JUNOS NETCONF and YANG Tutorial

Sept/2014
Overview

• This presentation covers the basics of NETCONF and YANG
• This describes the current capabilities and future plans of Junos for supporting NETCONF and YANG
• Use cases that will get covered with support of YANG on Junos
• This does not assume prior knowledge of NETCONF and YANG
• Towards the end of this document, the reader shall be comfortable with using Junos YANG and playing with some suggested tools
Agenda

- Introduction to NETCONF and YANG
  - What is it?
  - Why is it different?
  - SNMP vs YANG
- NETCONF Overview and Examples
- Off-box Automation with NETCONF Toolkits
- YANG Overview and Examples
- Junos support for NETCONF and YANG
- Examples of Junos YANG Modules
- Getting started kit with Junos YANG Modules
Data-Model (YANG):
• Determines the Structure, Syntax and Semantics of the Data.
• Externally visible

Protocol (NETCONF):
• Remote primitives to view and manipulate the data.
• Encoding of data as defined by the Data Model
# NETCONF AND YANG – How is it different?

<table>
<thead>
<tr>
<th></th>
<th>SNMP</th>
<th>NETCONF</th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard</strong></td>
<td>IETF</td>
<td>IETF</td>
<td>W3C</td>
<td>W3C (draft)</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td>OIDs</td>
<td>Paths</td>
<td></td>
<td>URLs</td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
<td>Defined in MIBs</td>
<td>YANG Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Modeling Language</strong></td>
<td>SMI</td>
<td>YANG</td>
<td>(WSDL, not data)</td>
<td>Undefined, (WSDL), WADL, text....</td>
</tr>
<tr>
<td><strong>Management Operations</strong></td>
<td>SNMP</td>
<td>NETCONF</td>
<td>In the XML schema, not standardized</td>
<td>HTTP operations</td>
</tr>
<tr>
<td><strong>Encoding</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transport Stack</strong></td>
<td>UDP</td>
<td>SSH TCP</td>
<td>SSL HTTP TCP</td>
<td>SSL HTTP TCP</td>
</tr>
</tbody>
</table>
## SNMP vs NETCONF

<table>
<thead>
<tr>
<th>Use case</th>
<th>SNMP</th>
<th>NETCONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get operational/configuration data</td>
<td>YES</td>
<td>YES (Bulk xfer is faster)</td>
</tr>
<tr>
<td>Set state/configuration data</td>
<td>YES (up to 64KB)</td>
<td>YES</td>
</tr>
<tr>
<td>Set configuration fields in transaction</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>Transactions across multiple network elements</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>Invoke administrative actions</td>
<td>No clear definition</td>
<td>YES</td>
</tr>
<tr>
<td>Event Notifications</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Backup and restore configuration</td>
<td>Usually not</td>
<td>YES</td>
</tr>
<tr>
<td>Secure Protocol</td>
<td>No security for v1/v2 v3 is fair</td>
<td>YES</td>
</tr>
<tr>
<td>Test Configuration before final commit</td>
<td>No</td>
<td>YES</td>
</tr>
</tbody>
</table>
NETCONF Overview

IETF’s perspective:
- NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices
- It uses XML encoding for the configuration data as well as the protocol messages
- The NETCONF protocol operations are realized as RPCs over SSH as transport layer protocol
- RFC 6241 describes the Protocol

Customer’s perspective:
- NETCONF helps simplify and speed network device configuration
- That results into low OPEX and quick time-to-market for new Services

Juniper’s perspective:
- Junos is 100% NETCONF complaint from day-one
- Standard was defined by Juniper authors
NETCONF Model

- Configuration Data
  - RPC input/output
  - Notification Content

- <get-config>
- <edit-config>
- <commit-config>
  - Plus Custom RPCs

- <rpc>
- <rpc-reply>
- <notification>

- SSH
- SSL/TLS
- REST
Client/Server Model

NETCONF Client
- Client Application
- Data Models
- NETCONF Library
- Transport (SSH)

NETCONF Server (on Device)
- System Components
- NETCONF Server
- NETCONF Library
- Transport (SSHD)
### NETCONF Base operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close-session</td>
<td>:base</td>
<td>Terminate this session</td>
</tr>
<tr>
<td>commit</td>
<td>:base AND :candidate</td>
<td>Commit the contents of the <code>&lt;candidate/&gt;</code> configuration database to the <code>&lt;running/&gt;</code> configuration database</td>
</tr>
<tr>
<td>copy-config</td>
<td>:base</td>
<td>Copy a configuration database</td>
</tr>
<tr>
<td>create-subscription</td>
<td>:notification</td>
<td>Create a NETCONF notification subscription</td>
</tr>
<tr>
<td>delete-config</td>
<td>:base</td>
<td>Delete a configuration database</td>
</tr>
<tr>
<td>discard-changes</td>
<td>:base AND :candidate</td>
<td>Clear all changes from the <code>&lt;candidate/&gt;</code> configuration database and make it match the <code>&lt;running/&gt;</code> configuration database</td>
</tr>
<tr>
<td>edit-config</td>
<td>:base</td>
<td>Modify a configuration database</td>
</tr>
<tr>
<td>get</td>
<td>:base</td>
<td>Retrieve data from the running configuration database and/or device statistics</td>
</tr>
<tr>
<td>get-config</td>
<td>:base</td>
<td>Retrieve data from the running configuration database</td>
</tr>
<tr>
<td>kill-session</td>
<td>:base</td>
<td>Terminate another session</td>
</tr>
<tr>
<td>lock</td>
<td>:base</td>
<td>Lock a configuration database so only my session can write</td>
</tr>
<tr>
<td>unlock</td>
<td>:base</td>
<td>Unlock a configuration database so any session can write</td>
</tr>
<tr>
<td>validate</td>
<td>:base AND :validate</td>
<td>Validate the entire contents of a configuration database</td>
</tr>
</tbody>
</table>
Examples...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
</table>
| **edit-config** | Modify a configuration datastore by loading all or part of a specified configuration to the specified target configuration datastore. This operation allows the new configuration to be expressed in several ways, such as using a local file, a remote file, or inline.  

Example: The `<edit-config>` examples in this section utilize a simple data model, in which by the `<name>` element within each `<interface>` element. Set the MTU to 1500 on an interface named "Ethernet0/0" in the running configuration:  

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>
```

```xml
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```
### Examples...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
</table>
| **get**   | Retrieve running configuration and device state information.  
**Example:**  
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
<get>  
<filter type="subtree">  
<top xmlns="http://example.com/schema/1.2/stats">  
<interfaces>  
<interface>  
<ifName>eth0</ifName>  
</interface>  
</interfaces>  
</top>  
</filter>  
</rpc>  

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
<data>  
<top xmlns="http://example.com/schema/1.2/stats">  
<interfaces>  
<interface>  
<ifName>eth0</ifName>  
<ifInOctets>45621</ifInOctets>  
<ifOutOctets>774344</ifOutOctets>  
</interface>  
</interfaces>  
</top>  
</data>  
</rpc-reply> |
NETCONF TOOLKITS
Off-box Automation with NETCONF Toolkits

- Secure and connection oriented … SSHv2 as transport
- Structured and transaction based … XML as RPC request / response
- User-class privilege aware … Native to Junos

Management System

Automate config changes, remote invocation of operational commands, collection of logs

Secure TCP/IP connections via SSHv2 (RFC4742)

NETCONF client libraries exist for a number of programming languages such as Java, Perl, Ruby, Python, and even SLAX!
NETCONF Toolkits - Programming support

• Perl
  Available from Juniper Support Software download:
  http://www.juniper.net/support/downloads/?p=netconf#sw

• Ruby
  Available from RubyGems.org, maintained by Juniper Networks
  gem install netconf

• Python
  Available from open source project, not affiliated with Juniper Networks
  https://github.com/vbajpai/ncclient

• Java
  Available from Juniper Support Software download:
  http://www.juniper.net/support/downloads/?p=netconf#sw

• SLAX
  Available from Google code, maintained by Juniper Networks
  http://code.google.com/p/libslax/
  http://code.google.com/p/juise/
YANG Overview

IETF's perspective:
- YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications
- RFC 6020 describes the Protocol

Customer's perspective:
- Simple and easy to learn
- Though it stresses on readability yet provides sufficient flexibility, extensibility, and power to allow it to be useful for large schemas

Juniper's perspective:
- IETF-ed version of Juniper's native data modeling language DDL
Data Model Provides Rules for Data

Details all aspects of the data

What values are acceptable?
- Data types
- Ranges
- Lengths
- Regex matches

How is the data organized?
- Hierarchy
- Identifiers
- Uniqueness
- Ordering
- Referential integrity

How do I manipulate it?
- CRUD operations
- merge, replace,
- insert before/after

What does the data mean?
- Documentation
- References

What values are acceptable?
How is the data organized?
How do I manipulate it?
What does the data mean?
## YANG Terminology

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base type</td>
<td>The type from which a derived type was derived, which may be either a built-in type or another derived type.</td>
</tr>
<tr>
<td>built-in type</td>
<td>A YANG data type defined in the YANG language, such as <code>uint32</code> or <code>string</code>.</td>
</tr>
<tr>
<td>choice</td>
<td>A schema node where only one of a number of identified alternatives is valid.</td>
</tr>
<tr>
<td>container</td>
<td>An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes.</td>
</tr>
<tr>
<td>derived type</td>
<td>A type that is derived from a built-in type (such as <code>uint32</code>), or another derived type.</td>
</tr>
<tr>
<td>device deviation</td>
<td>A failure of the device to implement the module faithfully.</td>
</tr>
<tr>
<td>extension</td>
<td>An extension attaches non-YANG semantics to statements. The extension statement defines new statements to express these semantics.</td>
</tr>
<tr>
<td>grouping</td>
<td>A reusable set of schema nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it.</td>
</tr>
<tr>
<td>identifier</td>
<td>Used to identify different kinds of YANG items by name.</td>
</tr>
<tr>
<td>leaf</td>
<td>A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.</td>
</tr>
<tr>
<td>leaf-list</td>
<td>Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.</td>
</tr>
<tr>
<td>list</td>
<td>An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.</td>
</tr>
<tr>
<td>module</td>
<td>A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and &quot;compilable&quot;.</td>
</tr>
<tr>
<td>submodule</td>
<td>A partial module definition that contributes derived types, groupings, data nodes, RPCs, and notifications to a module. A YANG module can be constructed from a number of submodules.</td>
</tr>
</tbody>
</table>
## Modules and submodules

### Module `acme-module`

```yang
module acme-module {
    namespace "http://acme.example.com/module";
    prefix acme;

    import "yang-types" {
        prefix yang;
    }
    include "acme-system";

    organization "ACME Inc.";
    contact joe@acme.example.com;
    description "The module for entities implementing the ACME products";

    revision 2007-06-09 {
        description "Initial revision.";
    }
    ...
}
```

### Leaf `host-name`

```yang
leaf host-name {
    type string;
    mandatory true;
    config true;
    description "Hostname for this system";
}
```

**NETCONF XML Encoding:**

```xml
<host-name>my.example.com</host-name>
```

A leaf has
- one value
- no children
- one instance
### Examples...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
</table>
| **leaf-list** | ```
leaf-list domain-search {
  type string;
  ordered-by user;
  description "List of domain names";
}
```  
  *NETCONF XML Encoding:*
  ```
  <domain-search>high.example.com</domain-search>
  <domain-search>low.example.com</domain-search>
  <domain-search>everywhere.example.com</domain-search>
  ```
  *A leaf-list has*
  - one value
  - no children
  - multiple instances |
| **container** | ```
container system {
  container services {
    container ssh {
      presence "Enables SSH";
      description "SSH service specific configuration";
      // more leafs, containers and stuff...
    }
  }
}
```  
  *NETCONF XML Encoding:*
  ```
  <system>
    <services>
      <ssh/>
    </services>
  </system>
  ```
  *A container has*
  - no value
  - holds related children
  - one instance
  *May have specific meaning (presence)*
  Or may simply contain other nodes |
| **must** | ```
container timeout {
  leaf access-timeout {
    description "Maximum time without server response";
    units seconds;
    mandatory true;
    type uint32;
  }

  leaf retry-timer {
    description "Period to retry operation";
    units seconds;
    type uint32;
    must "$this < ../access-timeout" {
      error-app-tag retry-timer-invalid;
      error-message "The retry timer must be less than the access";
    }
  }
}
```  
  *A leaf-list has*
  - one value
  - no children
  - multiple instances
  *Constrains nodes by XPath expression*
  *Also:*
  - keyref
  - unique
  - range
  - pattern |
<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>augment</strong></td>
<td><code>augment interfaces/interface {</code>&lt;br&gt;<code>    when &quot;type = 'ethernet'&quot;;</code>&lt;br&gt;<code>    container ether-options {</code>&lt;br&gt;<code>        leaf auto-negotiation {</code>&lt;br&gt;<code>            type empty;</code>&lt;br&gt;<code>        }</code>&lt;br&gt;<code>    }</code>&lt;br&gt;<code>}&lt;br&gt;</code>&lt;br&gt;<code>NETCONF XML Encoding:</code>&lt;br&gt;<code>&lt;interfaces&gt;</code>&lt;br&gt;<code>    &lt;interface&gt;</code>&lt;br&gt;<code>        &lt;name&gt;fe-0/0/0&lt;/name&gt;</code>&lt;br&gt;<code>        &lt;type&gt;ethernet&lt;/type&gt;</code>&lt;br&gt;<code>        &lt;ether-options&gt;</code>&lt;br&gt;<code>            &lt;auto-negotiation/&gt;</code>&lt;br&gt;<code>        &lt;/ether-options&gt;</code>&lt;br&gt;<code>    &lt;/interface&gt;</code>&lt;br&gt;<code>&lt;/interfaces&gt;</code>&lt;br&gt;`<strong>Extends data model</strong>&lt;br&gt;– Current or imported modules&lt;br&gt;<strong>Inserts nodes</strong>&lt;br&gt;– Into an existing hierarchy&lt;br&gt;– Nodes appear in current module's namespace&lt;br&gt;– Original (augmented) module is unchanged&lt;br&gt;<strong>Allows sparse augmentation</strong>&lt;br&gt;– Nodes are only added when condition is true&lt;br&gt;– &quot;when&quot; is XPath expression</td>
</tr>
</tbody>
</table>
| **grouping** | `grouping target {`<br>`    leaf address {`<br>`        type inet:ip-address;<br>`        description "Target IP"`<br>`    }`<br>`    leaf port {`<br>`        type inet:ip-port;<br>`        description "Target port"`<br>`    }`<br>`} container peer {`<br>`    container destination {`<br>`        uses target;`<br>`    }`<br>`}<br>`<br>`NETCONF XML Encoding:`<br>`<peer>`<br>`    <destination>`<br>`        <address>192.0.2.1</address>`<br>`        <port>22</port>`<br>`    </destination>`<br>`}</peer>`<br>`**Defines a reusable collection of nodes**<br>Use multiple times<br>– A modules may use groupings imported from other modules<br>**Refinement**<br>Use as structure, record, or object
### Examples...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
</table>
| rpc activate-software-image | ```<rpc xmlns="urn:mumble">  
<activate-software-image>  
<image-name>image.tgz</image-name>  
</activate-software-image>  
</rpc>``` |

#### Defines RPC
- method names
- input parameters
- output parameters
Tools (yang-central.org)

- **pyang (python)**
  - Validates YANG
  - Translates between YANG and YIN (XML)
  - Generates XSD
- **yangto (binary)**
  - Validates YANG
  - Generates XSD, dependencies, etc
- **libsmi**
  - Translates SMI/SMIv2 MIBs to YANG
- **Other goodies**
  - Emacs mode

Try out the tools @ www.yang-central.org
JUNIPER NETCONF/YANG SUPPORT
Routing Engine

Routing (RPD) → Interfaces (DCD) → Chassisd → Class of Service → Mgmt Socket → SNMP → Events → eventd → snmpd → mgd → XML API Client

Config Database

cli
JUNOS UI Architecture

Generic core:
- Token lexing
- BISON grammar
- Config database
- Config groups

Shared infrastructure
Feature parity
Permissions

Data models drive:
- Parsing of config and command
- Rendering of config and commands
- Completion
- Component interaction
- Actions (what to do with data)

Command line editing (emacs)
Display of completions
Automore (and most pipes)
Rendering XML into text (ODL)
Interactive commands

Generic core:
- Token lexing
- BISON grammar
- Config database
- Config groups

Shared infrastructure
Feature parity
Permissions

Data models drive:
- Parsing of config and command
- Rendering of config and commands
- Completion
- Component interaction
- Actions (what to do with data)

Generic core:
- Token lexing
- BISON grammar
- Config database
- Config groups

Shared infrastructure
Feature parity
Permissions

Data models drive:
- Parsing of config and command
- Rendering of config and commands
- Completion
- Component interaction
- Actions (what to do with data)
What does Junos have?

NETCONF:
- Releases - All Junos releases support Netconf
- Platforms - All Junos Platforms support Netconf. As the Netconf RFC keeps getting amended, we keep adding the support on Junos releases in development
- Netconf implementation to adhere to it

YANG:
- Releases
  - 14.2 (partial support - configuration schema in yang)
  - 15.2+ (full support - configuration and operational schema in yang)
- Platforms - All
Junos plan to support YANG

- **YANG for Managing JUNOS**
  - Enables customers to leverage on YANG based tools by publishing Junos configuration and operational schema in YANG, in 15.2 and above.

- **YANG for JUNOS Developers**
  - Enables Junos developers to write YANG models for future developments by providing YANG compiler and support utilities in 16.1 and above.

- **Extending JUNOS UI**
  - Support translation of Juniper's native schema to IETF's standard YANG schema and vice-versa using translation scripts in 16.2 and above.
Getting Junos YANG Modules

- **CLI Commands**

```plaintext
show system schema format module juniper-command output-file-name <filename>
```

If the output file name is not present, it will be displayed on console.

- **JUNOScript rpc**

```xml
<rpc>
  <get-schema>
    <format>yang</format>
    <identifier>juniper-command</identifier>
    <output-file-name>file-name</output-file-name>
  </get-schema>
</rpc>
```

Here `<output-file-name>file-name</output-file-name>`, file-name is local path in the device.
Example (Part of Interface schema)

<table>
<thead>
<tr>
<th>PART 1</th>
<th>PART 2</th>
<th>PART 3</th>
<th>PART 4</th>
</tr>
</thead>
</table>
| container interfaces {  
  list interface (  
    must "(duplex and speed) or (not(duplex) and  
    "Neither or both of 'speed'  
    and 'duplex' must be set";  
    key name;  
    leaf name {  
      type interfaceName;  
    }  
    leaf description {  
      type string;  
    }  
    leaf enabled {  
      type boolean;  
      default true;  
    }  
    leaf mtu {  
      type int16 {  
        range "68..1500";  
      }  
    }  
    leaf mac {  
      type ieee:mac-address;  
    }  
  }  
  list unit {  
    key name;  
    leaf name {  
      type interfaceLogicalUnit;  
    }  
    leaf description {  
      type string;  
    }  
    leaf vlan-id {  
      type ieee:vlanid;  
    }  
  }  
}  
| container status {  
  config false;  
  leaf link {  
    type enumeration {  
      enum up;  
      enum down;  
    }  
  }  
  leaf speed {  
    type interfaceSpeed;  
  }  
  leaf mac {  
    type ieee:mac-address;  
  }  
  container receive {  
    uses commonPacketsCounters;  
  }  
  container transmit {  
    uses txPacketsCounters;  
  }  
}  
| container status {  
  container receive {  
    uses commonPacketsCounters;  
  }  
  container transmit {  
    uses txPacketsCounters;  
  }  
}  
| case c2 {  
  container inet6 {  
    list address {  
      key name;  
      leaf name {  
        type inet:ipv6-address;  
      }  
    }  
    leaf prefix-length {  
      type prefixLengthIPv6;  
    }  
  }  
|}
Thank you