lieter_

pieterlexis

YANG and NETCONF for System Administration?

Pieter Lexis

NLNOG Day 2021 Amsterdam, September 2

Agenda

Introduction

whoami

Pieter Lexis

- SysAdmin by training, developer by accident[1]
- Senior PowerDNS Engineer at PowerDNS
- Responsible for CI/CD, deployment automation, packaging & more

POWERDNS

[1] Note the lack of "network engineer"

Configuration Management

No-one is, or should be, touching systems’ shells anymore

Configuration Management System Examples

- cfEngine (1993)
- bcfg2 (2004)
- Puppet (2005)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- mgmt config (2016)

Configuration Management Properties

- Declaritive
- Idempotent
- Convergent
- Code re-usability (modularity)

(Not every systems has all these properties)

Configuration Management Systems and Network Devices

(Based on the documentation)

- Mostly fancy CLI wrappers
- Not truly declarative
- Paradigms don’t match

Why YANG and NETCONF?

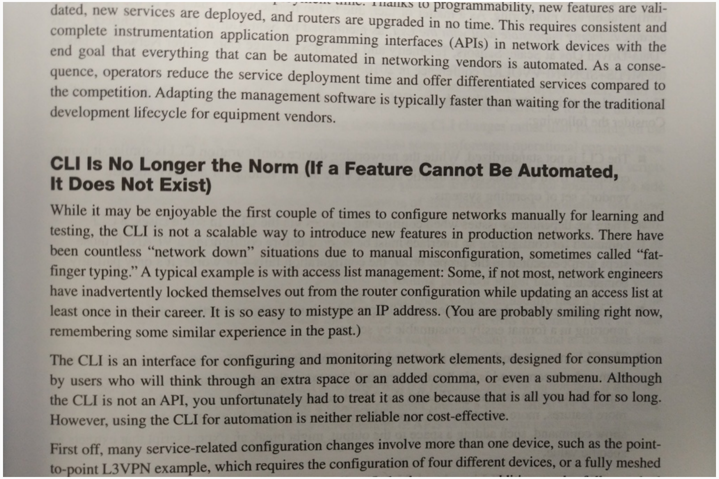
Network Device Configuration

- CLI differs between vendors
- Vendors have different configuration APIs
- SNMP has its share of drawbacks
- Few standardized MIBs, no “common” MIBs

RFC 3535, §3 “Operator Requirements”

- Configure the network, not separate devices
- Ability to perform transactions across devices
- Configuration should be stored centrally
- Common features between devices should have the same schema
- Separation between configuration, state, and statistics data

“Automation should be easy”



dated, new services are deployed, and routers are upgraded in no time. This requires consistent and complete instrumentation application programming interfaces (APIs) in network devices with the end goal that everything that can be automated in networking vendors is automated. As a consequence, operators reduce the service deployment time and offer differentiated services compared to the competition. Adapting the management software is typically faster than waiting for the traditional development lifecycle for equipment vendors.

CLI Is No Longer the Norm (If a Feature Cannot Be Automated, It Does Not Exist)

While it may be enjoyable the first couple of times to configure networks manually for learning and testing, the CLI is not a scalable way to introduce new features in production networks. There have been countless "network down" situations due to manual misconfiguration, sometimes called "fat-finger typing." A typical example is with access list management: Some, if not most, network engineers have inadvertently locked themselves out from the router configuration while updating an access list at least once in their career. It is so easy to mistype an IP address. (You are probably smiling right now, remembering some similar experience in the past.)

The CLI is an interface for configuring and monitoring network elements, designed for consumption by users who will think through an extra space or an added comma, or even a submenu. Although the CLI is not an API, you unfortunately had to treat it as one because that is all you had for so long. However, using the CLI for automation is neither reliable nor cost-effective.

First off, many service-related configuration changes involve more than one device, such as the point-to-point L3VPN example, which requires the configuration of four different devices, or a fully meshed

Figure 1: From "Network Programming with YANG", by Claise, Clarke, and Lindblad

The IETF Solution

NETCONF – Configuration Manipulation Protocol
YANG – Modeling Language

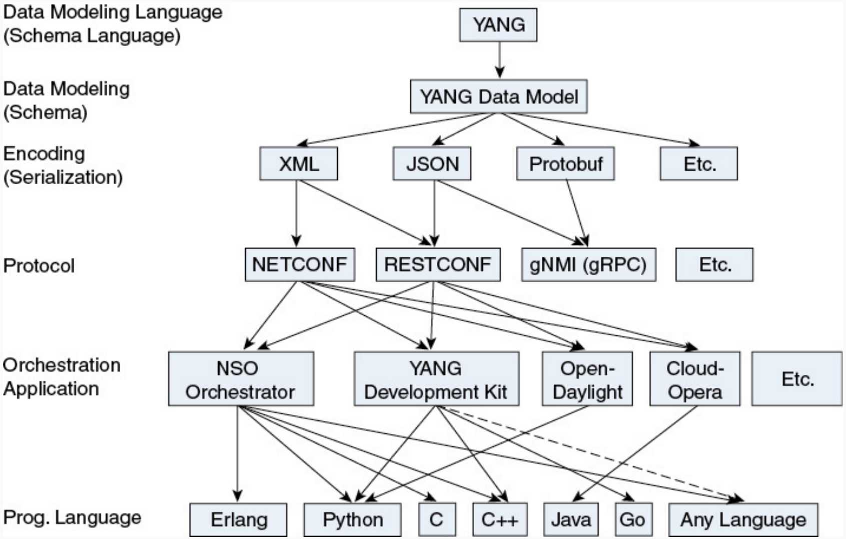
RFC 4741 – “NETCONF Configuration Protocol”, December 2006
RFC 6020 – “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”, October 2010
RFC 6241 – “Network Configuration Protocol (NETCONF)”, June 2011
RFC 6244 – “An Architecture for Network Management Using NETCONF and YANG”, June 2011
RFC 7950 – “The YANG 1.1 Data Modeling Language”, August 2016
RFC 7951 – “JSON Encoding of Data Modeled with YANG”, August 2016
...

NETCONF: Configuration Protocol

NETCONF architecture

```
         Layer                 Example
    +-------------+      +---------------------+      +--------------------+
(4) |   Content   |      |  Configuration data |      |  Notification data |
    +-------------+      +---------------------+      +--------------------+
           |                       |                          |
    +-------------+      +---------------------+              |
(3) | Operations  |      |    <edit-config>    |              |
    +-------------+      +---------------------+              |
           |                       |                          |
    +-------------+      +---------------------+      +--------------------+
(2) |  Messages   |      |  <rpc>, <rpc-reply> |      |   <notification>   |
    +-------------+      +---------------------+      +--------------------+
           |                       |                          |
    +-------------+      +-------------------------------------------------+
(1) |  Transport  |      |      SSH, TLS, BEEP/TLS, SOAP/HTTP/TLS, ...     |
    +-------------+      +-------------------------------------------------+
```

NETCONF architecture



NETCONF Protocol Features

- CRUD operations for configuration
- Configuration is *fully* declarative
- Configuration and operational state
- Network-wide transactions, with full ACID properties
- Rollback support
- One protocol to implement in orchestrators and controllers
- Inner workings of the device are invisible

What speaks NETCONF?

- Network devices
  - Alcatel Lucent
  - Arista
  - Brocade
  - Cisco
  - HP
  - Huawei
  - Juniper
- Orchestration frameworks
- Network Management Systems
- Configuration Management Software

YANG: Yet Another Next Generation

YANG Language

- Many built-in types
- Reusable types
- Structured types
- Derived types
- Data constraints
- Modular
- Versioned

my-example-model.yang

```
grouping endpoint {
  description
    "An IP endpoint, including the port";
  leaf ip-address {
    type inet:ip-address-no-zone;
    mandatory true;
  }
  leaf port {
    type inet:port-number;
  }
}
```

YANG Module

- Hierarchical structure
- Import other modules
- Refine types

my-example-model.yang

```
container listen-addresses {
  list listen-address {
    key "name";
    leaf name {
      type string;
    }
    unique "ip-address port";
    uses endpoint {
      refine port {
        default 25;
      }
    }
  }
}
```

YANG Module — Statistics

my-example-model.yang

```
  container counters {
    config false;
    leaf connection-count {
      type uint32;
    }
  }
```

Hierarchy

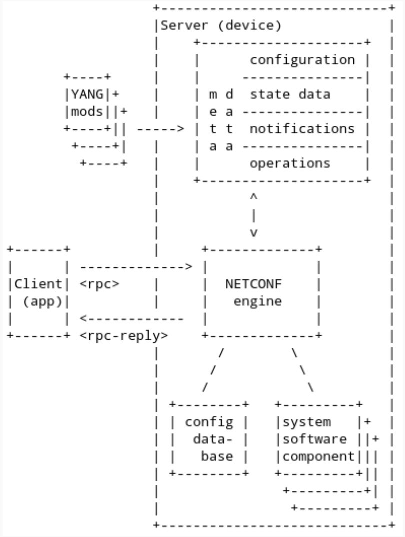
```
module: my-example-model
  +--rw listen-addresses
  |  +--rw listen-address* [name]
  |     +--rw name          string
  |     +--rw ip-address    inet:ip-address
  |     +--rw port?         inet:port-number
  +--ro counters
     +--ro connection-count?   uint32
```

Reuse of modules

- Large collection of modules
  - Interface types
  - IP addresses
  - TLS server and client configuration (including X.509)
  - SSH server and client configuration
- Used by vendors to model devices
- Published e.g. on YangModels/yang

YANG and NETCONF on *NIX

Server Architecture



```
                  +----------------------------+
                  |Server (device)             |
                  |    +--------------------+  |
                  |    |      configuration |  |
       +----+     |    |     ---------------|  |
       |YANG|+    |    | m d  state data    |  |
       |mods||+   |    | e a ---------------|  |
       +----+|| -----> | t t  notifications |  |
        +----+|   |    | a a ---------------|  |
         +----+   |    |      operations    |  |
                  |    +--------------------+  |
                  |           ^                |
                  |           |                |
                  |           v                |
+------+          |     +-------------+        |
|      | -------------> |             |        |
|Client| <rpc>    |     |  NETCONF    |        |
| (app)|          |     |   engine    |        |
|      | <------------  |             |        |
+------+ <rpc-reply>    +-------------+        |
                  |       /        \           |
                  |      /          \          |
                  |     /            \         |
                  | +--------+   +---------+   |
                  | | config |   |system   |+  |
                  | |  data- |   |software ||+ |
                  | |   base |   |component||| |
                  | +--------+   +---------+|| |
                  |               +---------+| |
                  |                +---------+ |
                  +----------------------------+
```

Does it make sense?

- Many applications could be “network functions”
- With the right orchestrator, have “versioned infra”
- Even without NETCONF, YANG is a powerful config language
- No need for something else when already using YANG/NETCONF

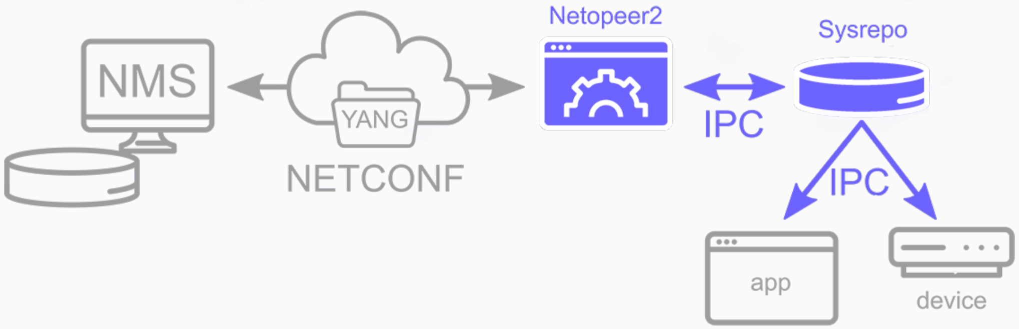
Server Software

Software and libraries

- `libyang` – YANG parser and toolkit
- `sysrepo` – YANG Datastore
- `Netopeer2` – NETCONF server and client

- Written and maintained by CESNET
- Written in C
- Wrappers for other languages

sysrepo



- Implements all YANG datastores
- Plugins “claim” parts of the tree

Does it work?

Yes

Does it work?

Yes

(-ish)

Does it work?

Yes

(-ish)

- ISC Kea DHCP in production
- Many plugins exist
- The C-API is… something
- Incompatible versions of Sysrepo (0.7, 1.0, 2.0)
- Not packaged for any OS
- Badly written plugins can crash the process

Wrap up

In conclusion

- YANG and NETCONF are (becoming) industry standard for configuration
- Concepts map quite well for system configuration management
- But the software is not truly production-ready

Questions?

References and further reading

- `http://www.netconfcentral.org/modulelist`
- `https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html`
- `https://www.sysrepo.org/`
- `https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2017/pdf/DEVNET-1070.pdf`
- “Network Programming with YANG”, Claise, Clarke, and Lindblad

Backup slides

Backup slides

YANG types

YANG Models — Tree elements

- *Grouping* — Set of nodes for re-use
- *Container* — A set of related nodes
- *List* — A keyed set of nodes
- *Leaf-list* — List of a single item

YANG Models — Built-in types

- (u)int8, (u)int16, (u)int32, (u)int64
- decimal64
- string
- bits
- boolean
- enumeration
- union

YANG Models — Other modeling tools

- Import: Enables re-use of models
- Augment: Add new nodes to previously defined nodes
- Grouping: Set of nodes for re-use
- Container: Group of related nodes
- Feature: Allows marking part of the tree as optional

Types — Derived types: Constraints

ietf-inet-types@2013-07-15.yang

```
  typedef port-number {
    type uint16 {
      range "0..65535";
    }
    description
     "The port-number type represents a 16-bit port number of an
      Internet transport-layer protocol such as UDP, TCP, DCCP, or
      SCTP.  Port numbers are assigned by IANA.  A current list of
      all assignments is available from <http://www.iana.org/>.

      Note that the port number value zero is reserved by IANA.  In
      situations where the value zero does not make sense, it can
      be excluded by subtyping the port-number type.
      In the value set and its semantics, this type is equivalent
      to the InetPortNumber textual convention of the SMIv2.";
```

Types — Derived types: Deriving further

ietf-inet-types@2013-07-15.yang

```
  typedef ipv4-address {
    type string {
      pattern
        '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
      +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'
      + '(%[\p{N}\p{L}]+)?';
    }
```

ietf-inet-types@2013-07-15.yang

```
  typedef ipv4-address-no-zone {
    type inet:ipv4-address {
      pattern '[0-9\.]*';
    }
```

Types — Union

ietf-inet-types@2013-07-15.yang

```
typedef ip-address-no-zone {
  type union {
    type inet:ipv4-address-no-zone;
    type inet:ipv6-address-no-zone;
  }
  description
   "The ip-address-no-zone type represents an IP address and is
    IP version neutral.  The format of the textual representation
    implies the IP version.  This type does not support scoped
    addresses since it does not allow zone identifiers in the
    address format.";
  reference
   "RFC 4007: IPv6 Scoped Address Architecture";
}
```

Moving around in the tree

- Addressing uses XPath
- XPaths can contain one or more expressions
- Expressions can also do arithmetic

```
/my-example-model:listen-addresses/listen-address[name='localhost']/ip-
↪  address
/my-example-model:listen-addresses/listen-address[name='localhost']/port
```

```
/ietf-interfaces:interfaces/interface[name='iface1']/ietf-ip:ipv4/ietf-
↪  ip:address[ietf-ip:ip='10.0.0.1']
```

```
  /ietf-interfaces:interfaces/interface[position() =
  ↪  last()]/ietf-ip:ipv4/*
```

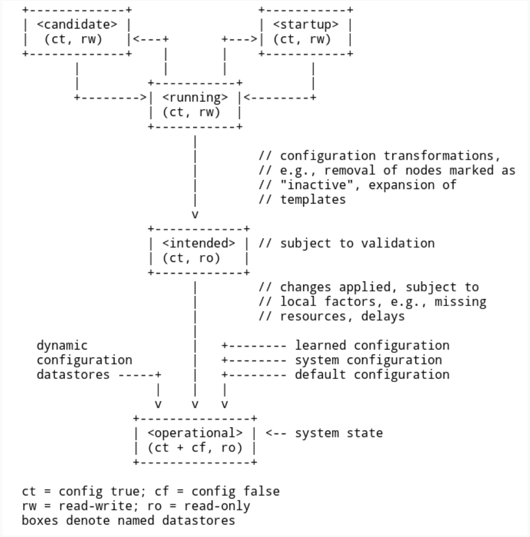
Backup slides

NETCONF Datastores

Datastores

- Startup — Config to use upon boot
- Running — Current configuration
- Candidate — Used for staging config changes
- Operational – Contains the config and state of the system

Datastores



```
+-------------+                 +-----------+
| <candidate> |                 | <startup> |
|  (ct, rw)   |<---+       +--->| (ct, rw)  |
+-------------+    |       |    +-----------+
       |           |       |           |
       |         +-----------+         |
       +-------->| <running> |<--------+
                 | (ct, rw)  |
                 +-----------+
                       |
                       |        // configuration transformations,
                       |        // e.g., removal of nodes marked as
                       |        // "inactive", expansion of
                       |        // templates
                       v
                 +------------+
                 | <intended> | // subject to validation
                 | (ct, ro)   |
                 +------------+
                       |        // changes applied, subject to
                       |        // local factors, e.g., missing
                       |        // resources, delays
                       |
  dynamic              |   +-------- learned configuration
  configuration        |   +-------- system configuration
  datastores -----+    |   +-------- default configuration
                  |    |   |
                  v    v   v
               +---------------+
               | <operational> | <-- system state
               | (ct + cf, ro) |
               +---------------+

ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote named datastores
```

Backup slides

pdns-sysrepo

pdns-sysrepo

- Configures PowerDNS Authoritative Server
- Stores zone-data in sysrepo (“just configuration” for the operator’s perspective)
- Exposes a Remote Backend endpoint for PowerDNS for zone data
- VM acts as a single DNS Server that is configured by NETCONF
- PowerDNS/pdns-sysrepo

pdns-sysrepo

```
                         +--------- DNS Server ------------------------+
                         |                                             |
[NMS] <= NETCONF => [Netopeer] <=> [sysrepo] <=> [   pdns-sysrepo   ]  |
                         |                        ^    |         | ^   |
                         |                  calls |    | writes  | |   |
                         |                        v    |         | |   |
                         |          /-------[systemd]  v         | |   |
                         |         | restarts     <config file>  | |   |
                         |         |                  ^          | |   |
                         |         |      /----------/           | |   |
                         |         |     /   reads               | |   |
                         |         |    /                       /  |   |
                         |         |   /  /--------------------/   |   |
                         |         |  /   |       API calls        |   |
                         |         |  |   |                        |   |
                         |         |  |   |       RemoteBackend   /    |
                         |         |  |   |   /------------------/     |
                         |         v  v   v  /                         |
                         +-----[  pdns_server  ]-----------------------+
```