1 + 1 }} is 2.
-	Subtract the second number from the first one. {{ 3 - 2 }} is 1.
/	Divide two numbers. The return value will be a floating point number. {{ 1 / 2 }} is {{ 0.5 }}. (Just like from __future__ import division.)
//	Divide two numbers and return the truncated integer result. {{ 20 // 7 }} is 2.
%	Calculate the remainder of an integer division. {{ 11 % 7 }} is 4.
*	Multiply the left operand with the right one. {{ 2 * 2 }} would return 4. This can also be used to repeat a string multiple times. {{ '=' * 80 }} would print a bar of 80 equal signs.
**	Raise the left operand to the power of the right operand. {{ 2**3 }} would return 8.

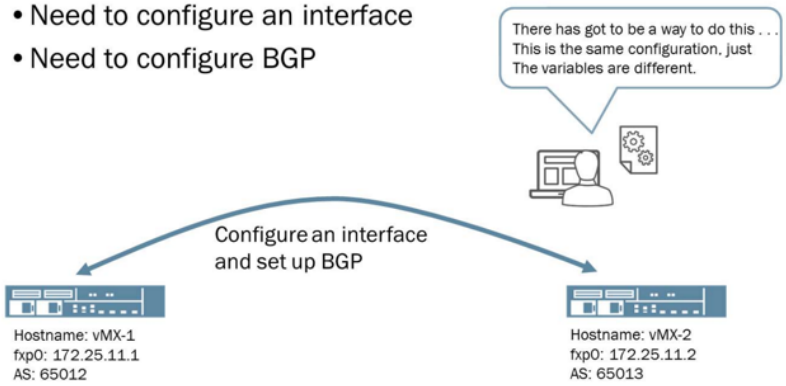
- Jinja2 Comparison Operators

Operator	Description
==	Compares two objects for equality.
!=	Compares two objects for inequality.
>	true if the left hand side is greater than the right hand side.
>=	true if the left hand side is greater or equal to the right hand side.
<	true if the left hand side is less than the right hand side.
<=	true if the left hand side is less than or equal to the right hand side.

- Jinja2 Logic and Other Operators

Logic	Description
and	Return true if the left and the right operand are true
or	Return true if the left or the right operand are true
not	negate a statement
(expr)	group an expression
Other	**Description**
in	Perform a sequence / mapping containment test. Returns true if the left operand is contained in the right. {{ 1 in [1, 2, 3] }} would, for example, return true
is	Performs a test
|	Applies a filter
~	Converts all operands into strings and concatenates them

- Creating a Junos PyEZ, YAML, and Jinja2 Solution
  - Jinja2 Case Study
    -
      - The case study scenario:
        - Routers have management access to fxp0
        - Need to configure an interface
        - Need to configure BGP



    - Identify relevant(liên quan) parts of configuration
      -

```
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 172.17.1.2/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 192.168.2.1/32;
            }
        }
    }
}
```

```
routing-options {
    static {
        route 0.0.0.0/0 next-hop 172.25.11.254;
    }
    autonomous-system 65513;
}
protocols {
    bgp {
        group ext-peers {
            type external;
            peer-as 65512;
            neighbor 172.17.1.1;
        }
    }
}
```

- Create the template

```
interfaces {
    {{bgp.interface}} {
        unit {{bgp.unit_num}} {
            family inet {
                address {{bgp.address}}/{{bgp.address_cidr}};
            }
        }
    }
    lo0 {
        unit {{loopback.unit_num}} {
            family inet {
                address {{loopback.address}}/32;
            }
        }
    }
}
```

```
routing-options {
    static {
        route 0.0.0.0/0 next-hop 172.25.11.254;
    }
    autonomous-system {{bgp.as_num}};
}
protocols {
    bgp {
        group ext-peers {
            type external;
            peer-as {{bgp.peer_as}};
            neighbor {{bgp.neighbor_ip}};
        }
    }
}
```

- Save the completed file as case1.j2

- Build the YAML file for vMX-1

```yaml
---
bgp:
    interface: ge-0/0/0
    unit_num: 0
    address: 172.17.1.1
    address_cidr: 24
    as_num: 65012
    peer_as: 65013
    neighbor_ip: 172.17.1.2

loopback:
    unit_num: 0
    address: 192.168.2.1
```

  - Save file as vMX-1.yml

- Build the YAML file for vMX-2

```yaml
---
bgp:
    interface: ge-0/0/0
    unit_num: 0
    address: 172.17.1.2
    address_cidr: 24
    as_num: 65013
    peer_as: 65012
    neighbor_ip: 172.17.1.1

loopback:
    unit_num: 0
    address: 192.168.2.2
```

  - Save file as vMX-2.yml

- Create the Juno PyEZ script

```python
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
from jnpr.junos.exception import *
from jinja2 import Template
import yaml
import sys

junos_hosts = [ 'vMX-1', 'vMX-2']
for host in junos_hosts:
    try:
        # Open and read the YAML file.
        myFile = host + '.yml'
        with open(myFile,'r') as fh:
            data = yaml.load(fh.read())
        # Open and read the Jinja2 template file.
        with open('case1.j2','r') as t_fh:
            t_format = t_fh.read()
```

```
18      # Associate the t_format template with the Jinja2 module
19          template = Template(t_format)
20          # Merge the data with the template
21          myConfig = template.render(data)
22
23          dev = Device(host=host, user='lab', password='lab123')
24          dev.open()
25          config = Config(dev)
26          config.lock()
27          config.load(myConfig, merge=True, format="text")
28          config.pdiff()
29          config.commit()
30          dev.close()
31      except LockError as e:
32          print "The config database was locked!"
33      except ConnectTimeoutError as e:
34          print "Connection timed out!"
```

- Return results

Script Output vMX-1

```
[edit interfaces]
+   ge-0/0/0 {
+       unit 0 {
+           family inet {
+               address 172.17.1.1/24;
+           }
+       }
+   }
+   lo0 {
+       unit 0 {
+           family inet {
+               address 192.168.2.1/32;
+           }
+       }
+   }
[edit]
+  routing-options {
+      static {
+          route 0.0.0.0/0 next-hop 172.25.11.254;
+      }
+      autonomous-system 65012;
+  }
+  protocols {
+      bgp {
+          group ext-peers {
+              type external;
+              peer-as 65013;
+              neighbor 172.17.1.2;
+          }
+      }
+  }
```

Script Output vMX-2

```
edit interfaces]
+   ge-0/0/0 {
+       unit 0 {
+           family inet {
+               address 172.17.1.2/24;
+           }
+       }
+   }
+   lo0 {
+       unit 0 {
+           family inet {
+               address 192.168.2.2/32;
+           }
+       }
+   }
[edit]
+  routing-options {
+      static {
+          route 0.0.0.0/0 next-hop 172.25.11.254;
+      }
+      autonomous-system 65013;
+  }
+  protocols {
+      bgp {
+          group ext-peers {
+              type external;
+              peer-as 65012;
+              neighbor 172.17.1.1;
+          }
+      }
+  }
```

- More Information on Jinja2 and Junos PyEZ
  - Information on Jinja2

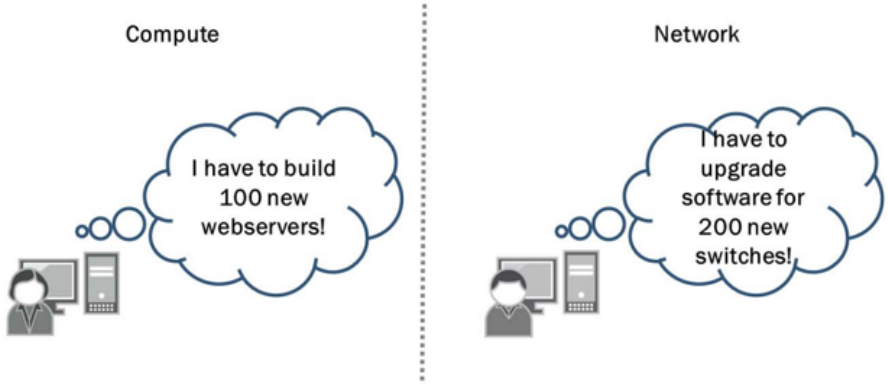
    Introduction — Jinja Documen...
  - Jinja2/YAML/Python examples
  - Junos PyEZ Developer Guide
- Questions/Answers

- 1. Which command is needed to install Jinja2?
  - Use the pip install jinja2 command to install Jinja2.
- 2. What is the syntax to create a comment in Jinja2?
  - Create a comment in Jinja2 by using the following format {#...#}.
- 3. What does set command do?
  - The set command assigns a value to variable.
- 4. What does this statement do? config.pdiff()
  - The config.pdiff() statement shows the difference between the candidate configuration and the active configuration.

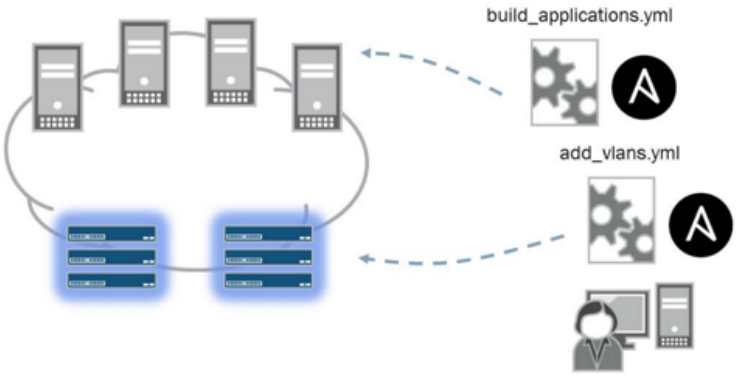
Ansible

- Objectives
  - Describe the format of an Ansible YAML playbook
  - Describe how Ansible interfaces with Junos OS devices
  - Explain how to operate Junos devices using Ansible
- Introduction to Ansible
  - Summary
    - Ansible provides tools to accelerate(nhanh chóng) the deployment of infrastructure
    - It also provides the ability to manage configuration and assist with operations
    - It was initially created for compute and cloud but now has been ported to support network infrastructure
    - Ansiable provides idempotent(lý tưởng) operations
      - Ansible playbooks may be run multiple times to yield the same result
      - Ansible modules only execute a change if required
  - Why use Ansible?
    - The challenge of scaling opreations

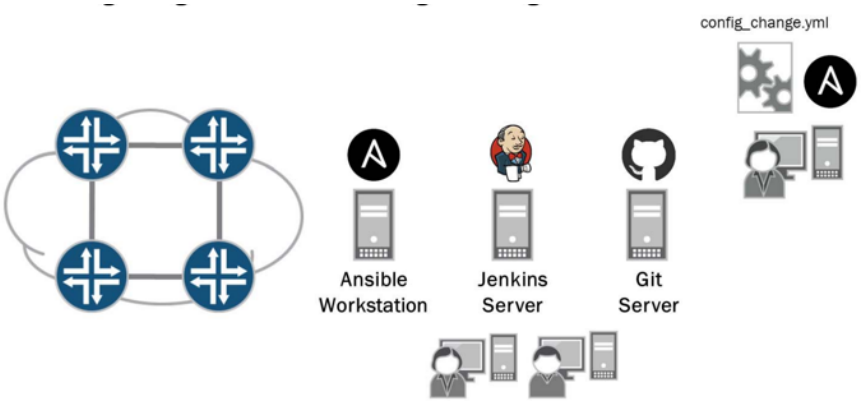
- Ansible uses simple tools and syntax to automate repetitive tasks for both network and compute



- Use Cases
  - Reducing time to deployment
    - Using Ansible tools, networks changes can be automated with application rollouts(triển khai) and server infrastructure updates

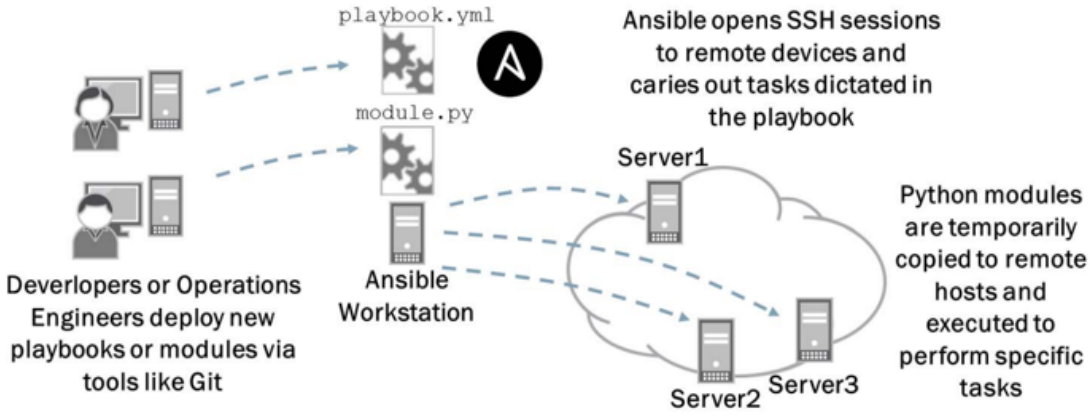


  - Improving change management
    - Reduce the risk associated(liên quan) with infrastructure changes by integrating Ansible into change management workflows



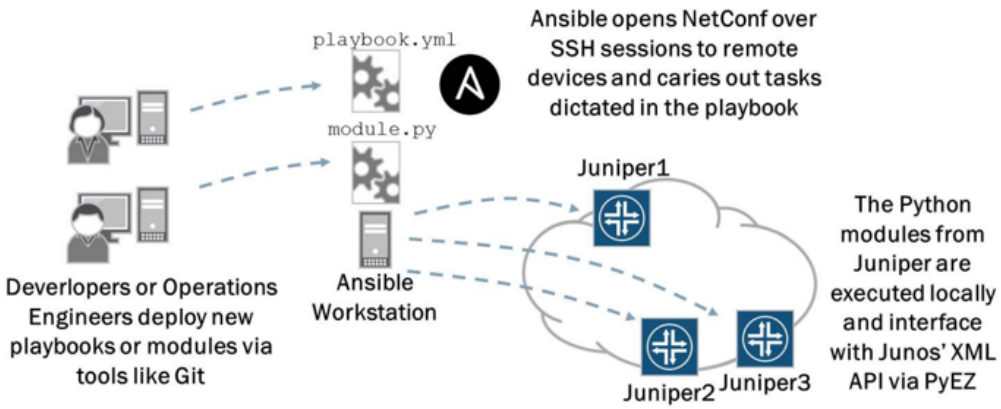
- Normal Ansible Operation

- How does Ansible typically(thường) work?



  - Python modules are copied and executed on each relevant remote device over SSH
  - Playbooks written in YAML carry out the tasks defined in Python modules

- Ansible Operation with Junos
  - How does Ansible work with Junos?



    - Junos platforms modules are executed locally and leverage(tận dụng) PyEZ

- Building a Basic Ansible Environment
  - Install
    - Python 2.7
    - Install PyEZ latest version
    - Install Ansible latest version
      - Using pip - pip install ansible
      - Refs

        Installing Ansible — Ansible D...

- Install Junos Ansible Galaxy role

```
(jaut_env) [lab@jaut-desktop ~]$ ansible-galaxy install
Juniper.junos -p ~/virtualenv/jaut_env/
- downloading role 'junos', owned by Juniper
- downloading role from https://github.com/Juniper/ansible-
junos-stdlib/archive/1.4.2.tar.gz
- extracting Juniper.junos to
/home/lab/virtualenv/jaut_env/Juniper.junos
- Juniper.junos (1.4.2) was installed successfully
```

- Config NETCONF on the Junos OS device

```
root@R1> show configuration system services
ssh;
netconf {
    ssh;
}
web-management {
    http {
        interface ge-0/0/0.0;
    }
}
```

- Ansible "Hello World!"

hello_world.yml

```
---
  - name: Hello World!
    hosts: vsrx
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    vars_prompt:
     - name: USERNAME
       prompt: User name
       private: no
     - name: DEVICE_PASSWORD
       prompt: Device password
       private: yes

    tasks:
    - name: Get Junos device information
      junos_get_facts:
        host={{ inventory_hostname }}
        user={{ USERNAME }}
        passwd={{ DEVICE_PASSWORD }}
      register: junos

    - name: Print Junos facts
      debug: msg="{{junos.facts}}"
```

Play name.

Group of hosts.

A play is the grouping of hosts to tasks. This playbook contains a single play. Multiple plays are allowed per playbook.

Use the prompt to gather username and password for the Junos device.

This play executes two tasks. The first retrieves device facts. The second prints them to the terminal.

-

Terminal Output

```
bash $ ansible-playbook hello_world.yml
User name: root
Device password:

PLAY [Hello World!] ************************************************************

TASK: [Get Junos device information] *******************
ok: [192.168.64.51]

TASK: [Print Junos facts] ******************************************************
ok: [192.168.64.51] => {
    "msg": "{u'domain': None, u'serialnumber': u'b34235e6bda7', u'ifd_style': u'CLASSIC',
u'version_info': {u'major': [12, 1], u'type': u'X', u'build': 5, u'minor': [46, u'D', 20]},
u'RE0': {u'status': u'Testing', u'last_reboot_reason': u'Router rebooted after a normal
shutdown.', u'model': u'FIREFLY-PERIMETER RE', u'up_time': u'4 days, 6 hours, 7 minutes, 17
seconds'}, u'hostname': u'R1', u'fqdn': u'R1', u'virtual': True, u'has_2RE': False,
u'switch_style': u'NONE', u'version': u'12.1X46-D20.5', u'srx_cluster': False, u'HOME':
u'/cf/root', u'model': u'FIREFLY-PERIMETER', u'personality': u'SRX_BRANCH'}"
}

PLAY RECAP *********************************************************************
192.168.64.51              : ok=2    changed=0    unreachable=0    failed=0
```

Use the `ansible-playbook` utility and pass the playbook name as an argument to execute.

Type out root user password for the Junos device.

The u simply indicates Unicode encoding. It may be safely ignored.

Two tasks completed. No changes to any hosts.

- The Ansible Command Line
  - Ansible has a number of CLI utilities to operate infrastructure
    - `ansible`
      - Allows you to run a module or basic commands on another host without playbooks
    - `ansible-doc`
      - Shows relevant documentation for modules
    - `ansible-galaxy`
      - Creates role skeletons or downloads roles from the Galaxy repository
    - `ansible-playbook`
      - Executes a playbook passed as an argument on remote hosts via push operations
    - `ansible-pull`
      - Executes plays on a local host via pull operations
    - `ansible-vault`
      - Creates and manages an encrypted password “vault” that saves SSH passwords
- Secure Automated Access to Junos
  - With Ansible, as seen on the previous slide, we may prompt users for username and password to get up and running quickly

```
bash$ ssh-agent bash
.
.
bash$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /home/ansible-guru/.ssh/id_rsa:
.
.
```

Enter password if needed.

    - We may also exchange public keys then follow the steps below to setup an SSH
    - Alternatively, you can also add the SSH key to the inventory file or a playbooks as an Ansible variable or user an Ansible vault

- The environment impacts which arguments are required when connecting to Junos devices

hello_world.yml

```
- name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
        user={{ USERNAME }}
        passwd={{ DEVICE_PASSWORD }}
      register: junos
```

Without transferring SSH certificates, `user` and `passwd` must be arguments for all tasks/

hello_world_certs.yml

```
- name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos
```

After transferring certificates, if the `USER` environment variable is adequate, both `user` and `passwd` arguments may be left off.

Ansible Command Line

- You can use Ansible to quickly run plays from the CLI without playbooks

Terminal Output

```
bash$ ansible -c local 192.168.64.51 -m junos_get_facts -M
/etc/ansible/roles/Juniper.junos/library/ -a "host={{inventory_hostname}}"
192.168.64.51 | success >> {
    "changed": false,
    "facts": {
        "HOME": "/cf/var/home/lab",
        "RE0": {
            "last_reboot_reason": "Router rebooted after a normal shutdown.",
            "model": "FIREFLY-PERIMETER RE",
            "status": "Testing",
            "up_time": "4 hours, 49 minutes, 1 second"
        },
        "domain": null,
        "fqdn": "R1",
        "has_2RE": false,
        "hostname": "R1",
        "ifd_style": "CLASSIC",
        "model": "FIREFLY-PERIMETER",
        .
        .
        .
        .
```

The host must reside in the inventory file and the proper certificates must be copied to the Junos host in this scenario.

Data is stored and passed via JSON formatting.

Ansible Command Line Arguments

- Ansible also uses arguments to control operations

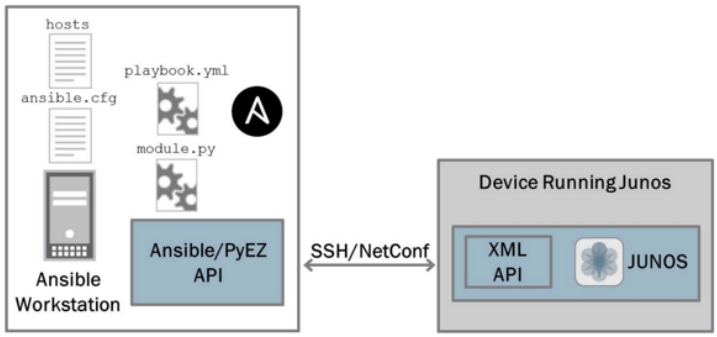
Terminal Output

```
bash $ ansible-playbook hello_world.yml --check --diff
User name: root
Device password:

PLAY [Hello World!] ************************************************************

TASK: [Retrieve information from devices running Junos OS] *******************

ok: [192.168.64.51]

TASK: [Print Junos facts] ****************************************************

ok: [192.168.64.51] => {
    "msg": "{u'domain': None, u'serialnumber': u'b34235e6bda7', u'ifd_style': u'CLASSIC',
u'version_info': {u'major': [12, 1], u'type': u'X', u'build': 5, u'minor': [46, u'D', 20]},
u'RE0': {u'status': u'Testing', u'last_reboot_reason': u'Router rebooted after a normal
shutdown.', u'model': u'FIREFLY-PERIMETER RE', u'up_time': u'2 days, 12 hours, 56 minutes, 52
seconds'}, u'hostname': u'R1', u'fqdn': u'R1', u'virtual': True, u'has_2RE': False,
u'switch_style': u'NONE', u'version': u'12.1X46-D20.5', u'srx_cluster': False, u'HOME':
u'/cf/root', u'model': u'FIREFLY-PERIMETER', u'personality': u'SRX_BRANCH'}"
}

PLAY RECAP ********************************************************************
192.168.64.51              : ok=2    changed=0    unreachable=0    failed=0
```

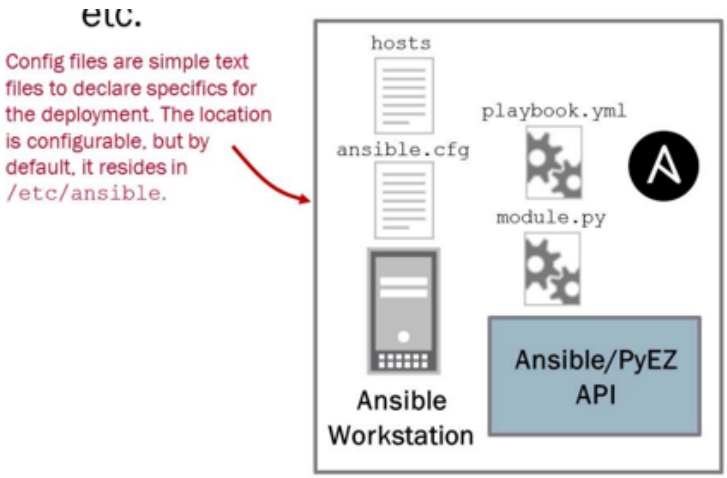
The `--check` argument does a dry run and will not execute changes.
The `--diff` displays any relevant changes that would occur.

Create Ansible Playbook

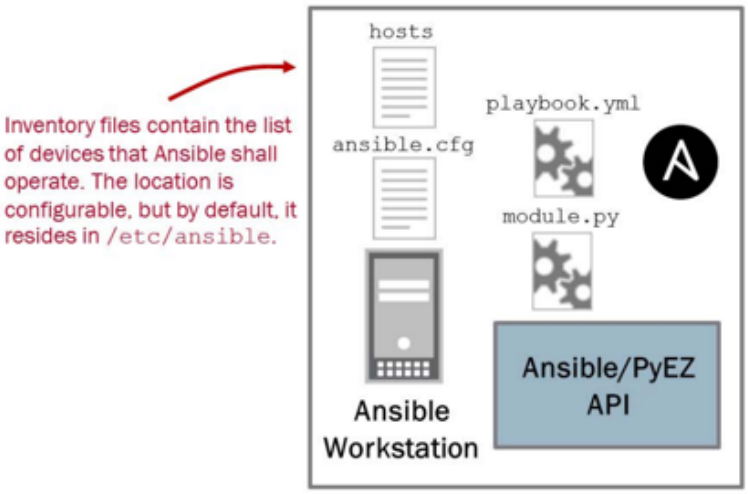
- Ansible and Junos Integration



- Ansible Configuration File
  - Ansible has a configuration file specify certain parameters for operation
    - Example: SSH username, SSH port, directory module, etc..

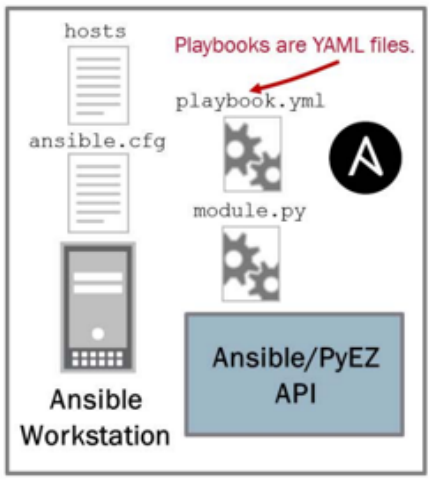


- Ansible Inventory File
  - Ansible has an inventory file called hosts which Ansible check by default
    - You can create custom inventory files and pass them as arguments to Ansible



- Overview of Playbooks
  - A playbook is a unit of work represented in YAML

- We can create playbooks to carry out plays or series of tasks on multiple hosts



- Different type of Ansible modules
  - Ansible module library
    - Officially supported by Ansible and the community, follow Ansible best practices and guidelines
    - Supported in Ansible Tower
    - Ships with Ansible
  - Ansible Galaxy
    - Appstore for Ansible, anyone is free to write and publish a module
    - Separate installation is required
- Junos Ansible Modules
  - Ansible Library and Ansible Galaxy Modules

Ansible Library	
junos_command	N
junos_config	N
junos_netconf	N
junos_package	N
junos_facts	N
Junos_rpc	N
Junos_template	N
Junos_user	N

Ansible Galaxy / Juniper.junos	
junos_get_facts	N, C
junos_install_config	N, C
Junos_ping	N
Junos_get_tables	N
junos_zeroize	N, C
junos_srx_cluster	N, C
junos_install_os	N
junos_shutdown	N
junos_get_config	N
junos_rollback	N
junos_commit	N
junos_rpc	N
junos_cli	N

N: Netconf
N, C: Netconf & Console

- Ansible Playbook Template
  - Ansible playbook template for Juniper devices

- Now that we have covered Ansible, we shall begin detailing how to produce playbooks and automate tasks
- Ansible has a number built-in features
- The below illustrates a template that may be used to start creating Ansible playbooks for Juniper tasks

juniper_template.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no
```

Create a name for the play.
Identify the host group from the inventory file.
Import the `Juniper.junos` role to access relevant modules.
Specify to execute locally. Recall we must use this option since we are not executing modules on the remote devices. They are executed locally using the PyEZ API which interfaces with the XML API of Junos. Since we are running locally, there is no reason to gather facts.

Create Ansible Playbooks

Tasks

playbook_example_1.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:
    - name: Checking NETCONF connectivity
      wait_for: host={{ inventory_hostname }} port=830 timeout=5
```

In this example, the task simply checks to make sure port 830 (NetConf) is reachable.
The `tasks:` section contains a list of operations to perform
Naming each task simplifies debug

- Tasks execute an operation for a playbook
- Playbook can contain multiple tasks, but a task may only perform a single operation

Variable substitution

- To reference variables, we may use basic Jinja2 syntax
- In the example below, 'inventory_hostname' is a reserved variable containing the hostname of IP address from the inventory

playbook_example_1.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:
    - name: Checking NETCONF connectivity
      wait_for: host={{ inventory_hostname }} port=830 timeout=5
```

We can substitute variables using Jinja2 syntax where double brackets surround the variable name.

- Saving results to variables

playbook_example_2.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:
    - name: Checking NETCONF connectivity
      wait_for: host={{ inventory_hostname }} port=830 timeout=5

    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos
```

Using `register:` we can save the results of commands to variables.

  - The 'register:' keyword allows us to save results from tasks to variable

- Conditions

playbook_example_3.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:
    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos

    - name: Reboot select devices
      junos_shutdown:
        host={{ inventory_hostname }}
        shutdown="shutdown"
        reboot="yes"
      when: junos.facts.hostname == "R1"
```

We can use the `when:` conditional like an `if`

  - The 'when:' keyword operates like an 'if' statement
  - A task is only execute if the 'when:' returns true

- Loops

playbook_example_4.yml

```
---
  - name: Play Name
    hosts: juniper_hosts
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:

    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos

    - name: Loop over a dictionary
      debug: msg={{ 'host-' + junos.facts.hostname }}
      when: junos.facts.hostname == item.host
      with_items:
        - { host: 'R1' }
        - { host: 'R2' }
```

We can use the `with_items:` keyword like a `for-each` loop from other languages. Rather than executing for all hosts, we can iterate over a specific host using `with_items:`.

  - The 'with_items:' keyword operates like a for-each loop

- The junos_shutdown Module
  - Example

- The junos_shutdown module allows Ansible to shutdown or reboot remote devices

reboot_hosts.yml

```
---
  - name: Reboot Hosts
    hosts: vsrx
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:

    - name: Reboot select devices
      junos_shutdown:
        host={{ inventory_hostname }}
        shutdown="shutdown"
        reboot="yes"
```

The junos_shutdown module allows us to reboot or shutdown junos devices.

Required to shutdown/reboot devices.

By default, without the reboot argument, the Junos device shall be powered off.

- The CLI output below shows the execution of junos_shutdown module

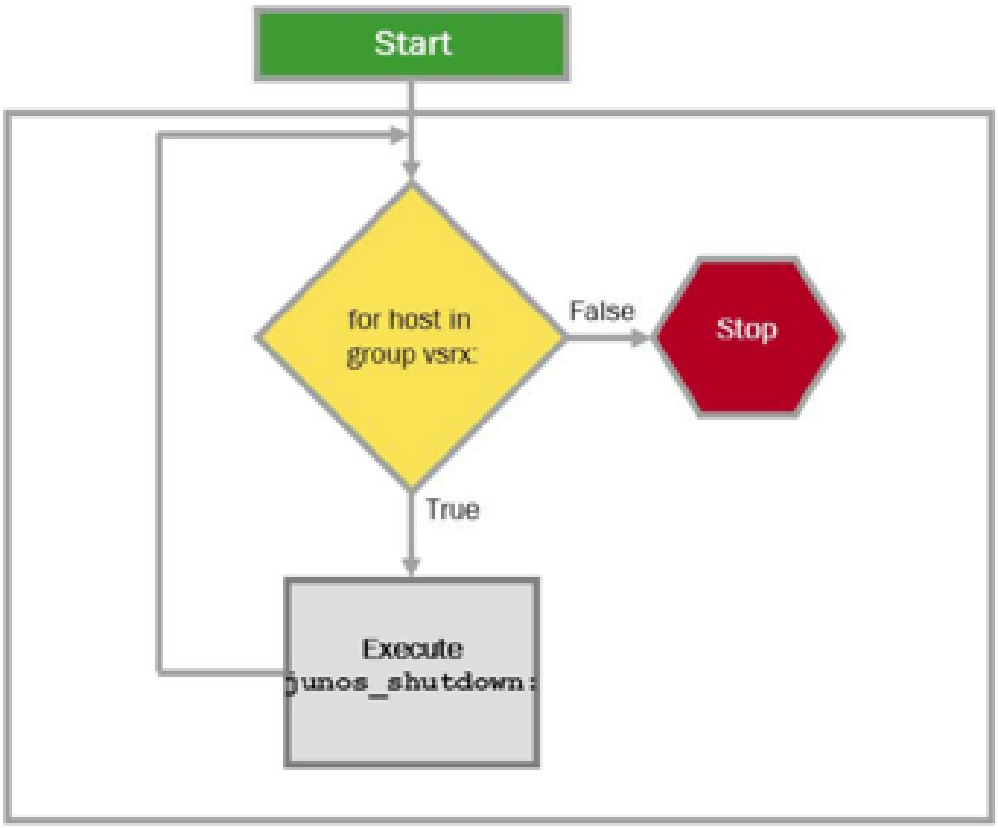
Terminal Output

```
bash $ more /etc/ansible/hosts
# Ansible Operating vSRX

[vsrx]

192.168.64.51
192.168.64.52


bash $ ansible-playbook reboot_hosts.yml

PLAY [Reboot Hosts] ************************************************************

TASK: [Reboot select devices] **************************************************
changed: [192.168.64.51]
changed: [192.168.64.52]

PLAY RECAP *********************************************************************
192.168.64.51              : ok=1    changed=1    unreachable=0    failed=0
192.168.64.52              : ok=1    changed=1    unreachable=0    failed=0
bash $
```

The inventory file has two hosts in the vsrx group.

On both devices, both tasks completed and caused a single change as a result of the reboot.

- The flow chart for the playbook running junos_shutdown



- The junos_get_facts Module
  - The junos_get_facts module allows Ansible to retrieve information from Junos devices

get_facts.yml

```
---
  - name: Get Facts
    hosts: vsrx
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:

    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos

    - name: Print Junos software version
      debug: msg="{{junos.facts.version}}"
```

The junos_get_facts module allows us to retrieve facts from Junos devices.

The register key word is used to save the result of a task to a variable.

The debug: module is meant for printing output during playbook runs.

- The CLI output below shows the execution of junos_get_facts module

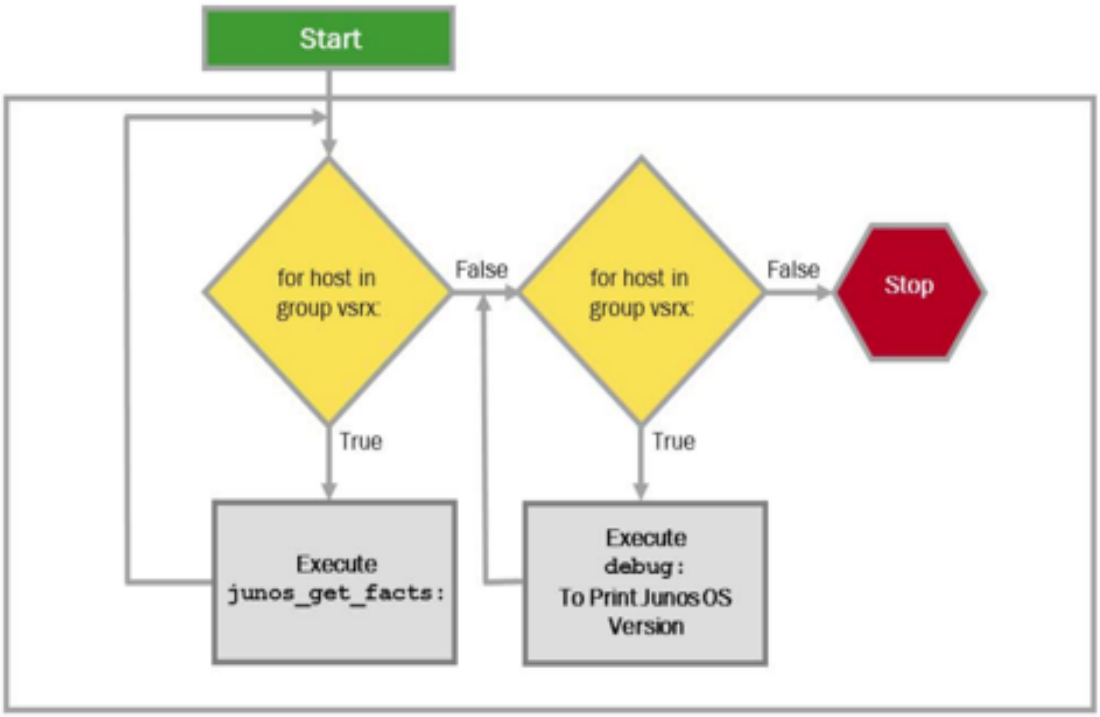
Terminal Output

```
bash $ ansible-playbook get_facts.yml

PLAY [Get Facts] ***************************************************************

TASK: [Retrieve information from devices running Junos OS] *******************
ok: [192.168.64.51]
ok: [192.168.64.52]

TASK: [Print Junos software version] *****************************************
ok: [192.168.64.52] => {
    "msg": "12.1X46-D20.5"
}
ok: [192.168.64.51] => {
    "msg": "12.1X46-D20.5"
}

PLAY RECAP *********************************************************************
192.168.64.51              : ok=2    changed=0    unreachable=0    failed=0
192.168.64.52              : ok=2    changed=0    unreachable=0    failed=0
```

Illustrates the output from the debug: task in the playbook.

On both devices, both tasks completed, but no changes were required since we only gathered tasks.

- The flow chart for the playbook running junos_get_facts



- The junos_install_config Module

- The junos_install_config module allows Ansible to install Junos configuration templates

load_config.yml

```
---
  - name: Install Junos Config
    hosts: vsrx
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no

    tasks:
    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos

    - name: Install Junos configuration template
      junos_install_config:
        host={{ inventory_hostname }}
        file={{ item.config }}
        diffs_file={{ item.host }}
      when: junos.facts.hostname == item.host
      with_items:
        - { host: 'R1', config: 'bgp-config-1.conf' }
        - { host: 'R2', config: 'bgp-config-2.conf' }
```

The junos_install_config module allows for installation of configuration templates.

Install the config file template for the proper host.

The diffs_file argument allows us to save the output of a "show | compare" operation to a file.

- The junos_install_config module shall install the Junos configuration templates illustrated

Terminal Output

```
bash $ more bgp-config-1.conf
protocols {
    bgp {
        group ANSIBLE-DEPLOY {
            type internal;
            local-address 10.0.0.1;
            log-updown;
            family inet {
                unicast;
            }
            neighbor 10.0.0.2;
            neighbor 10.0.0.3;
            neighbor 10.0.0.4;
            neighbor 10.0.0.5;
        }
    }
}
```

Config template for R1.

Terminal Output

```
bash $ more bgp-config-2.conf
protocols {
    bgp {
        group ANSIBLE-DEPLOY {
            type internal;
            local-address 10.0.0.2;
            log-updown;
            family inet {
                unicast;
            }
            neighbor 10.0.0.1;
            neighbor 10.0.0.3;
            neighbor 10.0.0.4;
            neighbor 10.0.0.5;
        }
    }
}
```

Config template for R2.

- The CLI output below shows the execution of junos_install_config module

Terminal Output

```
bash $ ansible-playbook load_config.yml

PLAY [Install Junos Config] ****************************************************

TASK: [Retrieve information from devices running Junos OS] *********************
ok: [192.168.64.51]
ok: [192.168.64.52]

TASK: [Install Junos configuration template] ***********************************
skipping: [192.168.64.52] => (item={'host': 'R1', 'config': 'bgp-config-1.conf'})
changed: [192.168.64.51] => (item={'host': 'R1', 'config': 'bgp-config-1.conf'})
skipping: [192.168.64.51] => (item={'host': 'R2', 'config': 'bgp-config-2.conf'})
changed: [192.168.64.52] => (item={'host': 'R2', 'config': 'bgp-config-2.conf'})

PLAY RECAP *********************************************************************
192.168.64.51              : ok=2    changed=1    unreachable=0    failed=0
192.168.64.52              : ok=2    changed=1    unreachable=0    failed=0
```

The configuration loops over the two hashes identified by the `with_items:` keyword. The `when:` statement allows us to apply configurations to specific hosts.

On both devices, both tasks completed, but one change is completed per router.

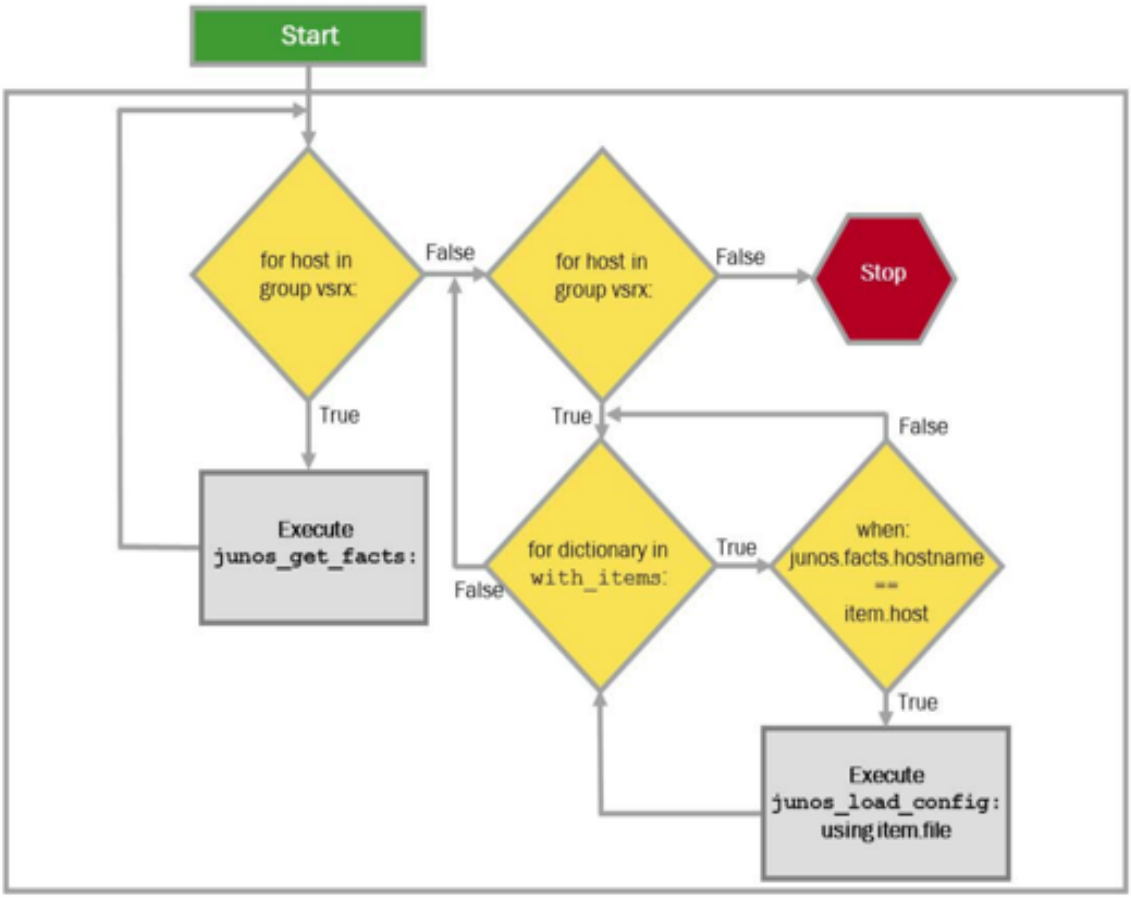
- The junos_install_config module can produce a log file illustrating the configuration change applied

Terminal Output

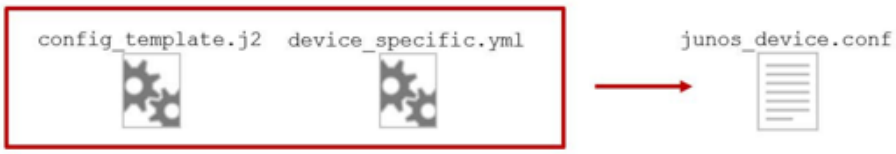
```
bash $ ls | grep R
R1
R2
bash $ more R1

[edit protocols]
+   bgp {
+       group ANSIBLE-DEPLOY {
+           type internal;
+           local-address 10.0.0.2;
+           log-updown;
+           family inet {
+               unicast;
+           }
+           neighbor 10.0.0.1;
+           neighbor 10.0.0.3;
+           neighbor 10.0.0.4;
+           neighbor 10.0.0.5;
+       }
+   }
```

The `diffs_file:` argument allows for review of changes after running the playbook.

- The flow chart for the playbook



- Advanced Configuration Deployment
  - Introducing Jinja2



    - Jinja2 is template language commonly used with Python
    - Using Jinja2 templates and YAML files containing variable definitions, we can automate creating common configurations
    - Also, Ansible uses some Jinja2 syntax in playbooks for conditionals and variable replacement
  - Deploying Jinja2 Template

- The Jinja2 template below shall be used to generate a single Junos configuration to install via Ansible

bgp_jinja_template.j2

```
{% for bgp_vars in item.itervalues() %}
protocols {
    bgp {
        group {{ bgp_vars.bgp_group_name }} {
            type internal;
            local-address {{ bgp_vars.local_address }};
            log-updown;
            family inet {
                unicast;
            }
{% for host, ip in bgp_vars.neighbors.iteritems() %}
            neighbor {{ ip }};
{% endfor %}
        }
    }
}
{% endfor %}
```

Using the {{}} notation, we can perform variable substitution. `bgp_vars` is the dictionary of key value pairs.

We can also embed `for` loops. The only syntax difference from Python, is the `endfor` statement.

deploy_bgp.yml

```
---
bgp_data:
- R1:
    bgp_group_name: ANSIBLE-DEPLOY
    local_address: 10.0.0.1
    neighbors:
      r2: 10.0.0.2
      r3: 10.0.0.3
      r4: 10.0.0.4
      r5: 10.0.0.5
```

When the YAML file is passed to the Jinja2 template, the first `for` loop allows us to iterate over the list of routers.

In the Jinja2 template, `bgp_vars` is a dictionary holding the (key, value) pairs.

-

load_jinja_configs.yml

```
---
  - name: Load Configs From Jinja2 Templates
    hosts: vsrx
    roles:
     - Juniper.junos
    connection: local
    gather_facts: no
    vars:
      - junos_jinja_template: "bgp_jinja_template.j2"
      - path: "junos_configs"
    vars_files:
      - "junos_yaml_templates/deploy_bgp.yml"

    tasks:

    - name: Retrieve information from devices running Junos OS
      junos_get_facts:
        host={{ inventory_hostname }}
      register: junos

    - name: Create Junos configurations from Jinja2
      template: src={{ junos_jinja_template }} dest="{{ path }}/{{ junos.facts.hostname }}.conf"
      with_items: bgp_data

    - name: Install Junos configuration template
      junos_install_config:
        host={{ inventory_hostname }}
        file="{{ path }}/{{ junos.facts.hostname }}.conf"
        diffs_file="{{ junos.facts.hostname }}.log"
```

We shall use the vars: keyword to reference the Jinja2 template file and directory path to target configurations by variable.

We can use the `vars_files:` keyword to identify our YAML template to generate the config from Jinja2.

After importing `deploy_bgp.yml`, we can iterate over the variables contents using the `with_items:` keyword to produce the target configuration from the Jinja2 template.

Juniper Ansible modules are executed locally. As such, we must explicitly run the `junos_install_config:` to install the generated template.

- The 'template': module in Ansible playbook from the previous slide produces the Junos configuration below

Terminal

```
bash$ more junos_configs/R1.conf
protocols {
    bgp {
        group ANSIBLE-DEPLOY {
            type internal;
            local-address 10.0.0.1;
            log-updown;
            family inet {
                unicast;
            }
            neighbor 10.0.0.4;
            neighbor 10.0.0.5;
            neighbor 10.0.0.2;
            neighbor 10.0.0.3;
        }
    }
}
```

deploy_bgp.yml

```
---
bgp_data:
- R1:
    bgp_group_name: ANSIBLE-DEPLOY
    local_address: 10.0.0.1
    neighbors:
      r2: 10.0.0.2
      r3: 10.0.0.3
      r4: 10.0.0.4
      r5: 10.0.0.5
```

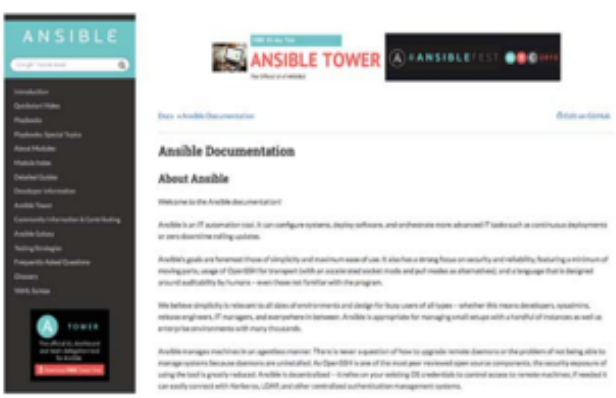
- After producing the Junos configuration, the 'junos_install_config': module installs the configuration file

Terminal Output

```
bash $ ansible-playbook load_jinja_configs.yml

PLAY [Load Configs From Jinja2 Templates] ***********************************

TASK: [Retrieve information from devices running Junos OS] ******************
ok: [192.168.64.51]

TASK: [Create Junos configurations from Jinja2] *****************************
changed: [192.168.64.51] => (item={'R1': {'neighbors': {'r4': '10.0.0.4', 'r5': '10.0.0.5', 'r2':
'10.0.0.2', 'r3': '10.0.0.3'}, 'bgp_group_name': 'ANSIBLE-DEPLOY', 'local_address': '10.0.0.1'}})

TASK: [Install Junos configuration template] ********************************
changed: [192.168.64.51]

PLAY RECAP ******************************************************************
192.168.64.51              : ok=3    changed=2    unreachable=0    failed=0
```

Three tasks completed and two changes executed (template generation and config change).

- Additional References
  - You are encouraged to review other resources to become proficient(thành thạo) at Ansible, YAML and Jinja2



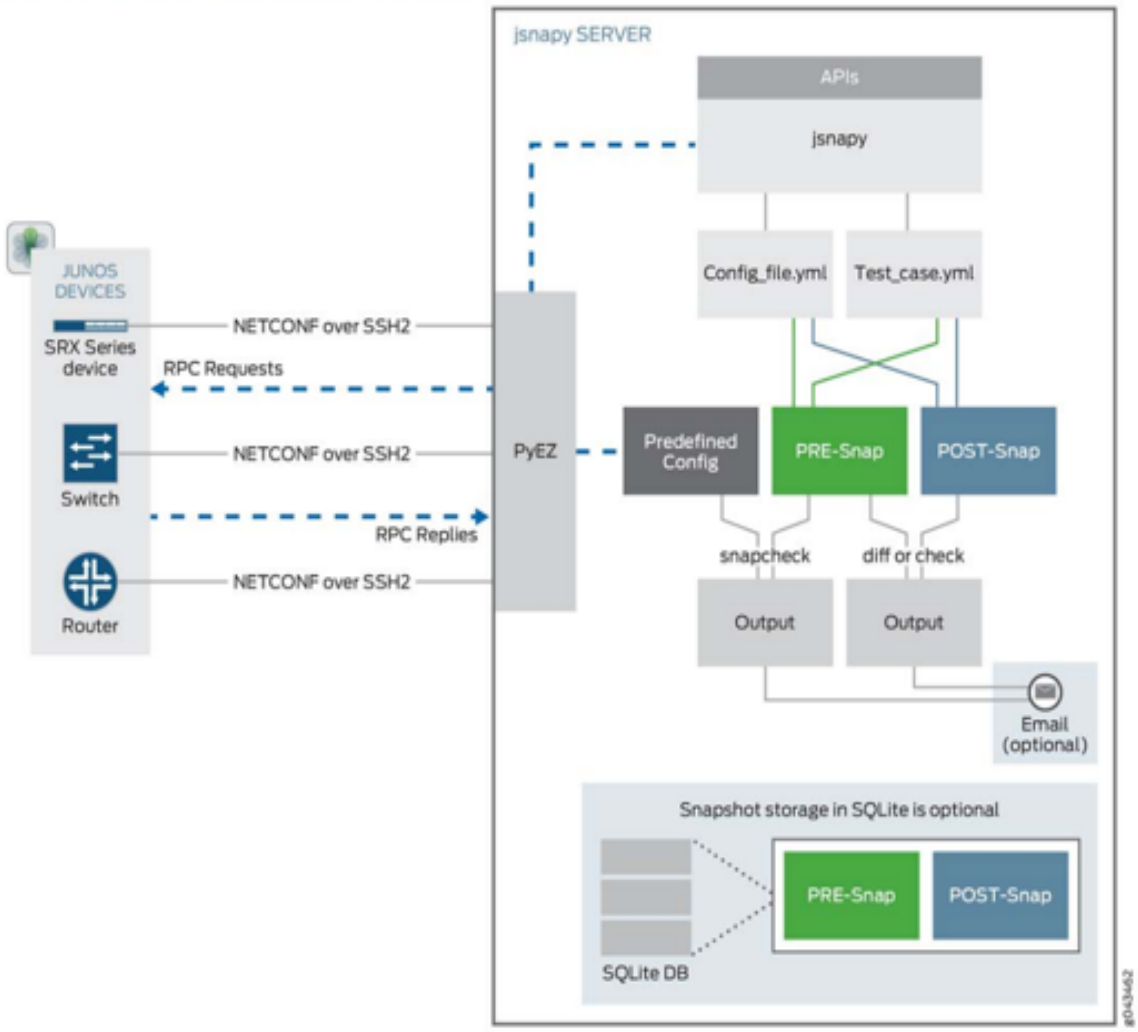
- Additional useful Ansible module and builtin features
  Ansible Documentation
- Detailed error handling
- Producing Jija2 templates
  Jinja — Jinja Documentation (...

- Question and Anwsers
  - What is a playbook?
    - A playbook contains a list of tasks for Ansible to perform. When we want to automate repetitive tasks using Ansible, we create playbooks to iterate over hosts to complete common tasks.
  - What is the required playbooks format?
    - Ansible playbooks are written in YAML. However, tasks are executed using Python modules, so some syntax looks similar to Python and Python's templating language: Jinja2.
  - How can you control flow in a playbook?
    - There are number of ways to perform flow control in Ansible. We discussed the usage of the conditional when: which operates like an if statement and the loop with_items: which operates like a for-each loop.
  - What is an example scenario where Jinja2 templates may be used with Ansible?
    - Jinja2 templates can be used to automatically generate Junos configurations. We can perform variable substitution to generate device specific configuration files.

- **JSNAPy**
  - Objectives
    - Describe how JSNAPy can help automate Junos
    - Install JSNAPy
    - Use JSNAPy to create snapshots
  - JSNAPy Overview and Installation
    - Automates Network Verifications
      - Create Snapshots
        - Take snapshots of Junos OS configuration files
        - Take snapshots of Junos RPC call results

- Analyze Snapshots
  - Compare a snapshot to a standard
  - Compare a pre change snapshot with a post changes snapshot
  - Get the different between two snapshots
- Store snapshots in text files or SQLite database
- Email snapshot results to adminstrators
- Embed JSNAPy into Python Scripts

- JSNAPy Architecture



- JSNAPy Prerequisites
  - JSNAPy Server
    - Linux or Mac OS (not Windows)
      - OS Dependencies seen below
      - Same dependencies as PyEZ
    - Python 2.6 or later

- Optional SQLite
- Device Running Junos OS
  - Management Port Reachability
  - NETCONF Enabled

```
[edit system services]
lab@vMX-1# set netconf ssh
```

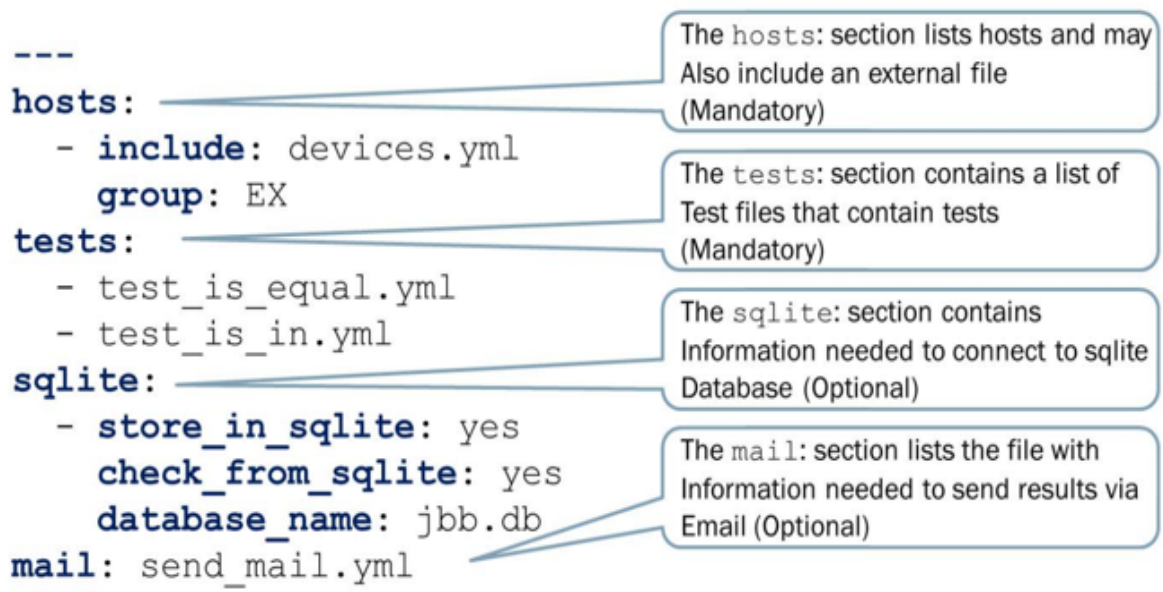
- Installing JSNAPy
  - Install using pip
    -

```
user@jsnapy-server:~> sudo pip install jsnapy
user@jsnapy-server:~> sudo pip install
git+https://github.com/Juniper/jsnapy.git
```

  - Update to the latest version with the -U option
    -

```
user@jsnapy-server:~> sudo pip install jsnapy -U
user@jsnapy-server:~> sudo pip install
git+https://github.com/Juniper/jsnapy.git -U
```

  - Download or clone source code from the git repository
    - git clone https://github.com/Juniper/jsnapy
- Create JSNAPy Configuration Files
  - JSNAPy configuration files are written in YAML and include the follow four parts



  - Configuration File hosts Section

- The `hosts`: section can specify one or more hosts

```
Hosts:
  - device: 172.25.11.1
        username: lab
        passwd: lab123

  - device: 172.25.11.2
        username: lab
        passwd: lab123

  - include:  devices.yml
        group: MX
```

You can include one or more devices

You can also import a YAML file with a list of devices Organized in groups

- Sample devices.yml File

```
MX:
  - 10.20.1.20:
      username: root
      passwd: root123
  - 10.21.13.14:
      username: root
      passwd: root123
EX:
  - 10.2.15.210:
      username: root
      passwd: root123
  - 10.9.16.22:
      username: abc
      passwd: pqr
```

You can include one or more devices per group

- Configuration File tests Section
  - The `tests`: section gives a list of one or more YAML files that contain tests

```
tests:
  - test_not_less.yml
  - test_not_more.yml
  - one_more_test.yml
```

You can have one or many tests

- Configuration File mail and sqlite Sections
  - The `mail`: section lists the YAML file that contains the email settings

```
mail: send_mail.yml
```

- Sample send_mail.yml File

```
to: foo@gmail.com
from: bar@gmail.com
sub: "Sample Jsnapy Results, please verify"
recipient_name: Foo
passwd: 123
server: smtp.gmail.com
port: 587
sender_name: "Juniper Networks"
```

  - The mail server and port sections are used only if you to override the defaults

- The `sqlite`: section lists the YAML file that contains three key value pairs with information necessary to connect to the SQLite Database

```
sqlite:
  - store_in_sqlite: yes
      check_from_sqlite: yes
      database_name: jsnapy.db
```

- Create JSNAPy Test Files

  - The test_sw_version Test

```
tests_include:
  - test_sw_version

test_sw_version:
  - command: show version
  - item:
      xpath: '//software-information'
      tests:
        - all-same: junos-version
          err: "Test Failed!!! The versions are not the same.
From the PRE snapshot, the version is: <{{pre['junos-
version']}}>. From the POST snapshot, the version is
<{{post['junos-version']}}>!! "
          info: "Test Succeeded!! The Junos OS version is:
<{{post['junos-version']}}>!!!"
```

  - The test_rpc_filtering Test

```
tests_include:
  - test_rpc_filtering

test_rpc_filtering:
  - rpc: get-config
  - kwargs:
      filter_xml: configuration/system/host-name
```

- The test_interfaces_terse Test

```
test_interfaces_terse:
  - rpc: get-interface-information
    kwargs:
      - terse: True
  - iterate:
      xpath: //physical-interface
      tests:
        - is-equal: admin-status, up
          info: "Test Succeeded !! admin-status is equal to
{{pre['admin-status']}} with oper-status {{pre['oper-
status']}}"
          err: "Test Failed !! admin-status is not equal to
up, it is {{pre['admin-status']}} with oper-status
{{pre['oper-status']}}"
```

  - There are many more test operators!

- JSNAPy Test Operators

  -

	Compare Elements or Element Values in Two Snapshots
delta	Compare the change in value of a specified data element, which must be present in both snapshots, to a specified delta. You can specify the delta as an absolute, positive, or negative percentage, or an absolute, positive, or negative fixed value.
list-not-less	Determine if the specified items are present in the first snapshot but are not present in the second snapshot.
list-not-more	Determine if the specified items are present in the second snapshot but are not present in the first snapshot.
no-diff	Compare specified data elements that are present in both snapshots, and verify that the value is the same.

	Operate on Elements with Numeric or String Values
all-same	Check if all content values for the specified elements are the same. Optionally, you can check if all content values for the specified elements are the same as the content value of a reference item.
is-equal	- Test if an XML element string or integer value matches a given value.
not-equal	- Test if an XML element string or integer value does not match a given value.

-

	Operate on Elements with Numeric Values
in-range	Test if the XML element value is within a given numeric range.
not-range	Test if the XML element value is outside of a given numeric range
is-gt	Test if the XML element value is greater than a given numeric value.
is-lt	Test if the XML element value is less than a given numeric value

	Operate on Elements with String Values
contains	Determine if an XML element string-value contains the provided string value.
is-in	Determine if an XML element string-value is included in from a specified list of string values.
not-in	Determine if an XML element string-value is excluded from a specified list of string values.

	Operate on XML Elements
exists	Verify the existence of an XML element in the snapshot.
not-exists	Verify the lack of existence of an XML element in the snapshot.

- Use the JSNAPy Command Line Tool
  - Snapshot File Names

Device Name	Snapshot Name	Test Name
vMX-1	Week1-Snap	OSPF-test BGP-test

    - The snapshot file name is a combination of the device name, the snapshot name and the test name
    - Each test will be in its own file even multiple tests were performed within the same test file

```
vMX-1_Week1-Snap_OSPF-test
vMX-1_Week1-Snap_BGP-test
```

    - By default snapshots are stored in /etc/jsnapy/snapshots
  - Taking a Snapshot
    - Creating a Snapshot syntax

```
user@jsnapy-server$ jsnapy --snap snapshot-name -
f configuration-filename
```

- Create a pre and post snapshot example

```
lab@jaut-desktop$ jsnapy --snap snap1
-f mySnapshotConfig.yml
lab@jaut-desktop$ jsnapy --snap snap2
-f mySnapshotConfig.yml
```

- Use the same configuration file for both snapshots

Comparing Two Snapshots

- Compare two snapshots using check

```
user@jsnapy-server$ jsnapy --check snapshot1
snapshot2 -f configuration-filename
```

- Example from previous slide

```
lab@jaut-desktop$ jsnapy --check snap1 snap2
-f mySnapshotConfig.yml
```

```
[lab@jaut-desktop jsnapy]$ jsnapy --check snap1 snap2 -f mySnapshotConfig.yml
**************************** Device: 172.25.11.1 ****************************
Tests Included: test_sw_version
************************** Command: show version **************************
PASS | Value of all "junos-version" at xpath "//software-information" is same
[ 1 matched ]
------------------------------ Final Result!! ------------------------------
test_sw_version : Passed
Total No of tests passed: 1
Total No of tests failed: 0
Overall Tests passed!!!
[lab@jaut-desktop jsnapy]$
```

Performing a Snapcheck

- Perform a snapcheck

```
user@jsnapy-server$ jsnapy --snapcheck snapshot2 -f
configuration-filename
```

- Example from previous slide

```
lab@jaut-desktop$ jsnapy --snapcheck  snap2
-f mySnapshotConfig.yml
```

Comparing Two Snapshots Using --diff

- Perform a snapcheck

```
user@jsnapy-server$ jsnapy --diff snapshot1 snapshot2
-f configuration-filename
```

- Example from previous slide

```
lab@jaut-desktop$ jsnapy -diff snap1 snap2
-f mySnapshotConfig.yml
```

Use JSNAPy as a Python Module

```
### Example showing how to pass yaml data in same file ###
from jnpr.jsnapy import SnapAdmin
from pprint import pprint
from jnpr.junos import Device

js = SnapAdmin()

config_data = """
hosts:
  - device: 198.51.100.10
    username : <username>
    passwd: <password>
tests:
  - test_exists.yml
  - test_contains.yml
  - test_is_equal.yml
"""
```

```
snapchk = js.snapcheck(config_data, "pre")
for val in snapchk:
    print "Tested on", val.device
    print "Final result: ", val.result
    print "Total passed: ", val.no_passed
    print "Total failed:", val.no_failed
    #pprint(dict(val.test_details))
```

- Additional Places For Information
  - Junos snapshot administrator in Python guide
    https://www.juniper.net/documentati...
  - The JSNAPy github
    GitHub - Juniper/jsnapy: Python vers...
  - Samples and examples that install with JSNAPy /etc/jsnapy/samples
- Questions and Answers
  - 1. What applications is the predecessor of JSNAPy?
    - JSNAP is the predecessor to JSNAPPY and was written in SLAX.
  - 2. The result of snapshots are returned in what format?
    - Snapshot results are returned in XML
  - 3. What Python module is needed to run JSNAPy within a Python script?
    - You need the SnapAdmin module to run JSNAPy within a Python script