Automated Config Generation with Python and YAML

Jere Julian / Daryn Johnson
June 13, 2018

Network to Code Overview

Network to Code

- Founded in mid 2014
  - First company to deliver network automation training
- Network Automation Solution Provider
  - Next-gen consulting and integration company
  - Focus on automation/programmability and reducing operational inefficiencies
- Vendor Independent
  - Infrastructure (Cisco, Juniper, Arista, HP, Cumulus, F5 etc.)
  - Tools (Ansible, Salt, Python, StackStorm)

Community Presence

- Team
  - Industry Experts and Evangelists
  - Team software developers and network engineers who made the transition to software engineers
- Pioneers in Network Automation Community
  - Developing in Ansible since 2014
  - Major content contributors and presenters at industry conferences, meet-ups, and online communities.
  - Over 5,000 members in Network to Code Slack Community
  - Open source contributors
    - Largest contributor of modules outside of Red Hat
  - Automation Modules, Ansible Playbooks, and Config templates, for our automation and network infrastructure vendors

We believe in the value network automation brings to our customers and we share our knowledge with our community.

Network to Code Services Offerings

Workflow Automation & Optimization

- Review Network Operations Workflows
- Optimize
- Drive Down operational inefficiencies

Software Development

- Custom network Applications
- DevOps Tool Plug-Ins & Integrations
- Helps Eliminate repetitive tasks

Training / Workshops

- Python for Network Engineers
- DevOps Tools
- Network Device APIs
- Private/Public Courses
- Bootcamps

Our Vision

Changing the way networks are deployed, consumed, and managed while enabling the network engineer of the future.

Building configurations with templates and data

```
pip install pyaml
pip install jinja2
git clone https://github.com/networktocode/clus-2018
cd clus-2018
```

Breaking down configs

Patterns

- Common blocks between devices
- Common blocks within a config
- Unique information

Data vs. Function

- Separate what from how
- What changes vs. what is more static
- Data → Template → Device

Structured Data

Structured data

```
Software
  kickstart: version 7.3(1)D1(1) [build 7.3(1)D1(0.10)]
  system:    version 7.3(1)D1(1) [build 7.3(1)D1(0.10)]


Hardware
  cisco NX-OSv Chassis ("NX-OSv Supervisor Module")
  Intel(R) Xeon(R) CPU E5-2670 with 4002196 kB of memory.
  Processor Board ID TM602D0DC9B

  Device name: nxos-spine2
  bootflash:    1582402 kB

Kernel uptime is 3 day(s), 22 hour(s), 35 minute(s), 6
second(s)
```


```
{
 "loader_ver_str": "N/A",
  "kickstart_ver_str": "7.3(1)D1(1) [build 7.3(1)D1(0.10)]",
  "sys_ver_str": "7.3(1)D1(1) [build 7.3(1)D1(0.10)]",
  "chassis_id": "NX-OSv Chassis",
  "module_id": "NX-OSv Supervisor Module",
  "cpu_name": "Intel(R) Xeon(R) CPU E5-2670",
  "memory": 4002196,
  "mem_type": "kB",
  "proc_board_id": "TM602D0DC9B",
  "host_name": "nxos-spine2",
  "bootflash_size": 1582402,
  "kern_uptm_days": 3,
  "kern_uptm_hrs": 22,
  "kern_uptm_mins": 35,
  "kern_uptm_secs": 32,
}
```

XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:sysmgrcli">
 <nf:data>
  <show>
   <version>
    <__XML__OPT_Cmd_sysmgr_show_version___readonly__>
     <__readonly__>
      <sys_ver_str>7.3(1)D1(1) [build 7.3(1)D1(0.10)]</sys_ver_str>
      <chassis_id>NX-OSv Chassis</chassis_id>
      <module_id>NX-OSv Supervisor Module</module_id>
      <host_name>nxos-spine2</host_name>
      <kern_uptm_days>3</kern_uptm_days>
    </__readonly__>
    </__XML__OPT_Cmd_sysmgr_show_version___readonly__>
   </version>
  </show>
 </nf:data>
</nf:rpc-reply>
```

JSON

```
# show version | json
{
  "kickstart_ver_str": "7.3(1)D1(1) [build 7.3(1)D1(0.10)]",
  "sys_ver_str": "7.3(1)D1(1) [build 7.3(1)D1(0.10)]",
  "chassis_id": "NX-OSv Chassis",
  "module_id": "NX-OSv Supervisor Module",
  "cpu_name": "Intel(R) Xeon(R) CPU E5-2670",
  "memory": 4002196,
  "mem_type": "kB",
  "proc_board_id": "TM602D0DC9B",
  "host_name": "nxos-spine2",
  "bootflash_size": 1582402,
  "kern_uptm_days": 3,
  "kern_uptm_hrs": 22,
  "kern_uptm_mins": 35,
  "kern_uptm_secs": 32,
  "manufacturer": "Cisco Systems, Inc."
}
```

YAML

```
---
kickstart_ver_str: "7.3(1)D1(1) [build 7.3(1)D1(0.10)]"
sys_ver_str: "7.3(1)D1(1) [build 7.3(1)D1(0.10)]"
chassis_id: "NX-OSv Chassis"
module_id: "NX-OSv Supervisor Module"
cpu_name: "Intel(R) Xeon(R) CPU E5-2670"
memory: 4002196
mem_type: "kB"
proc_board_id: "TM602D0DC9B"
host_name: "nxos-spine2"
bootflash_size: 1582402
kern_uptm_days: 3
kern_uptm_hrs: 22
kern_uptm_mins: 35
kern_uptm_secs: 32
manufacturer: "Cisco Systems, Inc."
```

YAML

```
---
key1: value
key2: value2
# Some comments
My_list:
  - item0
  - item1
  - item2
```

Data model

- Representation of your network
  - Not the configuration
- Optimized to be readable, maintainable

Data model

```
---
hostname: nydc-r02-tor-a
Vlans:
  - 1:
    name: default
  - 100:
    name: test_100
  - 200:
    name: vm_network
```

Template capabilities

- Variable substitution
- Conditionals
- Iteration
- Inheritance

Jinja2 - basic

```
interface {{ port_name }}
   description {{ description }}
   no shutdown
```

Jinja2 – intermediate + data

```
{% for port in ports %}
   interface {{ port.name }}
   description {{ port.descr }}
   {% if port.vlan is defined-%}
    switchport access vlan {{ port.vlan }}
   {% else -%}
   no switchport access vlan
   {% endif -%}
{% endfor %}
```

```
---
- ports:
   - name: 'ethernet1/1'
     descr: host32-eth0
     vlan: 100
   - name: 'ethernet1/2'
```

Jinja2 Templates and Python

```
#!/usr/bin/env python
''' Render a template '''
from jinja2 import Template

template = Template("interface {{ intf }}\n shutdown")

for x in [1, 2]:
    print template.render(intf="Ethernet{}”.format(x))
```

```
pip install pyaml
pip install jinja2
git clone https://github.com/networktocode/clus-2018
cd clus-2018
```