NETWORK AUTOMATION AND PROGRAMMABILITY:

Reality Versus The Vendor Hype When Considering Legacy And NFV Networks

P. Moore
NANOG 70
June 7, 2017
INTRODUCTION

- Automation is a Journey
- Traditional vs Future Automation
- Vendor Hype
- Network Reality
Modest improvements and limited benefits to this track.
 PROGRAMMABILITY: WHAT IS IT?

• Software-like interfaces to network
  – APIs
  – NETCONF
• Intelligent Models/Templates
  – YANG
  – YAML
  – TOSCA
**TERMINOLOGY**

- **NETCONF** – NETwork CONFiguration – created to achieve config goals SNMP could not

- **YANG** – Yet Another Next Generation – modeling language for data sent via NETCONF

- **YAML** – YAML Ain’t Markup Language – modeling used by tools such as NAPALM and OpenStack HEAT (among others) to map items to native config
TERMINOLOGY

• NFV – Network Function Virtualization – virtualization of network devices

• TOSCA – Topology and Orchestration Specification for Cloud Applications – modeling (YAML) of cloud based network services.
TRADITIONAL SCRIPT-BASED

- Script-based
  - Not as scalable
  - Labor intensive to maintain
  - Avoids need for IT involvement
  - Human Driven Automation is best case
FUTURE MODEL-BASED

- Model-based automation
  - Scalable
  - Smaller number of Models to manage
  - May require IT involvement
  - Policy/Event Driven Automation is goal
THE HYPE

• Just use NETCONF and you can automate everything
• YANG is easy and standard
• NFV is going to virtualize everything, which means it will be automated
• Automation is easy
REALITY: NETCONF/YANG

- Network Devices:
  - Most are not NETCONF compliant
  - Some vendors are approaching it from an API perspective versus NETCONF, but:
    - Most have no API access available
  - All YANG is not created equally
typedef ip_types {
    type enumeration {
        enum ipv4;
        enum ipv6;
    }
}typedef interface_types {
    type enumeration {
        enum Ethernet;
        enum FastEthernet;
        enum GigabitEthernet;
        enum TenGbE;
        enum xe;
    }
}

container intental_policy {
    common:action acl-load {
        common:info "Load acls with bulk payload";
        common:actionpoint intental-policy-acl-load;
        input {
            leaf payload {
                type string;
                description "bulk payload";
            }
        }
        output {
            leaf config {
                type string;
                description "Native config response";
            }
        }
    }
}

list access_list {
    uses ncs:service-data;
    ncs:servicepoint "intental-policy-access-control-list";
    common:info "Intental Policy: Access List Entries";
    key "name device_name ip_type";
    leaf name {
        type string;
    }
    leaf device_name {
        type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
        }
    }
    leaf ip_type {
        type ip_types;
        common:info "IPV4 or IPV6 Access List";
    }
    list rule {
        key order;
        ordered-by user;
        leaf order {
            type string;
            common:info "The order position of the rule.";
        }
        leaf action {
            type enumeration{
                enum permit;
                enum deny;
            }
            common:info "Permit or Deny the rule";
        }
        leaf protocol {
            type string;
            common:info "The protocol of the rule. Example: icmp, ipv4.tcp, udp, etc.";
        }
        leaf-list source_host {
            type string;
            common:info "The source host in IPV4, IPV6, or range.";
        }
        leaf-list destination_host {
            type string;
            common:info "The source host in IPV4, IPV6, or range.";
        }
    }
}
admin@ncs% set intental_policy access_list acl_1 mgracl_junos@Iptd rule 10 action deny destination_host 4.21.1.10 source_host [ 3.3.3.3 4.4.4.4 ] protocol tcp service ssh
[ok][2017-03-28 13:41:07]

[edit]
admin@ncs% commit dry-run outformat native
native {
  device {
    name mgracl_junos
    data <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      message-id="2">
      <edit-config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
        target="candidate">
        <test-option>test-then-set</test-option>
        <error-option>rollback-on-error</error-option>
        <config>
          <configuration xmlns="http://xml.juniper.net/jxm/1.1/jxm">
            <firewall>
              <family>
                <inet>
                  <filter>
                    <name>acl_1</name>
                    <term>
                      <name>term10</name>
                      <then>
                        <discard/>
                      </then>
                    </term>
                  </filter>
                </inet>
              </family>
            </firewall>
          </configuration>
        </config>
      </edit-config>
    </data>
  </device>
}</from>
</term>
</filter>
</interface>
</firewall>
</configuration>
</config>
</edit-config>
</rpc>
[ok][2017-03-28 13:41:09]

[edit]
admin@ncs%
CONCLUSION: NETCONF/YANG

- Operational complexity is high
- To be effectively used management tools are needed:
  - Cisco NSO (formerly Tail-f) seems to be the best option from a vendor at the moment
- Not supported widely enough to be useful, yet
- Intersection of developer and network engineering skillsets
NETCONF/YANG TOOLING

**Commercial:**
- Applied Informatics
- GoAhead
- SNMP Research
- Cisco/Tail-f Systems
- Many NFV MANO solutions use YANG

**Open Source:**
- Ncclient (client)
- netopeer (client/server)
- YencaP (client/server)
- Yuma (client/server)
- YANG on top of NAPALM – in beta
• NFV is a focus for many, but NFV ≠ Programmable
  – NFV is still in its infancy
  – Virtualization is near maturity, BUT…
  – The management and tooling on top of it is not
• Collision of IT Ops and network engineering skillsets
THE ANSWER?

- Creating an automation framework that can be flexible and grow with the emerging technology
- Programmability matters, the technology enabling it does not...much
HIGH LEVEL APPROACH

Network Automation Control

APIs

Orchestration

MANO

Interpreters

Programmability

Virtual Network Devices ≠ Programmable
TOOLING NEEDS & OPTIONS

- Interpreter: translates models/templates to device understood commands
  - NAPALM
  - Ansible Network Modules
  - Chef Cookbooks for Cisco and Juniper
  - Puppet Modules
  - Proprietary: Cisco NSO NEDs, etc.
TOOLING NEEDS & OPTIONS

• Orchestration: provides modeling/templating capabilities and communicates to Network via Interpreters
  – Ansible
  – Open-O
  – OpenDaylight
  – Proprietary: Cisco NSO, Blue Planet, Affirmed Networks, etc.
TOOLING NEEDS & OPTIONS

- Automation Platform: Combines workflow, scripting, and API aggregation to provide ops automation applications
  - Activiti
  - Red Hat JBoss BPM
  - Proprietary: Pronghorn, ServiceNow, Remedy, Resolve, etc.
OPEN SOURCE OPTIONS

Activiti

Network Automation Workflow

JBoss

Salt

Orchestration Layer

Chef

Layer

OpenContrail

OpenBaton

OpenStack

ONAP

OSM

Ansible

Interpreters

Programmability

Interpreter

Interpreter

Physical Network Devices

Virtual Network Devices

NAPALM

NAPALM
CONCLUSIONS

• Listen to what vendors say with a grain of salt, but use the information provided for inspiration
• Consider open source tools that can do what is promised
• Set goals for automation – you never hit a target if there isn’t one to aim for
• Thoroughly evaluate your need versus vendor and open source possibilities
QUESTIONS?
REFERENCES

- http://networkop.co.uk/blog/2017/01/25/netconf-intro/
- https://medium.com/@anthonypjshaw/netops-with-saltstack-and-pynso-3ce45211501#.yqa0x43us
- Network to Code Slack Channel: https://networktocode.herokuapp.com/