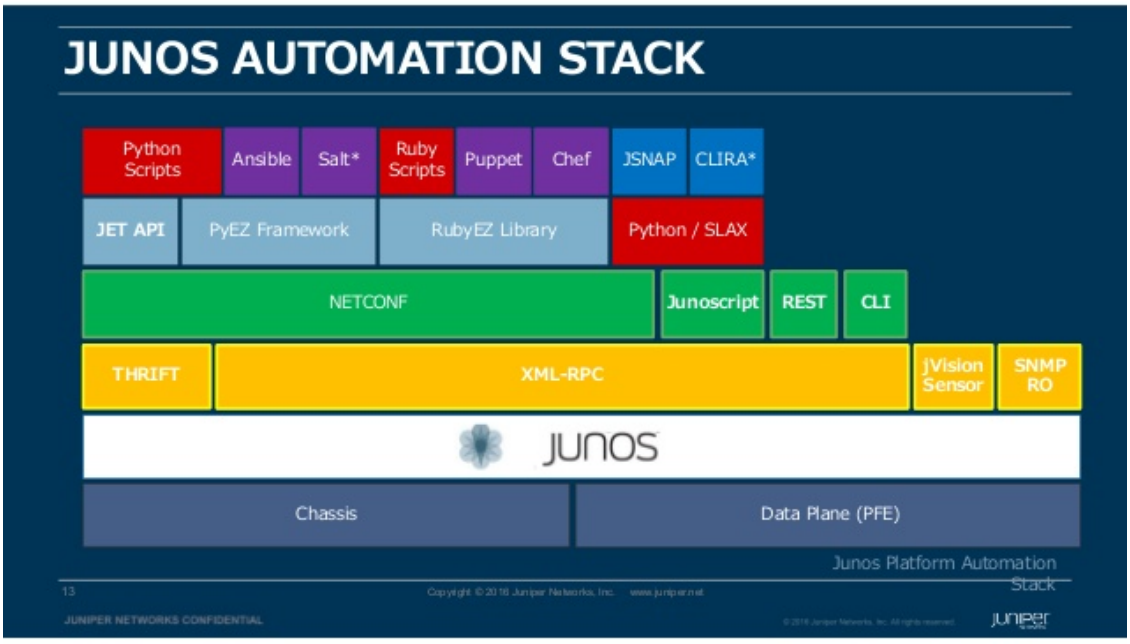
Juniper_Automation

July 4, 2020


```
configure
    load override [conf_file]   !load configuration from a file
    commit and-quit             !commit changes
```


1 Junos Automation Stack

This module discusses various APIs and tools that Junos provides to support devops



The most important process are * mgd : Management daemon, provides XML API which is further

used by REST/NETCONF/Python/SLAX etc for application extension
* jsd : Junos Daemon is responsibele for JET (junos extension) API that uses gRPC and MQTT for communication with JET application (onbox/offbox) * snmpd : responsible for SNMP access but its read-only

Junos and DevOps * API support * Transactioal configuration updates: Commit model * Data modelling support: Internal (ODL,DDL), YANG, OpenConfig (Vendor Agnostic) * Integration to automation tools such as Ansible, Pupper, Chef and Saltstack

1.1 Onbox Automation

Script types * Commit script: runs at commit, performs sanity check to enfore consistancy * Op Script: Works as custom op-mode commands
* Event Script: Initiated by events * SNMP Script: typical SNMP

Onbox scripts are powerful but they help in automation but not orchestration. The script interactis with the devices using XML. To perform orchestration use NETCONF, it uses same XML API but uses a secure connection using SSH from a central node and uses YANG as a data model. All Junos system supports NETCONF. Additioally varius programming laguages support NETCONG using libraries (Python, Java, Ruby, Perl, C, C++ etc.)

1.1.1 PyEZ

Although python supports NETCONF library but it needs various addiotional config, therefore PyEZ is a juniper library that provides easy access. It is built on top of NETCONF library. It parses XML-RPC and works on config and automation.

Junos support several automation tools with configuration templates such as Ansible, Puppet, Chef and Salt-stack.

Ansible	Salt	Puppet	Chef
Agentless	Agentless Event Driven	Agent Based (netDev)	Agent Based

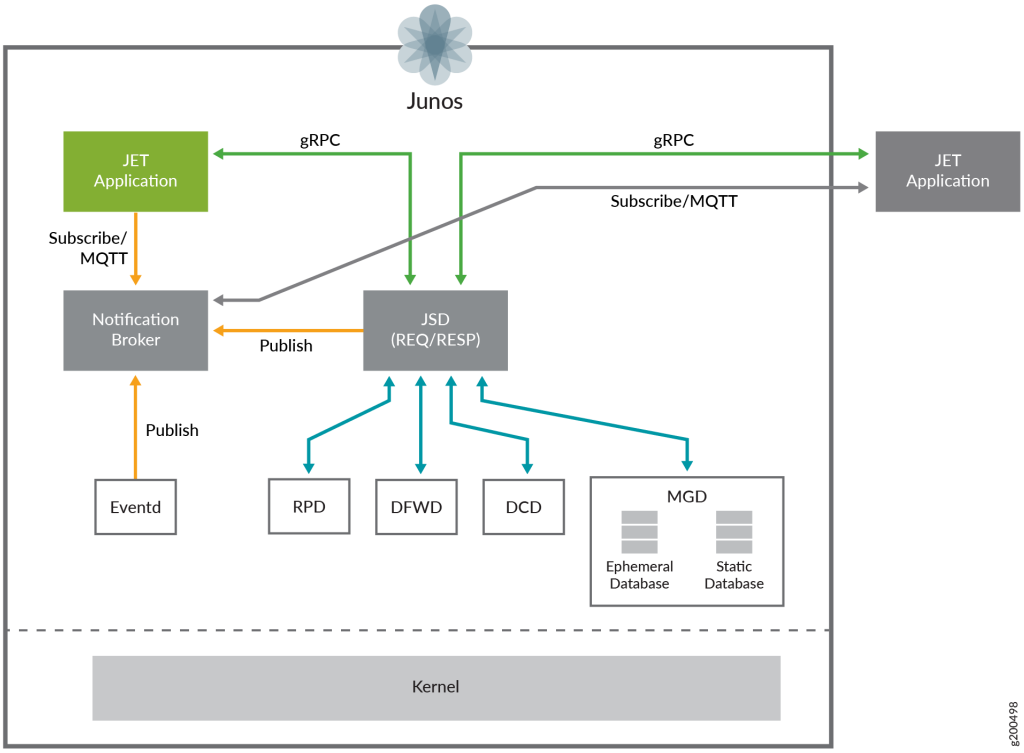
1.2 REST API

- Locate transfer and manipulate data by the API server
- Uses HTTP(s) for trasport (Request/Response Model)
- Stateless, each query is independent
- Most programming languages support REST
- Supported by MX,M,T,PTX and SRX series platform

1.2.1 Benifits of Junos REST

- Easy to setup and configure
- NETCONF and SSH are not nessesary
- GUI API explorer
- Can retrive data in text, JSON and XML formats

1.3 JET (Junos Extension Toolkit)



* enanles 3rd parties to create application for JUNOS * program the junos CP and create custom commands for junos * JSD, Junos Services Daemon interfaces with individual API through common RPC Daemons to access the subject daemons underneath. * JET uses gRPC and MQTT for external access * Uses python and C/C++ for onBox * Fast programmatic config datatabase with 1000s of config changes per sec * JSD and MQTT notification broker are separate services and runs on different TCP Port.

1.3.1 JSNApy

- takes snapshot and compares snapshot
- used to audit environment against provided data
- python version of JSNAP (was

1.3.2 Junos ZTP

- Allow for Zero-Touch-Provissioning form junos devices
- Performs auto-installation
  - Configure DHCP server
  - DHCP server reports the config server location
  - ZTP enable device will fetch the config and apply it

2 XML and XPATH

- Both on-box and off-box tools uses XML to interact with Junos infra.
- Request-Response model, Extensive markup language

- Tag based : `<tag> data </tag>`
- Shorthand for tags without any data : `<tag\>` = `<tag></tag>`
- Any hierarchy can be represented

```
<course>
    <name> Intro to python </name>
    <sections>
        <sec1> data </sec1>
        <sec2> data </sec2>
        <sec3> data </sec3>
    </sections>
</course>
```

- Namespace: `<tag xmlns=NAMESPACE>` its possible to have multiple namespace in a single XML document. namespace is provided as a URL using **xmlns** attribute.

2.1 XML Scehema Definition (XSD)

An XML doc represent a hierarchical structure, There are many way to express a single struct thus we need a standardization process called XML schema def or **XSD**. * defined the elements allowed in a XML doc and its hierarchy * all XML users will use same tags * XSD is written in an XML format

The junos CLI uses XML API to communicate with the junos infra, every commnad is translated to XML docs. use `CMD | display xml rpc` command to see the equivalant XML RPC request. and use `CMD | display xml` command to see the XML resposne

```
> show route active-path | disp xml rpc

<get-route-information>
    <active-path/>
</get-route-information>
```

Junos uses XSD specified for each resease, there is seperarte schemas for operational and configuration data, XSD is specified in namespace of the openning tag.

2.2 Some more terms

- Element nodes : `<tag attribute1=value1> Text_node </tag>`. Elemet nodes can be nested.
- Document node : root of the XML doc, root element is a child of document node
- **XPATH** Axes
  - *Ansestor* : from parent to root
  - *Desendent* : from child to leaf
  - *self* : me
  - *sibling* : having common parent
  - *Child* : One level down
  - *Parant* : One level up
- Operator
  - *Logical* : `AND,OR`
  - *Comparison* : `=, !=, <, >`

- *numarical* : `+, -, *`

2.3 XPATH

- Select specific XML node using text notation, similar to UNIX file path.

- e.g. `/config/system/root-authentication/encrypted-password/`
- XPATH can be **absolute** or ___relative___0

3 XML-API & NETCONF

3.1 Why NETCONF

- Orchestrating a number of network devices remotely using automation tool securely
- RFC 6241
- runs on TCP port 830 by default
- a successor of Junos-Script API
- **NETCONF Protocl Layers**

Layer	Task
Content	RPC req, resp paylaod and some additional variables
Operation	`<lock>,<unlock>`: ConfigDB, `<get>,<get-config>,<edit-config>,<copy-config>,<delet` `<commit>,<discard-changes>,<validate>`:Config change, `<create-subscription>,<close-session>,<kill-session>`: Session info
Message	`<rpc>`:Req, `<rpc-reply>`:ACK, `<notification>`:one way Server to client on subscription ack
Transport	SSH(Junos uses),TLS,HTTP(S)

- Configuring NETCONF, SSH alone activates NETCONF by default

```
configure
    set netconf ssh

show
ssh;
netconf{
    ssh;
}
```

- Activate NETCONF from CLI using `netconf` commnad, this will NOT autocomplete. `]]>]]>` is the message termination symbol
  - issue a netconf call `<rpc><get-system-uptime-information/></rpc>]]>]]>`, you'll get a XML response.

3.2 XML-API

- Usage : issues op mode command and changes the device config
- XML based encoding :

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

- To know **RPC for a CLI command** `CMD`
  - use `CMD | disp xml rpc` and see element within `<rpc>` tags, or
  - download the junos **XSD** file and look for the RPC from the CMD or
  - use XML API explorer from Juniper website
  - Finally, refer to the **XML API operational development guide** document
- XML-API Programming Language

script type	Language
On-box	SLAX, XSLT, **Python**
Off-box	**Python**, C++, C, C#, Perl, Java, Go, PHP, Obj C

3.2.1 Language Comparison

XSLT	SLAX	Python
* XSLT engine processes XSLT script* Designed by W3C and orginally used for Junos* XML-XML transformer * Converts nodes hierarchy into **if-then** tree also performs **loop**	* XSLT is difficult to program fot its XML like syntax* SLAX was develop as a Opensource language * C like syntax* works as a preprocessor to XSLT	* both onbox and offbox * NETCONF library or PyEZ* PyEZ is implemented over NETCONF* More general purpose

3.3 Use case

Router side	Server side
enable netconf `set netconf`	install `netconf` library
wait for reqest	write a script to fetch/fonfig data using netconf

4 JSON &YML

4.1 Data formatting basics

- Automation requires instruction to be sent in a formal manner to a machine
- Data needs to be formatted in a structed way before sending to an application
- Possible formats XML, YML, JSON
- Data Types:

- Simple types: Strings, boolean, Numbers, Null
- Complex types: List (array in json, sequence in YAML), Dictionary: `<key>:<value>` pair

4.2 JSON & YAML

- Junos can show configuration in both XML and JSON for YAML a manual conversion is needed.

JSON	YAML
* JavaScript Object Notation* Easy to read* Programming lang independent	* YAML Ain't a Markup Language * Easy to read, write and edit* Language independent * Uses indentation to show the structure * A superset of JSON

- PyEZ uses YML to describe a task also Ansible uses it for playbook.

4.3 JSON & YML Lab

1. Loading the base config

```
conf
    load ovverride FILE
    commit
    show
```

2. Display output in JSON

```
show int IFACE terse | display json
```

3. Modify Junos confug using JSON

```
# read_inventory.py
import yaml
import json

FILENAME = 'inventory.yaml'

with open(FILENAME) as f:
    content = yaml.load(f)

print(json.dumps(content))
```

return to terminal and run : `load merge json terminal [paste JSON]`

4. Create a YAML file

```
---
inventory:
    -hostname: vMX-1
     model: vMX
     address: 172.25.11.1
```

```
-hostname: vMX2
 model: vMX
 address: 172.25.11.2
```

run the `inventory.py` file

5 Ansible

5.1 Ansible Architecture

- A software to accelerate IT infrastructure deployment by automatig configuration and management operations.
- initially was created for compute and cloud, but now used for networking as well.
- Provides **idempotent** operation
  - operations are specified in **Playbooks** written in
  - No redundant change is permitted. Running a playbook multple times only new tasks that have not carried out will be take into effect.
- Integrates with devops tools and workflows
  - Palybook is first sent to VC system such as git
  - The playbook then pushed into CI/CD pipeline using Jenkins
  - finally gets deployed into network devices using Ansible server
- Ansible Operation priciple
  - Operators uploads playbooks into Git
  - Git interacts with the Ansible server that for deployment
  - Playbook uses modules which are copled from repo and distributed to the hosts
- Ansible in Junos
  - Operaters uploads playbook to git
  - Ansible server loads the playbook and required modules
  - The server runs the modules locally and configure the routers remotely using NETCONF

5.2 Building Ansible Environment

- Control serrver : OS (Linux, OS X, BSD), Python 2.6+, 3.5+
- Install Ansible : `pip install ansible`
- Additional Steps for Junos :

```
pip install junos-eznc jxmlease
ansible-galaxy install Juniper.junos
```

- Configure NETCONF on the devices

5.2.1 Different Ansible Modules

Ansible module	**Ansible Galaxy**
* officially supported by ansible community and follow ansible best practices * supported in Ansible tower * Ships with ansible	* Official ansible community hub * **Ansible Role** reusable piece of ansible code * supported by juniper community * requires seperate installation via ansible-agalay

5.2.2 Ansible Inventory file

- Lists the (group of) host(s) that is going to be managed (network devices)
- default location: `/etc/ansible/hosts`
- can have seperate inventory for testing, staging and production
- can contain variables
- seperate host and group variables files can be used for large environment

```
vMX-1   //DNS resolvable host names
vMX-2

[vmx_devices] // group name
vMX-1
vMX-2
```

5.3 Ansible PLaybook

- a playbook is a unit of work represnted in YAML
- we can create playbook to carry out multiple plays or series of tasks on multiple hosts
- Ansible palybook template for Junos

juniper_template.yml

```
---
    - name: play name        # name of the play
      hosts: vMX1            # hostnmname from inventory file
      roles:
         - Juniper.junos    # import relevent modules
      connection: local      # execute scrriot locally and connect with netconf
      gather_facts: no       # don't gather facts
```

- Lets see a complete playbook

#hello_world.yml

```
---
    - name: play name
      hosts: vMX1
      roles:
         - Juniper.junos
      connection: local
      gather_facts: no

      var_prompt:
          - name: USERNAME
            prompt: User name
            Private: no

          - name: DEVICE_PASSWORD
            prompt: password
```

```
      private: yes

  tasks:
      - name: get device info   #works as a function
        juniper_junos_facts:    #function inputs
            user: '{{ USERNAME }}'
            passwd: '{{ DEVICE_PASSWORD }}'
        register: junos_facts   #function output

      - name: print Junos facts
        debug:
            msg: '{{ junos_facts }}'
```

to run the playbook : `ansible-playbook hello_world.yml`

5.4 Ansible to retrive Junos info

module	task
`juniper_junos_facts`	retrives basic device facts
`juniper_junos_rpc`	executes one or more rpc and returns the result
`juniper_junos_command`	executes a CLI command on a junos device
`juniper_junos_table`	retrives data as PyEZ operational tables
`juniper_junos_jsnapy`	integrates ansible with JSNAPy too

Output can be strored into file system, assert can be performed to ensure devices are in proper state. An example is given below.

```
tasks:
    - name: get junos device info
      juniper_junos_commnad:
          user: '{{USERNAME}}'
          password: '{{PASSWORD}}'
          commands:
              - show int ge-0/0 terse
          register: cmd_output
```

5.5 Ansible to retrive and modefy Junos Config

- Junos ansible module `juniper_junos_config` allwos to, retrive, load, rollbak, commit config.
- Exclusive and public modules are supported
- Fortmat supported XML,Set, Text anf JSON.

```
tasks:
    - name: Retrive the committed config
      juniper_junos_config:
          user: {{USERNAME}}
          password: {{PASSWORD}}
          retrive: committed
```

```
        filter: interface
    register: response
```

5.6 Case Study

- Configure NTP server on vMX devices
  - provision NTP server config on all devices with a single playbook
  - make sure NTP server config can be changed without altering the playbook
  - NTP server config must be same for all devices

5.6.1 Create an inventory file

```
# /etc/ansible/host
vMX-1                       #list if host name
vMX-2

[vmx_devices]               #group name includes hosts
vMX-1
vMX-2

[vmx_devices:vars]          #setup a group variable
ntp_server = 172.25.11.254
```

5.6.2 Create a playbook

```
---
- name: config NTP
  hosts: vmx_devices
  roles:
      - juniper.junos
  connection: local
  gather_factcs: no

  var_prompt:
      - name: USERNAME
        prompt: user name
        private: no
      - name: PASSWORD
        promot: password
        private: yes

  tasks:
      - name: load the NTP server info
        juniper_junos_config:
            user: "{{USERNAME}}"
            password: "{{PASSWORD}}"
            config_mode: exclusive
            load:replace
            lines:
```

```
    - <configuration>
    -    <system>
    -          <ntp replace='replace'>
    -                <server>
    -                        <name> {{ntp_server}} </name>
    -                <server>
    -          </ntp>
    -   </system>
    - </configuration>
format: xml
commit: true
register: response
```

[ ]: