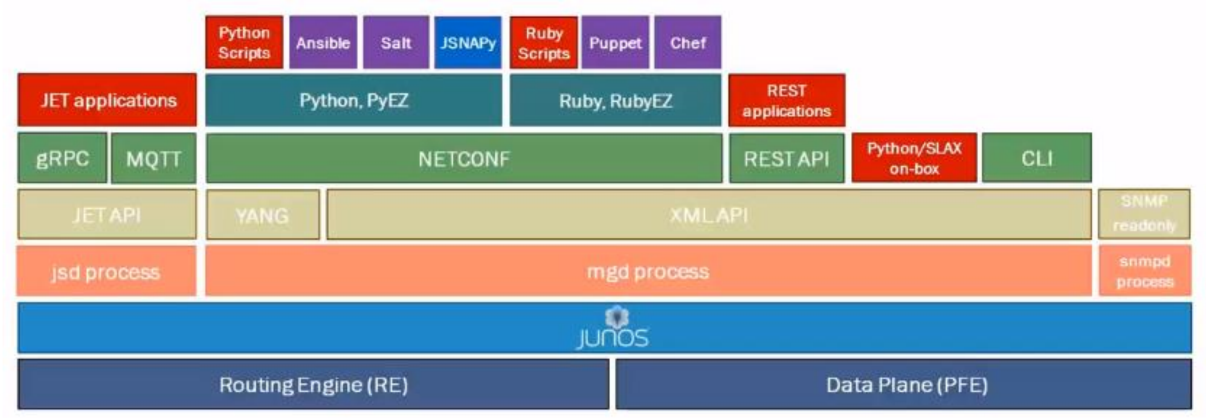
JNCIA-DevOps Study Notes

Junos Automation Stack

Automation Overview:



jsd process is responsible for the JET API. The JET API provides the ability for 3rd party vendors to create compatible applications that utilise the JET API. Makes writing API for Routing Protocols Daemon / Interfaces Daemon / Firewall Daemon easier and consistent.

mgd process is the management process. The XML API and YANG run on top of this process. NETCONF, REST API, CLI, and Python/SLAX on-box run on top XML API. NETCONF also runs on top of YANG.

- Ansible & Salt are automation tools that run on top of Python, PyEZ.
- Puppet & Chef are automation tolls that run on top of Ruby and RubyEZ. These both require Agents to be configured on the remote devices.

XML Request:

```
<rpc>
     <get-software-information>
     </get-software-information>
</rpc>
```

Response:

```
<rpc-reply>
          <software-information>
             ...
             <junos-version>18.1R1.9</junos-version>
             ...
          </software-information>
</rpc-reply>
```

REST API Request:

curl http://192.168.1.1:3000/rpc/get-software-information -u "admin:admin123" -H "Content-Type: application/xml" -H "Accept: application/xml"

XML & XPath

```
<name>Introduction to Junos</name>
```

opening tag closing tag

`<name></name>` can also be written as `<name/>`

Example XLM

```
<course>
     <name>Intro to Junos</name>
     <section>
        <name>Overview of the Course</name>
     </section>
     <section>
        <name>Junos OS</name>
       <subsection>
          <name>Junos CLI</name>
       </subsection>
     </section>
</course>
```

xmlns: XML Namespace. Can be defined at start, all sections then belong to the namespace.

More than one namespace is possible in a XML document.

```
<cource xlmns="http://xml.example.com/CBT"
        xlmns:graphics="http://xml.com/GRAPHICS">
   <name>Intro to Junos</name>
   <section>
      <name>Overview of the course</name>
      <graphics:image>image1.png</graphics:image>
   </section>
</course>
```

XML Schema Definition - An XSD file defines the elements allowed in an XML document and its hierarchy.

Ensures everyone is working with a standard set of tags to communicate information.

The XSD file itself uses XML format.

```
<xs:element name>"device-name" type="xs:string" />
```

```
<xs:element name>"interface-count" type="xs:integer" />
```

CLI commands - CLI communicates with Junos Infrastructure using XML API. Use *"| display xml rpc"* to see the translation.

Example:

```
user@route1# show route active-path
=
<get-route-information>
   <active-path/>
</get-route-information>
```

CLI translates the <rpc-reply> from XML into human readable format. To see the XML response use *"|display xml"* after command.

XML Nodes:

`<version>18.1R1.9</version>` - This is called an Element Node

`<version>18.1R1.9</version>` - this is a Text Node, inside the Element Node

`<configuration junos:commit-seconds="123456" junos:commit-localtime="2020-11-01 20:00:00 UTC">` - Attribute Node, inside the opening tag

A Document Node is an XML scheme for a single conf doc.

Element Nodes can be nested. Is a child Node of the Document Node.

XPath Axes

- Ancestor
- Parent
- Self
- Sibling
- Child
- Descendant

Operators

- Logical: **AND**, **OR**
- Comparison: **=**, **!=**, **<**, **>**
- Numerical: **+**, **-**, *****

XPath syntax allows selection of different types of nodes:

/configuration/system/root-authentication/encrypted-password

Each bit refers to an Element Node in an XML schema. Very similar to how to navigate to a file in Linux, absolute path from root or relative path from current Element Node. Can also use “../” to go up a level.

If there are more than one Element Node matching the XPath then it is possible to use [ ] in path:

system/login/user[name=='dai']/uid

Elements and Attributes

Hierarchical elements:

```
<wrapper>
   <element>
     <name>Dai</name>
   </element>
</wrapper>
```

XPath to retrieve name: element/name

Element with an attribute:

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

XPath to retrieve name: element/@name (use the @ to get the contents of an attribute)

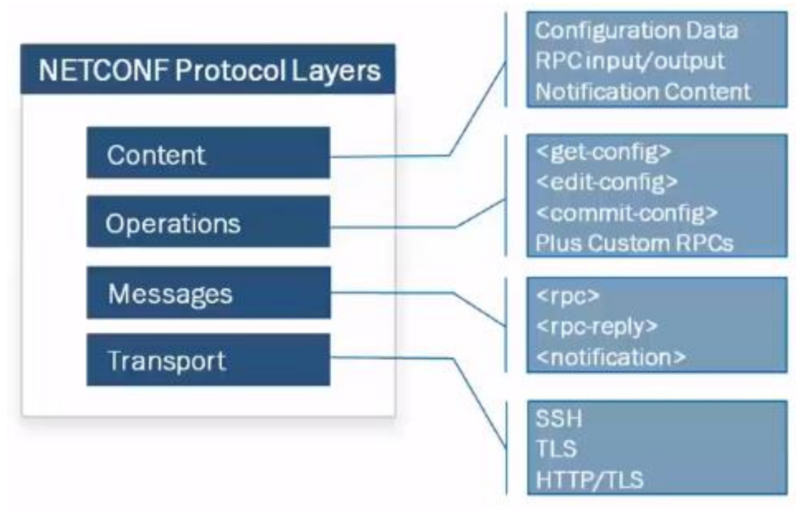
`junos:changed` is attached to nodes that have changed in the candidate configuration

`junos:group` is attached to nodes that have been inherited from a configuration group

XML API & NETCONF

NETCONF

NETCONF (RFC 6421) provides a standard connection protocol to communicate with network devices. It is supported by various vendors. Evolved from JunoScript which was proprietary.



Messages layer:

`<rpc>` - encapsulates all remote procedure 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>` - a one-way message and is sent to the client who initiated a `<create-subscription>` command when an event of interest has occurred.

Operations layer:

```
<lock>
<unlock>
<get>
<get-config>
<copy-config>
<commit>
<discard-changes>
<delete-config>
<validate>
<create-subscription>
<close-session>
<kill-session>
```

These are the standard operators defined by the RFC; however, vendors are able to, and often do, write their own operations.

Content layer:

Contains the RPC request and response payload.

Contains the configuration data.

NETCONF SSH must be configured to work over **TCP port 830.**

```
dai@vMX-1# set netconf ssh
```

To start a manual NETCONF session from CLI type `'netconf'`.

`]]>]]>` - This is a NETCONF termination signal, at the end of the reply.

From manual NETCONF session, then type RPC, example:

```
<rpc><get-system-uptime-information/></rpc>]]>]]>
```

Junos OS XML API

Realistically, manual NETCONF sessions will not be used. Junos provides Junos OS XML API. Uses:

- Issue operational mode commands
- Change the device configuration

Find the right RPC

- `| display xml rpc` – use this in CLI to identify the correct RPC to use
- Use the Junos XML schema (.XSD) file downloaded from Juniper. Each `element name` is a valid option.
- XML API Explorer on Juniper website
- Junos XML API Operation Developer Reference guide (pdf version of XML API Explorer)

Reply data from XML API is always in `<data>` tags.

XML API Programming Languages

On-box scripting options:

- SLAX
- Python
- XSLT

Off-box Scripting options:

- Python
- C++
- C
- Ruby
- C#
- Perl
- Java
- Go
- PHP
- Objective C

XSLT – standard language designed for XML to XML transformations. The XSLT script is written as an XML document. Can be very confusing.

SLAX – open source language. Directly translates XSLT. Works closely with XSLT but looks more like C.

Python – flexible high-level language. There is a NETCONF library and a PyEZ library. A few simple lines of code can get great results.

Perl – Retrieve, modify and commit configs. Download sample scripts from Juniper**.**

Java – Java library for NETCONF available on Git Hub.

Ruby - a NETCONF gem is available for this also.

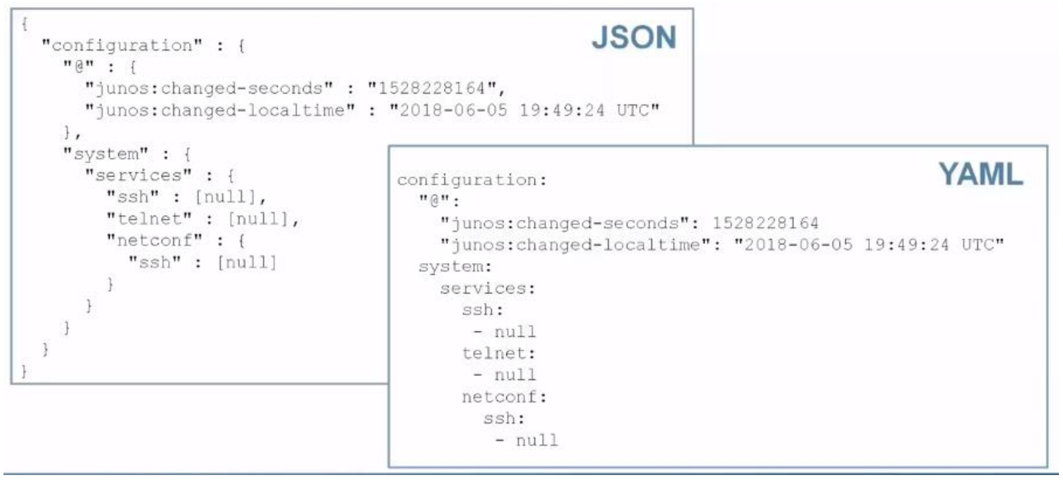
NETCONF Case Study

Use Ruby to update DNS config on a remote device. Need to review slides to grasp this, never seen Ruby script before.

JSON & YAML

Strings – "hello world"
Numbers – integer, floating point
Boolean values – true, false
Null value - null
Lists – sequence, array
Dictionaries – hash, object, mapping, associative array

JSON is easier to read than XML. JSON is also a subset of YAML.



JSON

```
{
  "configuration" : {
    "@" : {
      "junos:changed-seconds" : "1528228164",
      "junos:changed-localtime" : "2018-06-05 19:49:24 UTC"
    },
    "system" : {
      "services" : {
        "ssh" : [null],
        "telnet" : [null],
        "netconf" : {
          "ssh" : [null]
        }
      }
    }
  }
}
```

YAML

```
configuration:
  "@":
    "junos:changed-seconds": 1528228164
    "junos:changed-localtime": "2018-06-05 19:49:24 UTC"
  system:
    services:
      ssh:
        - null
      telnet:
        - null
      netconf:
        ssh:
          - null
```

JSON	**YAML**
• Case sensitive • Uses curly braces {} and square brackets [] for structure • Whitespace is ignored in JSON • Does not offer a way to comment the code • Structure: ○ Objects ○ Arrays	• Case sensitive • Uses indents for structure. Whitespace is important • Uses spaces, not tabs • Document starts with three dashes --- • Comments begin with a # • Strings do not need quotes unless they include special characters • YAML Structure: ○ Mappings

	○ Sequences, each sequence item starts with a dash

{
  "doe": "a deer, a female deer",
  "ray": "a drop of golden sun",
  "pi": 3.14159,
  "xmas": true,
  "french-hens": 3,
  "calling-birds": [
    "huey",
    "dewey",
    "louie",
    "fred"
  ],
  "xmas-fifth-day": {
  "calling-birds": "four",
  "french-hens": 3,
  "golden-rings": 5,
  "partridges": {
   "count": 1,
   "location": "a pear tree"
  },
  "turtle-doves": "two"
  }
}
	--- 	
	doe: "a deer, a female deer"
	ray: "a drop of golden sun"
	pi: 3.14159
	xmas: true
	french-hens: 3
	calling-birds:
	  - huey
	  - dewey
	  - louie
	  - fred
	xmas-fifth-day:
	  calling-birds: four
	  french-hens: 3
	  golden-rings: 5
	  partridges:
	    count: 1
	    location: "a pear tree"
	  turtle-doves: two

Ansible

- Accelerates the deployment of infrastructure#
- Automates configuration management and operations
- Initially created for compute and cloud but now supports networking as well
- Provides idempotent operations
  - Playbooks may be run multiple times to yield the same result
  - Modules only execute a change if required

Ansible and Junos:

- DevOps deploy new playbooks or modules via tools like Git →
- Ansible opens NETCONF over SSH sessions to remote devices and caries out tasks dictated in the playbooks →
- Python modules from Juniper are executed locally and interface with Junos' XML API via PyEZ

Install Ansible:

```
$pip install ansible

$pip install junos-eznc

$pip install jxmlease
```

If using Juniper.Junos Ansible Galaxy role:

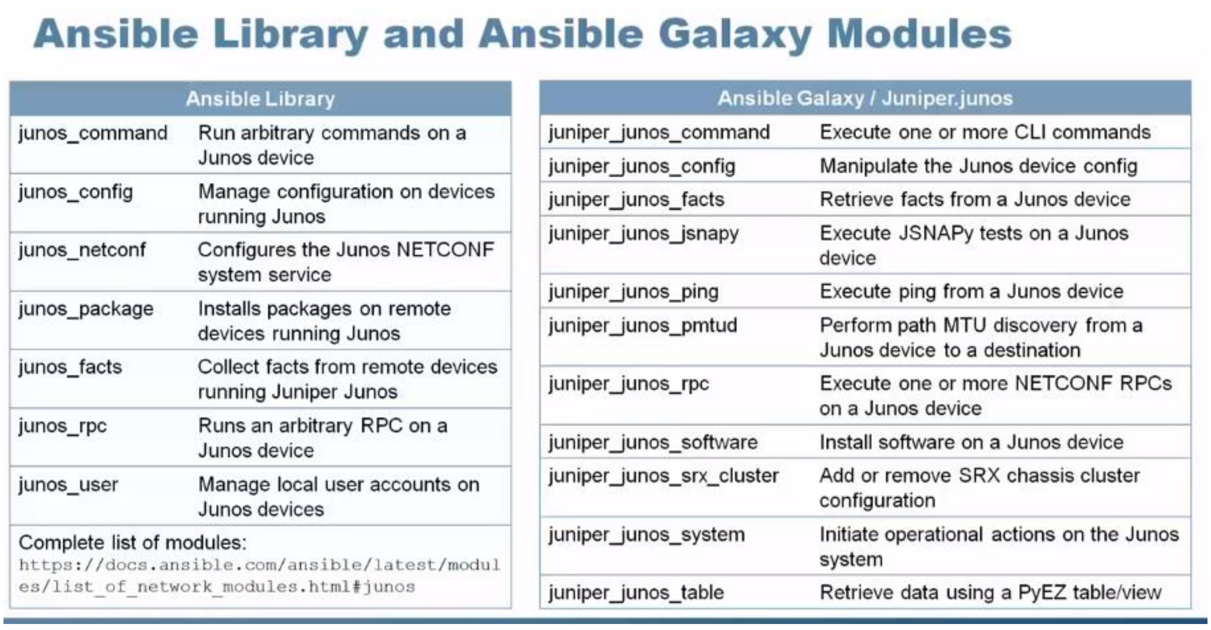
```
$ansible-galaxy install Juniper.junos
```

Configure NETCONF on the Junos OS Device

```
#set Netconf ssh
```

Ansible Modules

Ansible Module Library	**Ansible Galaxy – modules Juniper.junos role**
Officially supported by Ansible and the community, follow Ansible best practices and guidelines	Ansible Galaxy is an official Ansible community hub for sharing roles
Supported by Ansible Tower	Ansible Role is a reusable piece of Ansible code
Ships with Ansible	Supported by Juniper and community
	Separate installation using ansible-galaxy



Ansible Library and Ansible Galaxy Modules

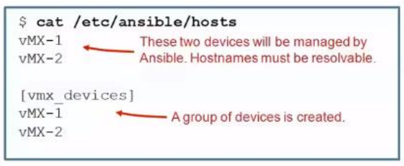
Ansible Library	
junos_command	Run arbitrary commands on a Junos device
junos_config	Manage configuration on devices running Junos
junos_netconf	Configures the Junos NETCONF system service
junos_package	Installs packages on remote devices running Junos
junos_facts	Collect facts from remote devices running Juniper Junos
junos_rpc	Runs an arbitrary RPC on a Junos device
junos_user	Manage local user accounts on Junos devices
Complete list of modules: `https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html#junos`	

Ansible Galaxy / Juniper.junos	
juniper_junos_command	Execute one or more CLI commands
juniper_junos_config	Manipulate the Junos device config
juniper_junos_facts	Retrieve facts from a Junos device
juniper_junos_jsnapy	Execute JSNAPy tests on a Junos device
juniper_junos_ping	Execute ping from a Junos device
juniper_junos_pmtud	Perform path MTU discovery from a Junos device to a destination
juniper_junos_rpc	Execute one or more NETCONF RPCs on a Junos device
juniper_junos_software	Install software on a Junos device
juniper_junos_srx_cluster	Add or remove SRX chassis cluster configuration
juniper_junos_system	Initiate operational actions on the Junos system
juniper_junos_table	Retrieve data using a PyEZ table/view

Default location of Ansible hosts' file:

`/etc/ansible/hosts`

May contain host and group variables.

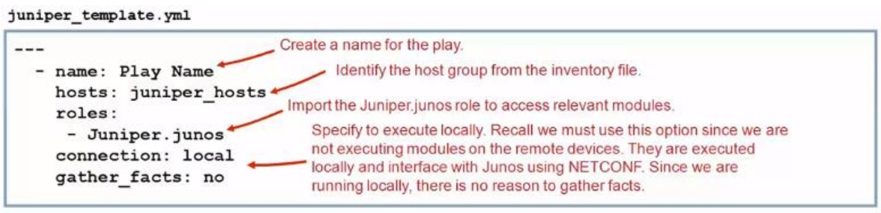


```
$ cat /etc/ansible/hosts
vMX-1
vMX-2

[vmx_devices]
vMX-1
vMX-2
```

These two devices will be managed by Ansible. Hostnames must be resolvable.

A group of devices is created.

Example Ansible playbook:



juniper_template.yml

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

Create a name for the play.

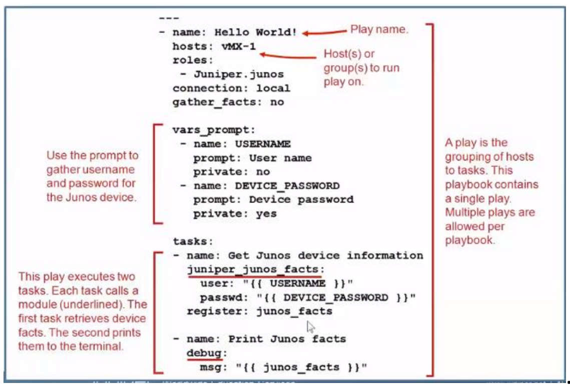
Identify the host group from the inventory file.

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 and interface with Junos using NETCONF. Since we are running locally, there is no reason to gather facts.

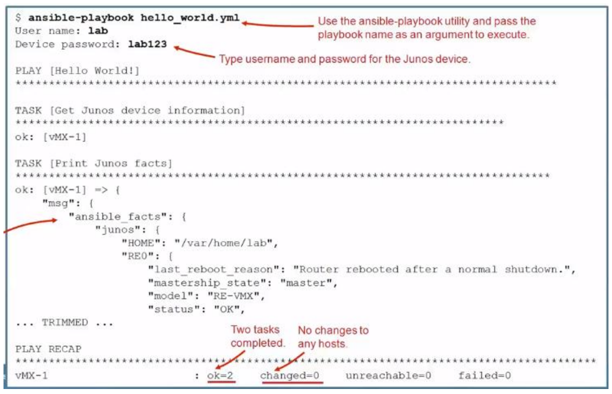
Hello World! Playbook:

`hello_world.yml`



```
---
- name: Hello World!
  hosts: vMX-1
  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
    juniper_junos_facts:
      user: "{{ USERNAME }}"
      passwd: "{{ DEVICE_PASSWORD }}"
    register: junos_facts

  - name: Print Junos facts
    debug:
      msg: "{{ junos_facts }}"
```

Play name.
Host(s) or group(s) to run play on.
Use the prompt to gather username and password for the Junos device.
This play executes two tasks. Each task calls a module (underlined). The first task retrieves device facts. The second prints them to the terminal.
A play is the grouping of hosts to tasks. This playbook contains a single play. Multiple plays are allowed per playbook.



```
$ ansible-playbook hello_world.yml
User name: lab
Device password: lab123

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

TASK [Get Junos device information]
**************************************************************************
ok: [vMX-1]

TASK [Print Junos facts]
********************************************************************************
ok: [vMX-1] => {
    "msg": {
        "ansible_facts": {
            "junos": {
                "HOME": "/var/home/lab",
                "RE0": {
                    "last_reboot_reason": "Router rebooted after a normal shutdown.",
                    "mastership_state": "master",
                    "model": "RE-VMX",
                    "status": "OK",
... TRIMMED ...

PLAY RECAP
*********************************************************************************************
vMX-1                          : ok=2    changed=0    unreachable=0    failed=0
```

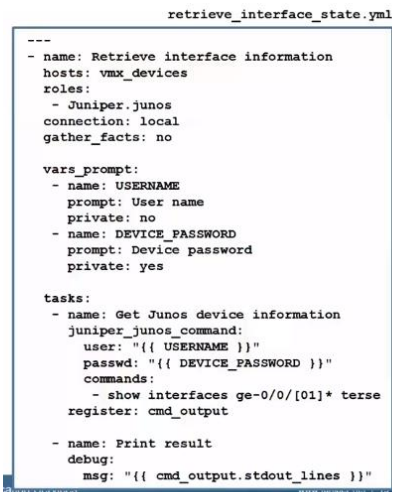
Use the ansible-playbook utility and pass the playbook name as an argument to execute.
Type username and password for the Junos device.
Two tasks completed.
No changes to any hosts.

{{ }} - variables are stored in double curly bracket.

Retrieving Junos Status Information:

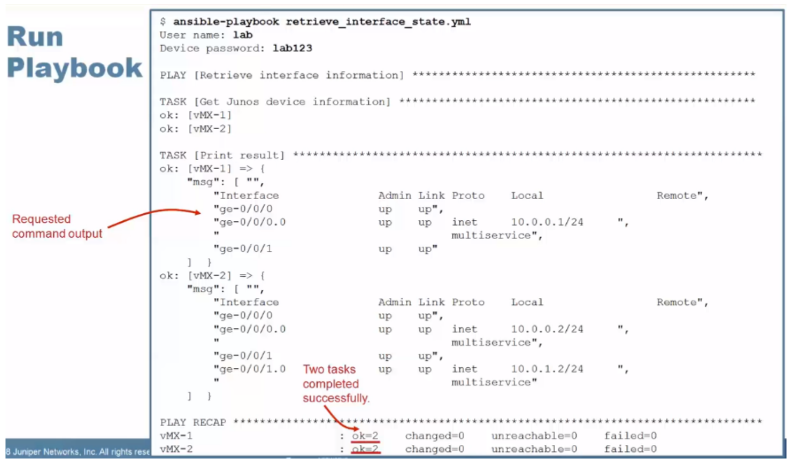
Juniper_junos_facts	Retrieves basic device facts
Juniper_junos_rpc	Executes one or more RPCs and returns results
Juniper_junos_command	Executes CLI commands on a Junos device
Juniper_junos_table	Retrieves data as PyEZ operational tables
Juniper_junos_jsnapy	Integrates Ansible with JSNAPy tool

- Output can be printed or stored to file system
- Asserts can be performed to ensure devices ae in proper state



retrieve_interface_state.yml

```
---
- name: Retrieve interface information
  hosts: vmx_devices
  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
     juniper_junos_command:
       user: "{{ USERNAME }}"
       passwd: "{{ DEVICE_PASSWORD }}"
       commands:
        - show interfaces ge-0/0/[01]* terse
     register: cmd_output

   - name: Print result
     debug:
       msg: "{{ cmd_output.stdout_lines }}"
```



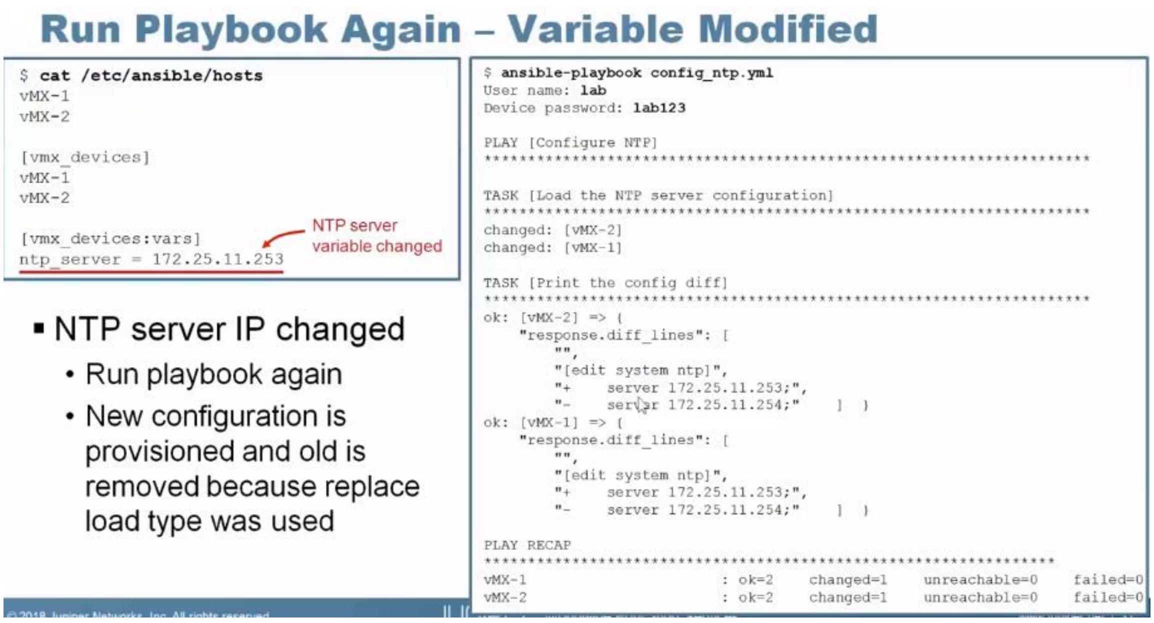
Run Playbook
Requested command output
Two tasks completed successfully.
8 Juniper Networks, Inc. All rights rese

```
$ ansible-playbook retrieve_interface_state.yml
User name: lab
Device password: lab123

PLAY [Retrieve interface information] ****************************************************

TASK [Get Junos device information] ******************************************************
ok: [vMX-1]
ok: [vMX-2]

TASK [Print result] **********************************************************************
ok: [vMX-1] => {
    "msg": [ "",
        "Interface               Admin Link Proto    Local                 Remote",
        "ge-0/0/0                up    up",
        "ge-0/0/0.0              up    up   inet     10.0.0.1/24      ",
        "                                   multiservice",
        "ge-0/0/1                up    up"
    ]  }
ok: [vMX-2] => {
    "msg": [ "",
        "Interface               Admin Link Proto    Local                 Remote",
        "ge-0/0/0                up    up",
        "ge-0/0/0.0              up    up   inet     10.0.0.2/24      ",
        "                                   multiservice",
        "ge-0/0/1                up    up",
        "ge-0/0/1.0              up    up   inet     10.0.1.2/24      ",
        "                                   multiservice"
    ]  }

PLAY RECAP *******************************************************************************
vMX-1                      : ok=2    changed=0    unreachable=0    failed=0
vMX-2                      : ok=2    changed=0    unreachable=0    failed=0
```

Juniper_junos_config – retrieve and modify Junos configuration:

- Retrieve config
- Load config
- Rollback config
- Commit config
- Exclusive and private modes supported
- XML, set, text, or JSON format
- Extensive docs available http://junos-ansible-modules.readthedocs.io

Ansible Case Study:



Run Playbook Again – Variable Modified

```
$ cat /etc/ansible/hosts
vMX-1
vMX-2

[vmx_devices]
vMX-1
vMX-2

[vmx_devices:vars]
ntp_server = 172.25.11.253
```

- NTP server IP changed
  - Run playbook again
  - New configuration is provisioned and old is removed because replace load type was used

```
$ ansible-playbook config_ntp.yml
User name: lab
Device password: lab123

PLAY [Configure NTP]
**********************************************************************

TASK [Load the NTP server configuration]
**********************************************************************
changed: [vMX-2]
changed: [vMX-1]

TASK [Print the config diff]
**********************************************************************
ok: [vMX-2] => {
    "response.diff_lines": [
        "",
        "[edit system ntp]",
        "+    server 172.25.11.253;",
        "-    server 172.25.11.254;"     ]  }
ok: [vMX-1] => {
    "response.diff_lines": [
        "",
        "[edit system ntp]",
        "+    server 172.25.11.253;",
        "-    server 172.25.11.254;"     ]  }

PLAY RECAP
******************************************************************
vMX-1                      : ok=2    changed=1    unreachable=0    failed=0
vMX-2                      : ok=2    changed=1    unreachable=0    failed=0
```

Additional Ansible Features:

- Run "ad hoc" commands using `ansible` CLI
- Run playbooks in "dry run" `--check` mode
- Secure communications with SSH keys or encrypted Ansible Vault files
- Process loops and conditional statements
- Notify handlewrs when a change is performed
- Retry playbook only for hosts that failed previous run
- Apply configuration templates
- Create roles to modularize and reuse the code
- Visit: https://docs.ansible.com

Python

Junos PyEZ

Junos REST API