NETCONF, YANG, RESTCONF
TECH-SDN-SP: Software Defined Networking for Service Providers

Martin Kramolis,
Systems Engineer,
CCIE #4738
Agenda

- Brief Overview of XML
- Introduction to NETCONF
- Introduction to YANG
- Introduction to RESTCONF
Brief Overview of XML
What is XML?

- eXtensible Markup Language
- A language to describe data
- Useful for serialization and data classification
- Not a complete programming language or database

Compare to [traditional] HTML
- XML: describe data, case-sensitive (similar to: JSON, YAML)
- HTML: display data, case-insensitive (similar to: TeX, troff)
Sample XML Data

```xml
<person>
  <name>
    <first>Thomas</first>
    <middle>Alva</middle>
    <last>Edison</last>
  </name>
  <occupation>
    Inventor and businessman
  </occupation>
</person>
```
XML Prolog

`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

- **version** – Currently, only 1.0 is valid (mandatory)
- **encoding** – Character set of the data to follow (optional, UTF-8 is default)
- **standalone** – yes if no external DTD is required, no otherwise (optional, no is default)
XML Elements

- XML tags are called **elements**
- Data between start and end tags are the element’s **content**
- Element content, including white space are **character data** where as tags are **markup**
- All elements must have start and end tags

```
<occupation>Inventor and businessman</occupation>
```

- **Attributes** can further describe elements

```
<name first="Thomas" last="Edison"> </name>
```

- **Empty elements** can simply end with a “/”

```
<name first="Thomas" last="Edison" />
```
XML Comments

- Further explain to the reader what the XML code is trying to describe
- Single and multi-line comments supported
- Comments can be inline with parsed XML
- All comments start with `<!--` and end with `-->`

```xml
<!-- This is a single line comment -->

<!--
    This is a multi-line comment.
    A multi-line comment spans multiple lines.
-->

<example name="Comment Example">
    <content>
        This text will be parsed as #PCDATA <!-- This text will not. -->
    </content>
</example>
```
XML Namespaces

- Disambiguates elements and attributes from different vocabularies with the same name
- Groups together related elements and attributes for easy processing
- Namespace objects start with a prefix followed by a colon (:) followed by the element or attribute name

```xml
<lab:annotation>
  <lab:documentation>Lab File Version</lab:documentation>
  <lab:docinfo>
    <LabFileMajorVersion>1</LabFileMajorVersion>
    <LabFileMinorVersion>3</LabFileMinorVersion>
  </lab:docinfo>
</lab:annotation>
```
Introduction to NETCONF
Why NETCONF?

- Typical Network configuration/monitoring still seen in majority of networks
  - Manual typing/scripting proprietary CLIs + backup repository to track changes, labor intensive, expensive, error prone
  - SNMP extensively used for fault handling and monitoring, but failed for configuration tasks

- Some operator’s requirements that paved the way for NETCONF and YANG (detailed in RFC 3535 – “Overview of the 2002 IAB Network Management Workshop”)
  - Must be easy to use
  - Clear distinction between configuration and operational data
  - Must scale to network-wide configurations rather than being focused on single devices
  - Must provide a way to backup and restore configurations
  - Must provide error-checking to ensure consistent configurations
  - Desirable to be able to process and store results using text-management tools like diff and VCS
  - Distinguish between modifying configuration and activating those modifications
  - Desirable to have multiple configuration stores on devices

- Need for move from “The Network is the Record” approach to “Network-wide” configuration database
NETCONF – high level concept

## NETCONF Protocol (RFC 6241)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>Configuration Data</td>
</tr>
<tr>
<td>Operations (methods)</td>
<td><code>&lt;get-config&gt;</code>, <code>&lt;edit-config&gt;</code></td>
</tr>
<tr>
<td>Messages</td>
<td><code>&lt;rpc&gt;</code>, <code>&lt;rpc-reply&gt;</code></td>
</tr>
<tr>
<td>Secure Transport</td>
<td>SSH, TLS, ..</td>
</tr>
<tr>
<td>Notification Data</td>
<td><code>&lt;notification&gt;</code></td>
</tr>
</tbody>
</table>

### Diagram:

- **Client**
- **Server**
- **NETCONF Configuration Datastore**

- **YANG defined**

---

Cisco and/or its affiliates. All rights reserved.

Cisco Public
NETCONF Data Stores and Transaction models

- Data stores are named contains that may hold an entire copy of the configuration
- Not all data stores are supported by all devices
- **Running** is the only mandatory data store
- Not all data stores are writable
- Check the device’s capabilities
- To make changes to a non-writeable data store, copy from a writable one
- **URL** is supported by IOS (for config-copy)

**Direct model**

```
<edit-config> Running
```

**Candidate model (optional)**

```
<edit-config> Candidate
<commit> Running
```

**Distinct Startup model (optional)**

```
<edit-config> Running
<commit> <copy-config> Startup
```
NETCONF Capabilities

- Capabilities are exchanged in hello messages
- RFC 6241 defines some base capabilities
  - :writable-running – the running data store can be modified directly
  - :candidate – the candidate data store is supported
  - :confirmed-commit – the NETCONF server will support the <cancel-commit> and the <confirmed>, <confirm-timeout>, <persist>, and <persist-id> parameters for the <commit> operation
  - :rollback-on-error – server will rollback the configuration to the previous state if an error is encountered
  - :validate – the server will validate the requested data store or config
  - :startup – the startup data store is supported
  - :url – the URL data store is supported
  - :xpath – filtering can be done using XPATH notation
  - :notification – NETCONF asynchronous event messages (RFC 5277)
NETCONF Capabilities

S: <?xml version="1.0" encoding="UTF-8"?>
S: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <capabilities>
S:     <capability>
S:       urn:ietf:params:netconf:base:1.1
S:     </capability>
S:     <capability>
S:     </capability>
S:   </capabilities>
S:   <session-id>4</session-id>
S: </hello>
S: ]]>]]>

C: <?xml version="1.0" encoding="UTF-8"?>
C: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:   <capabilities>
C:     <capability>
C:       urn:ietf:params:netconf:base:1.1
C:     </capability>
C:   </capabilities>
C: </hello>
C: ]]>]]>
# NETCONF Protocol Operations

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>REQ. CAPABILITY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;get-config&gt;</td>
<td>:base</td>
<td>Retrieve data from the running configuration database</td>
</tr>
<tr>
<td>&lt;get&gt;</td>
<td>:base</td>
<td>Retrieve data from the running configuration database and/or device statistics</td>
</tr>
<tr>
<td>&lt;edit-config&gt;</td>
<td>:base</td>
<td>Modify a configuration database</td>
</tr>
<tr>
<td>&lt;copy-config&gt;</td>
<td>:base</td>
<td>Copy a configuration database</td>
</tr>
<tr>
<td>&lt;delete-config&gt;</td>
<td>:base</td>
<td>Delete a configuration database</td>
</tr>
<tr>
<td>&lt;discard-changes&gt;</td>
<td>:base and :candidate</td>
<td>Clear all changes from the &lt;candidate/&gt; configuration database and make it match the &lt;running/&gt; configuration database</td>
</tr>
<tr>
<td>&lt;create-subscription&gt;</td>
<td>:notification</td>
<td>Create a NETCONF notification subscription</td>
</tr>
<tr>
<td>&lt;lock&gt;</td>
<td>:base</td>
<td>Lock a configuration database so only my session can write</td>
</tr>
<tr>
<td>&lt;unlock&gt;</td>
<td>:base</td>
<td>Unlock a configuration database so any session can write</td>
</tr>
<tr>
<td>&lt;commit&gt;</td>
<td>:base and :candidate</td>
<td>Commit the contents of the &lt;candidate/&gt; configuration database to the &lt;running/&gt; configuration database</td>
</tr>
<tr>
<td>&lt;cancel-commit&gt;</td>
<td></td>
<td>Cancels an ongoing confirmed commit.</td>
</tr>
<tr>
<td>&lt;close-session&gt;</td>
<td>:base</td>
<td>Terminate this session</td>
</tr>
<tr>
<td>&lt;kill-session&gt;</td>
<td>:base</td>
<td>Terminate another session</td>
</tr>
</tbody>
</table>
NETCONF Protocol Operations

- Client initiates session (typically over SSH) to Server
- Both sides exchange capabilities using <hello> message
- Operations are wrapped in XML-encoded RPC
- Client performs tasks using set of RPC transactions
- Example: Edit-config for device with <running> and <startup> datastore
  - Lock<running>, lock<startup>, edit-config<running>, copy<running>to<startup>,
    unlock<startup>,unlock<running>
- Example: Edit-config for device with <candidate> datastore
  - Lock<running>, lock<candidate>, edit-config<candidate>, commit<candidate>,
    unlock<candidate>,unlock<running>
NETCONF - Flow Breakdown – Request (IOS-XR)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <Configuration>
        </Configuration>
    </filter>
  </get-config>
</rpc>
```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="11" xmlns="urn:ietf:params:netconf:base:1.0">
  <data>
    <xml-config-data>
      <Device-Configuration xmlns="urn:cisco:xml-pi">
        <version>
          <Param>15.2</Param>
        </version>
        <service>
          <timestamps>
            <debug>
              <datetime>
                <msec/>
              </datetime>
            </debug>
          </timestamps>
        </service>
      </Device-Configuration>
    </xml-config-data>
    ...
  </data>
</rpc-reply>
Introduction to YANG
Why YANG?

- In order for NETCONF to be useful as a network-wide protocol, it must have a common data model.
- Simply wrapping CLI in XML is not enough as each vendor has its own CLI.
- YANG provides the common data model necessary for to consume NETCONF data from any network device.
- Each vendor must implement common YANG modules.
- Work on defining these modules is happening in the NETMOD group in the IETF.
What is YANG?

- YANG is a modeling language defined in RFC 6020
- Used by NETCONF to define the objects and data in requests and replies
- Analogous to XML schema and SMI for SNMP (but more powerful)
- Models configuration, operational, and RPC data
- Provides semantics to better define NETCONF data
  - Constraints (i.e., “MUSTs”)
  - Reusable structures
  - Built-in and derived types
- YANG is extensible and modular
- YANG modules are for NETCONF what MIBs are for SNMP
NETCONF concept versus SNMP

<table>
<thead>
<tr>
<th>“Framework”</th>
<th>“Content”</th>
<th>“Payload”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition language:</td>
<td>Information model:</td>
<td>Instantiated info/ transfer syntax:</td>
</tr>
<tr>
<td>YANG</td>
<td>YANG modules</td>
<td>XML</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Management services:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Netconf</td>
</tr>
<tr>
<td>Ability to express hierarchy</td>
<td>Import conversion rules exist</td>
<td>Bulk vs only incremental ops</td>
</tr>
<tr>
<td>(compare MIBs: flat + tables)</td>
<td>(MIBs (\rightarrow) YANG)</td>
<td>(manipulation of config files, e.g. edit-config)</td>
</tr>
<tr>
<td>Richer conditions, constraints</td>
<td>“instant content”</td>
<td>Transaction support</td>
</tr>
<tr>
<td>Facilities for easier reuse</td>
<td>Human readability</td>
<td>Configuration vs monitoring</td>
</tr>
<tr>
<td>RPC/Action support</td>
<td>Dynamic extensibility</td>
<td>or possibly other</td>
</tr>
<tr>
<td></td>
<td>B2B, Web toolkits</td>
<td>(no inherent dependency but will require different bindings)</td>
</tr>
</tbody>
</table>

Definition language: SMIv2

Information model: MIB modules

Instantiated info/ transfer syntax: ASN.1 BER

Management services: SNMP

Content

Payload
Example of YANG Module

module SystemTime {
  namespace "urn:cisco:params:xml:ns:yang:SystemTime";
  prefix "Cisco-SystemTime";
  organization "CISCO";
  contact "MKRAMOLI@CISCO.COM";
  revision "2014-06-16" {
    description "Example of YANG Schema";
  }
}

typedef time_source {
  type enumeration {
    enum TIME_SOURCE_ERROR { value 0; description "Error"; }
    enum TIME_SOURCE_NONE { value 1; description "Unsynchronized"; }
    enum TIME_SOURCE_NTP { value 2; description "NTP protocol"; }
    enum TIME_SOURCE_MANUAL { value 3; description "User configured"; }
    enum TIME_SOURCECALENDAR { value 4; description "HW calendar"; }
  }
  description "Time source";
}

grouping time_date {
  leaf Year { type uint16; description "Year [0..65535]"; }
  leaf Month { type uint8; description "Month [1..12]"; }
  leaf Day { type uint8; description "Day [1..31]"; }
  leaf Hour { type uint8; description "Hour [0..23]"; }
  leaf Minute { type uint8; description "Minute [0..59]"; }
  leaf Second { type uint8; description "Second [0..60]"; }
  leaf Millisecond { type uint16; description "Millisecond [0..999]"; }
  description "Date and time";
}

grouping system_uptime {
  leaf Hostname { type string; description "Host name"; }
  leaf Uptime { type uint32; description "Seconds Up"; }
  description "System uptime";
}

container SystemTime {
  description "System time";
  container Clock {
    config false;
    uses "time_date";
    description "System clock";
  }
  container Uptime {
    config false;
    uses "system_uptime";
    description "Sys. uptime";
  }
  description "System time";
}

container Clock {
  config false;
  uses "time_date";
  description "System clock";
}

container Uptime {
  config false;
  uses "system_uptime";
  description "Sys. uptime";
}
YANG models and structure

- YANG modules
  - Can be Automatically Validated
  - Can be Visualized to UML diagrams, compact Trees, etc.
  - Can be Translated to schemas like DSDL, XSD, etc.
  - Can be Converted to YIN
  - Can be Derived from YIN
  - Can drive Code Generation

---

**Module:** SystemTime

```yang
module: SystemTime
  +++-rw SystemTime
  |     +++-ro Clock
  |         | +++-ro Year?       uint16
  |         | +++-ro Month?      uint8
  |         | +++-ro Day?        uint8
  |         | +++-ro Hour?       uint8
  |         | +++-ro Minute?     uint8
  |         | +++-ro Second?     uint8
  |         | +++-ro Millisecond? uint16
  |         | +++-ro TimeSource? time_source
  |         |     +++-ro Uptime? uint32
  |         |         +++-ro Hostname? string
  |         |         +++-ro Uptime? uint32
```
YANG model execution in NETCONF

- Query/Response for System Time aligned with YANG module definition
- Note: screenshots taken from IOS XRv 5.1.1
YANG models – Industry and Cisco

- **IETF**
  - Interface management [RFC 7223]
  - IP management [draft-ietf-netmod-ip-cfg]
  - System management [draft-ietf-netmod-system-mgmt]
  - SNMP configuration [draft-ietf-netmod-snmp-cfg]
  - Generic OAM [Cisco Involvement, draft-tissa-netmod-oam]
  - OSPF [Cisco Involvement, draft-yeung-netmod-ospf-01]
  - BGP [Cisco Involvement, draft-zhdankin-netmod-bgp-cfg-00]
  - IPFIX configuration [Cisco involvement, RFC6728]
  - ACL configuration [Cisco involvement, draft-huang-netmod-acl-03]
  - Network topology [Cisco involvement, draft-clemm-i2rs-yang-network-topo-00.txt]
  - Routing management [draft-ietf-netmod-routing-cfg]
  - RIB [I2RS] [Cisco involvement, draft-clemm-i2rs-yang-network-topo-00]
  - Netconf monitoring [RFC6022], Netconf access control [RFC6536]

- **Cisco**
  - PIM, IPSLA, L2VPN, VLAN, DNA, Synthetic models XR
  - Cablelabs: CCAP (Converged Cable Access Point)
  - ONF: Openflow Switch Configuration (OF-Config)
  - MIBs (for monitoring data) via SMIv2 ->YANG conversion

- **YANG@CISCO to be supported over NETCONF, REST, or XMPP**
  - YANG modules of interest
    - draft-ietf-netmod-system-mgmt
    - draft-ietf-netmod-interfaces-cfg
    - draft-ietf-netmod-ip-cfg
    - draft-ietf-netmod-routing-cfg
    - draft-ietf-ipfix-configuration-model

- **Customer-driven modules for VLAN, QoS, environment, and ACL configuration**
Introduction to RESTCONF
RESTCONF

- Still an emerging story (draft-bierman-netconf-restconf-4)
- RESTful protocol to access YANG defined data
- Representational State Transfer, i.e. server maintains no session state
- URIs reflect data hierarchy in a Netconf datastore
- HTTP as transport
- Data encoded with either XML or JSON
- Operations

<table>
<thead>
<tr>
<th>RESTCONF</th>
<th>Netconf</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>&lt;get-config&gt;, &lt;get&gt;</td>
</tr>
<tr>
<td>POST</td>
<td>&lt;edit-config&gt; (“create”)</td>
</tr>
<tr>
<td>PUT</td>
<td>&lt;edit-config&gt; (“replace”)</td>
</tr>
<tr>
<td>PATCH</td>
<td>&lt;edit-config&gt; (“merge”)</td>
</tr>
<tr>
<td>DELETE</td>
<td>&lt;edit-config&gt; (“delete”)</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>(discover supported operations)</td>
</tr>
<tr>
<td>HEAD</td>
<td>(get without body)</td>
</tr>
</tbody>
</table>
YANG Mapping to JSON

- JSON is a popular compact and easy to parse data format used by many REST APIs
- Subset of YANG compatible XML documents can be translated to JSON text
- Translation driven by YANG data model (must be known in advance)
- YANG datatype information is used to translate leaf values to the most appropriate JSON representation
- Slightly more compact (irrelevant with compression)
- Increased human readability (less noise)
YANG mapping to JSON vs XML

**JSON – 214 octets***

```json
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "ethernetCsmacd",
        "location": "0",
        "enabled": true,
        "if-index": 2
      },
      {
        "name": "eth1",
        "type": "ethernetCsmacd",
        "location": "1",
        "enabled": false,
        "if-index": 2
      }
    ]
  }
}
```

**XML – 347 octets***

```xml
<interfaces xmlns:="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  <interface>
    <name>eth0</name>
    <type>ethernetCsmacd</type>
    <location>0</location>
    <enabled>true</enabled>
    <if-index>2</if-index>
  </interface>
  <interface>
    <name>eth1</name>
    <type>ethernetCsmacd</type>
    <location>1</location>
    <enabled>false</enabled>
    <if-index>7</if-index>
  </interface>
</interfaces>
```

*all white space removed*
RESTCONF Example

C: GET /restconf/operational/opendaylight-inventory:nodes HTTP/1.1
C: Host: example.com

S: HTTP/1.1 200 OK
S: Date: Fri, 6 June 2014 17:01:00 GMT
S: Server: example-server
S: Content-Type: application/json
S:
S: {
S:   "nodes": {
S:     "node": [
S:       {
S:         "flow-node-inventory:hardware": "Test vSwitch",
S:         "flow-node-inventory:software": "1.1.0",
S:         "id": "openflow:1",
S:         "flow-node-inventory:switch-features": {
S:           "flow-node-inventory:capabilities": [
S:             "flow-node-inventory:flow-feature-capability-flow-stats",
S:             "flow-node-inventory:flow-feature-capability-port-stats"
S:           ],
S:           "flow-node-inventory:maxBuffers": 256,
S:           "flow-node-inventory:maxTables": 255
S:         }
S:       }
S:   }
S: }
S:}