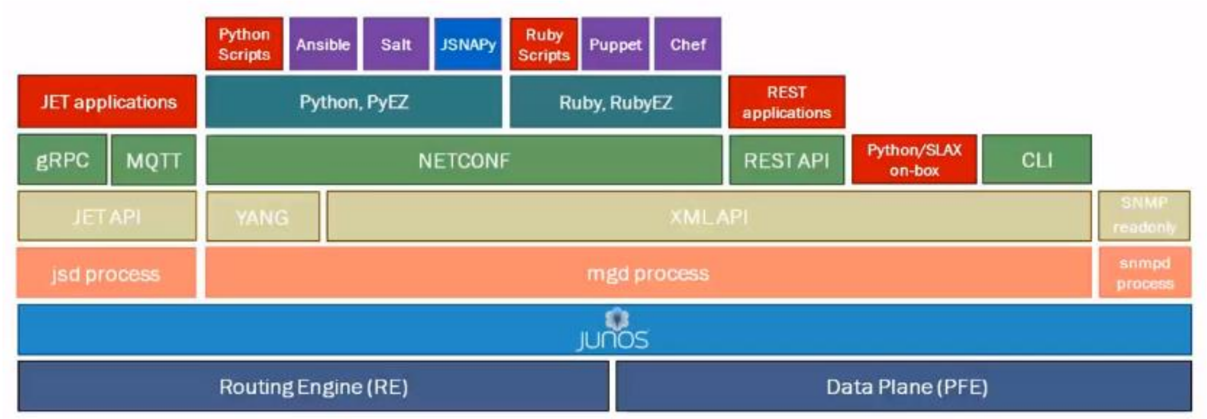
JNCIA-DevOps Study Notes

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>
```

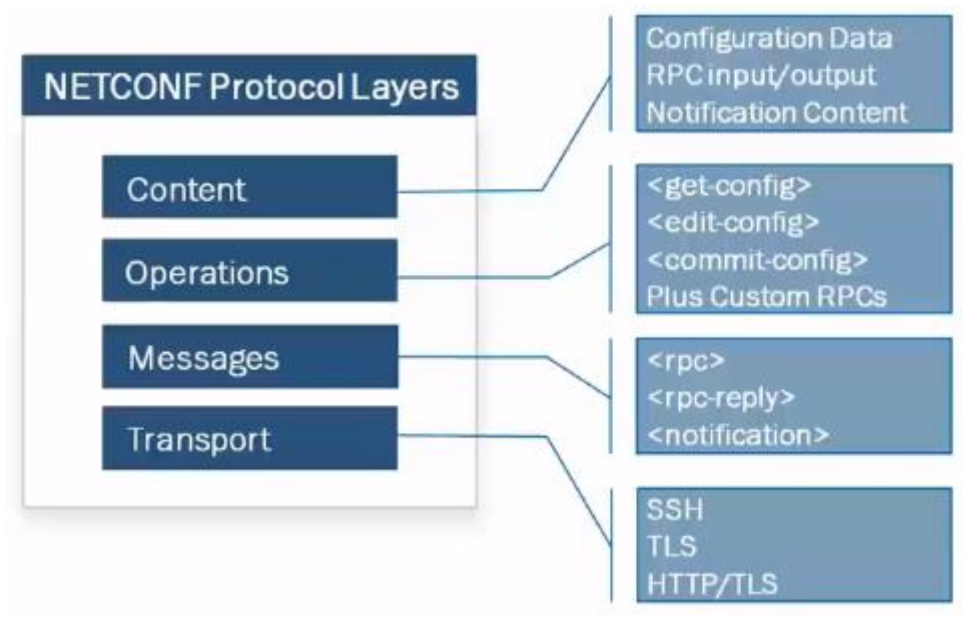
XPath to retrieve name: element/@name (use the @ to get the contents of an attribute) `junos:changed` is attached to nodes that have changed in the candidate configuration

`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>` - the `<rpc>` element 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 write their own operations.

Content layer:

Contains the RPC request and response payload.

Contains the configuration data.

NETCONF SSH has to 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. Have to go back through the slides to grasp this, never seen Ruby script before.

JSON & YAML

Ansible

Python

Junos PyEZ

Junos REST API