



IT – NSO Training

Brandon Black, Elliot Wise, Patrick Huynh, Jason Belk

Agenda

- Day 1: What is NSO? Using NSO
- Day 2: XML & Yang
- Day 3: Basic Services
- Day 4: Advanced Services
- Day 5: Python



WELCOME TO DAY ONE!

Agenda

- CONTEXT
- WHY AUTOMATE? WHY NSO?
- THE WHAT & WHY OF NSO
- NSO COMPONENTS
- BASH BASICS

Training Expectations

- Learning is best when focused, laptops down
- Ask questions
- If unclear, please ask for clarification
- Spark room for question (1 trainer presenting, 1 on Spark)
- Labs are for learning, not merely tasks to be done

Context

Automation Maturity Model

NSO: Python API

Python Templates

Service Models

Ad-hoc /
Scripting

Re-useable
Frameworks

Orchestration



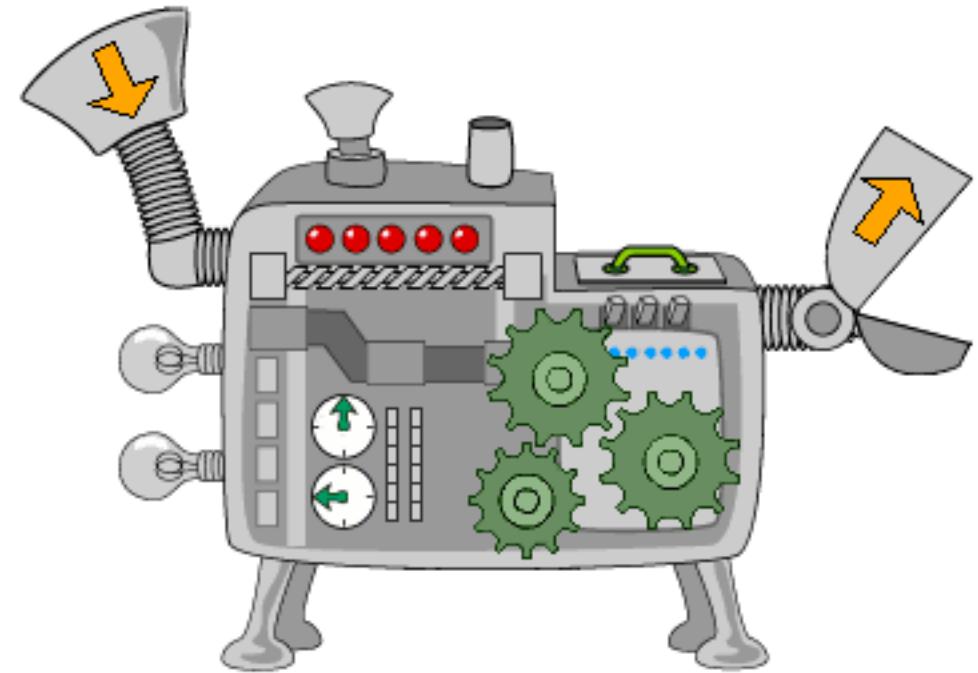
Engineer run
one-off scripts
and tools

Centrally managed
frameworks & templates
for faster development

Automated configuration lifecycle.
Creation, Modification and Removal
automated in one place

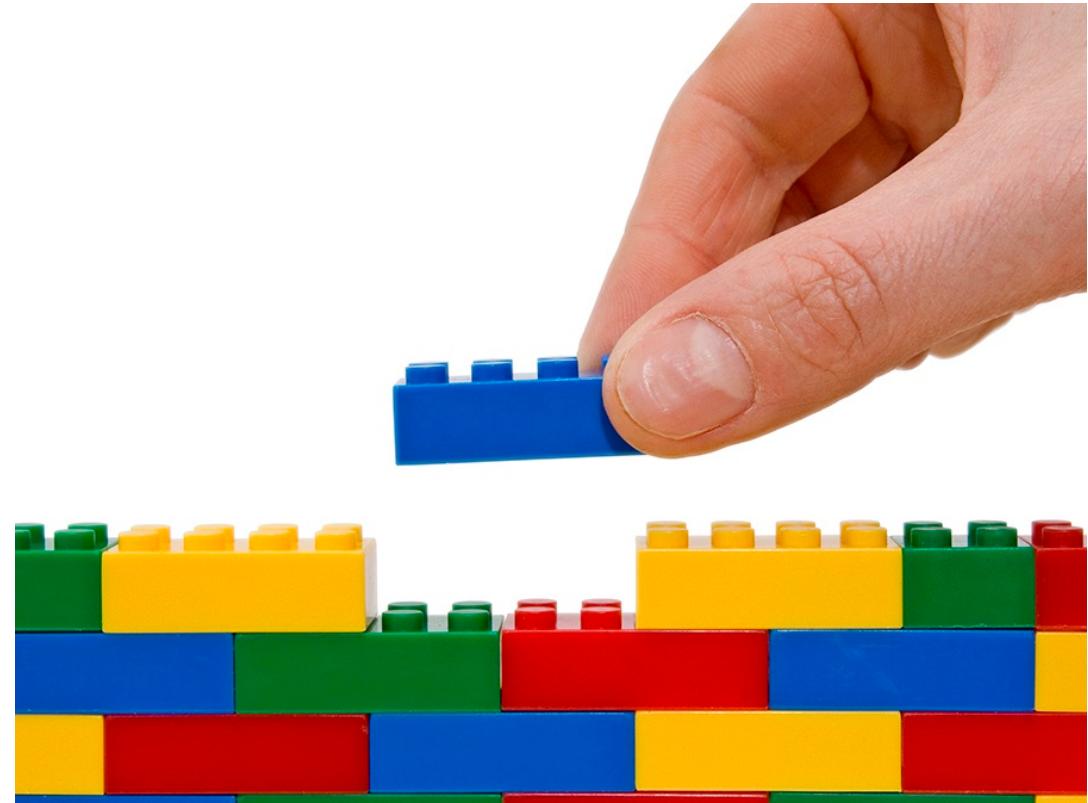
Automation Maturity Model – Ad Hoc / Scripting

- 1 slide ad-hoc slide example
- 802.1X Script used for EM/MM rollout, generating config
- N2i script
- Most python scripts (using Paramiko, Netmiko, Cisco-Connect)
- Built for a specific purpose, with some minor variations.



Automation Maturity Model – Re-Usable Frameworks

- Auto-config
- Templates
- Code scaffolds (insert your code here), NO2 modules, Cisco-Connect
- Ansible



Automation Maturity Model – Orchestration

- APIC-EM
- NSO
- ACI APIC



APIC-EM vs NSO Operational Comparison

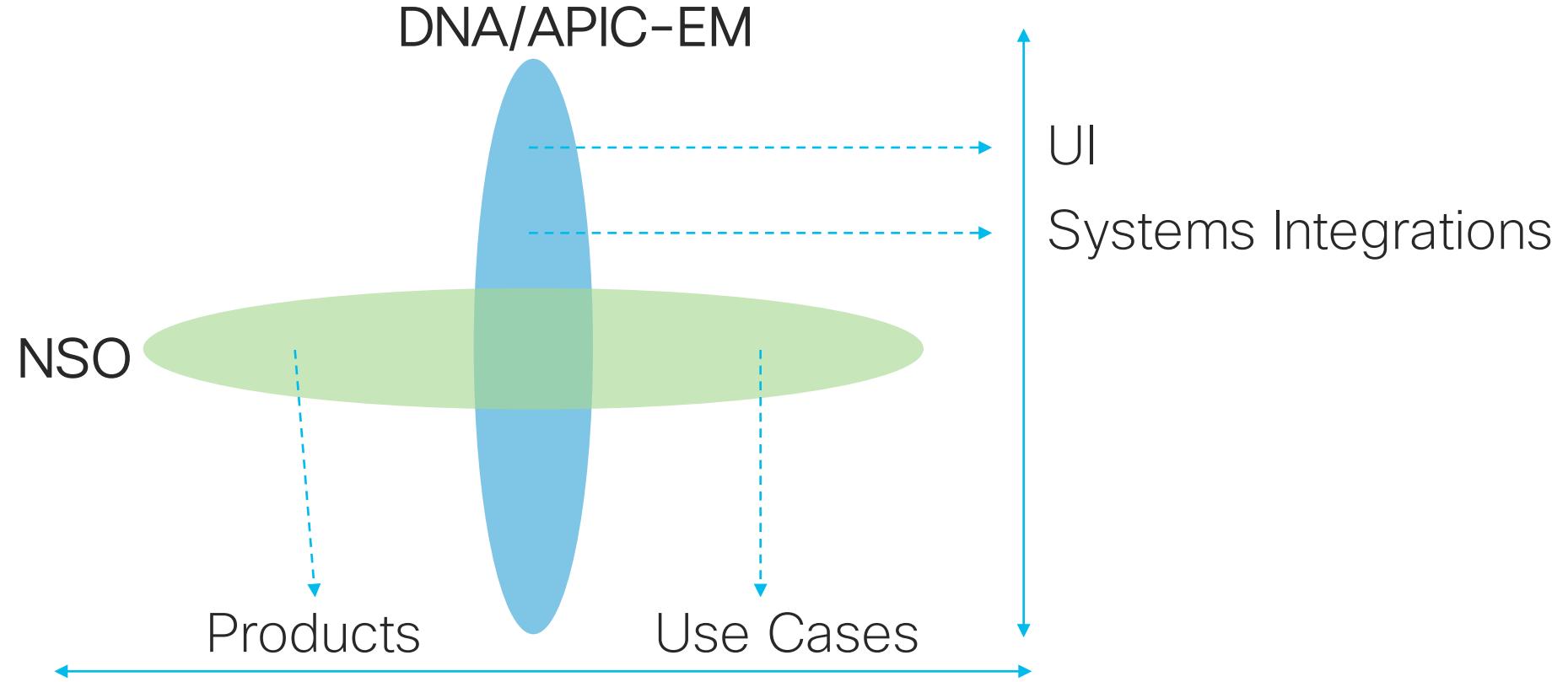
DNA/APIC-EM

- Turn-key
- Prescriptive
- Abstracted
- Designed to be more responsive to changing conditions
- Easy to adopt, limited scope

NSO

- No turn-key offerings
- Flexible
- Abstracted but user-defined
- More static, repetitive
- More work to adopt, limits are developers imagination

APIC-EM vs NSO Operational Comparison



Complete solution requires both platforms

What is one example from each of the 3 categories in the Automation Maturity Model ?



Demo – Script vs Orchestration

WHY AUTOMATE? WHY NSO?

Why Automate?

- Changing Skillset of a Network Engineer (everyone automates, not just a dedicated automation engineer)
- Network Infrastructure moving to Software-Defined Everything (NFV, SDA, etc.)
- Maintain and Create a predictable and uniform network, making operations much easier
- Speed to delivery at scale (making changes to 100s of devices)

Common Objections

- I am too busy to automate, or learn/practice automation skills
- I do not know where to start
- I get stuck setting up my environment
- I get interrupted when I finally get time to try
- I don't see how I can automate this task, it is just easier to do it manually, since I know that will work
- There seems to be too many competing platforms (ansible, NSO, python, NO2, etc.), I don't know which one to use
- I know how to use bash/SNMP/python/perl scripts, I don't need anything else

Engineers getting started in automation

How to fail at network automation?

1. Start with high-risk, difficult problems.
2. Assume an all-or-nothing mindset (everything has to be automated or nothing can be automated).
3. Try to reinvent everything yourself.
4. Superficially copy code and patterns without comprehension.
5. Fail to learn good debugging processes.
6. [Hugely] over-engineering the solution.

Engineers getting started in automation

How to fail at network automation?

7. Fail to apply things that you learned on a small scale.
8. Being too busy to automate.
9. Fail to learn how to reuse your code [longer term].
10. Fail to use available developer tools: Git, linters, unit-testing, CI-tools [longer term].

How to Succeed at Network Automation

- Carve out time in your week where you can learn and practice the skills without interruption (no cell phone, no jabber/spark/email, etc.)
- Have a humble and teachable attitude, being patient while adjusting to the learning curve
- Teach others what you are learning to reinforce the concepts
- Pick a very small task to start with
- Read lots of examples and follow the logic
- Consider not only the problem at hand, but the business process around it
- Don't be afraid to ask others for help if you get stuck
- Use version control (git) and do not hard code passwords in code
- Actively seek to apply your skills to your daily job
- Write code / templates with readability and reusability in mind

Every Network is Unique vs. Predictable



VS



(at least) Two Types of Automation

Configuration Management
(auto-config for example)

```
R1(config-ext-nacl)#do sh access-list OutBoundAccess
Extended IP access list OutBoundAccess
 10 permit ip 192.168.1.0 0.0.0.255 any
 11 deny tcp 192.168.2.0 0.0.0.127 any eq smtp
 12 deny tcp 192.168.2.0 0.0.0.127 any eq sunrpc
 13 deny tcp 192.168.2.0 0.0.0.127 any eq pop2
 14 deny tcp 192.168.2.0 0.0.0.127 any eq nntp
 15 deny tcp 192.168.2.0 0.0.0.127 any eq ftp
 16 deny tcp 192.168.2.0 0.0.0.127 any eq ftp-data
 17 deny tcp 192.168.2.0 0.0.0.127 any eq telnet
 18 deny tcp 192.168.2.0 0.0.0.127 any eq cmd
 19 deny tcp 192.168.2.0 0.0.0.127 any eq irc
 20 permit ip 192.168.2.0 0.0.0.255 any
 30 permit ip 192.168.3.0 0.0.0.255 any
 40 permit ip 192.168.4.0 0.0.0.255 any
 50 permit ip 192.168.5.0 0.0.0.255 any
R1(config-ext-nacl)#[
```

Operational Alerting
(EMAN Monitoring, Science Logic, etc)



WHAT IS NSO?

The Flexibility of NSO

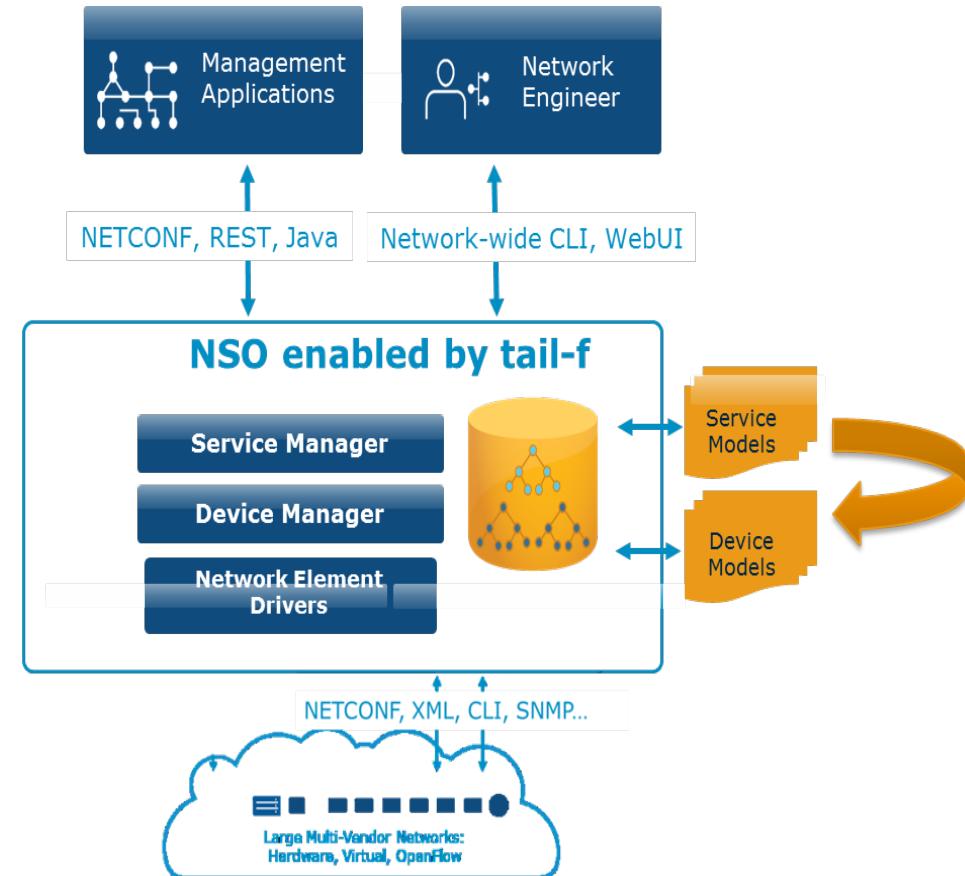
Do you want to build a house or a boat?

NSO gives the tools to do either (metaphorically).



Network Service Orchestrator Overview

Model Driven
Transactional
Automation Engine
Multi-Vendor
Built for Network engineers
API into the Network



Network Service Orchestrator Components

NorthBound API

REST

WebUI

Python & Java API

NSO Servers

Service Manager

Device Manager

Configuration Database

Multi-Vendor
Network Element Drivers
(NED)

IOS

IOS-XR

NX-OS

The “Service” in Network Service Orchestrator

A Network Service is a collection of configurations across one or multiple devices and servers that enable a capability.

An example is Basic Wireless Service:

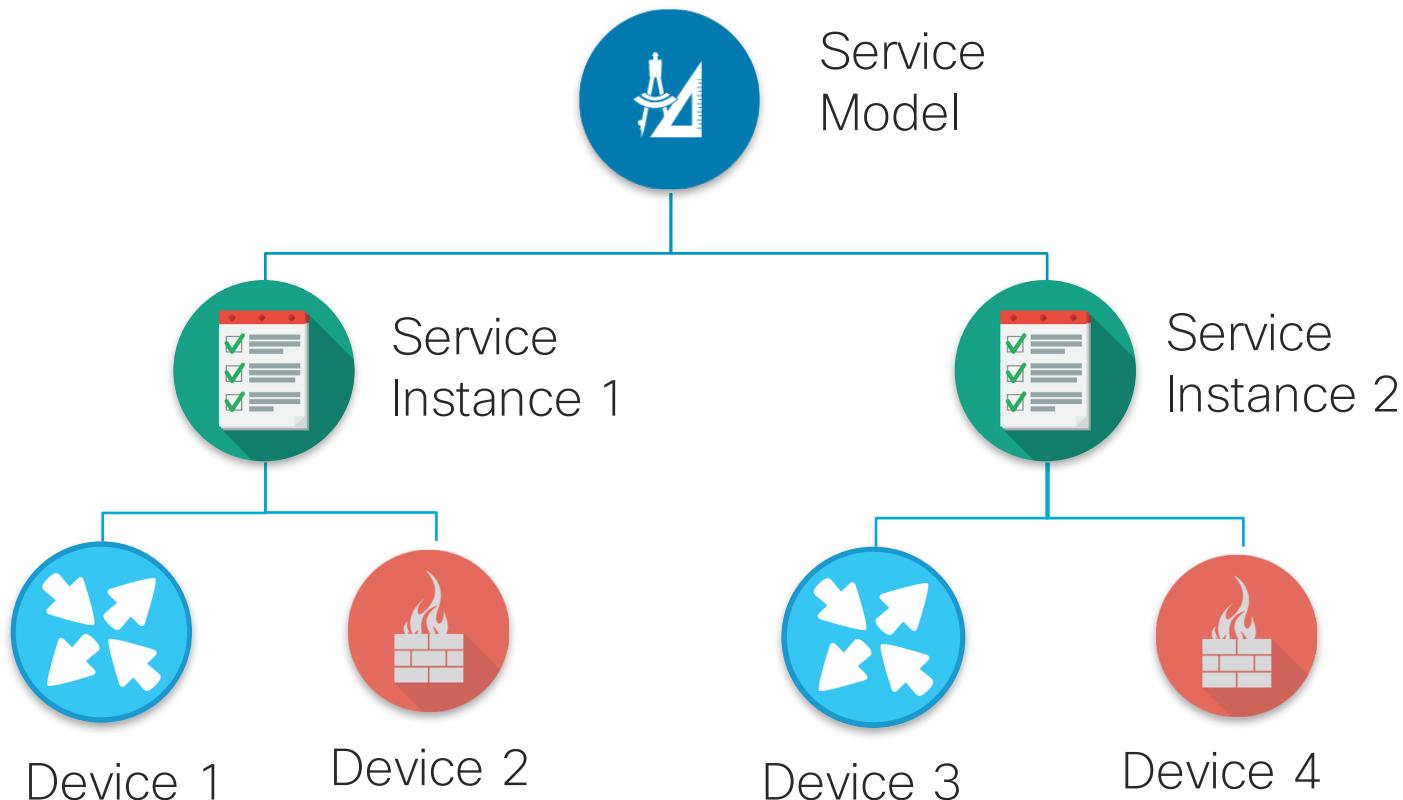


Why is the ‘S’ in NSO so useful? And what does it stand for?



NSO Service Manager & Models

NSO Manages Network Services through the Service Model Construct:



Designed in YANG and modeled with XML templates, code or both

Instantiated in NSO with input parameters
Service's lifecycle is managed by NSO
(Deployment, Compliance and Removal)

NSO Pushes configuration based upon service logic
And input parameters to the devices.

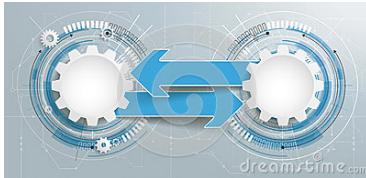
NSO Device Manager

- Devices are added to NSO to be managed
- NSO Syncs the devices running config into the NSO config Database
- Devices can be bundled into groups and managed collectively
- Makes simple large scale changes trivial (example: ipv6 unicast-routing)



Network Element Drivers (NED)

```
<spanning-tree xmlns="urn:ios" >
  <extend>
    <system-id/>
  </extend>
  <mode>
    pvst
  </mode>
</spanning-tree>
```



spanning-tree extend system-id
spanning-tree mode pvst

Configuration Database

- Keeps copy of managed devices configuration
- Maintains Configurations for network services across instances and devices



Northbound APIs



REST API

GET,PUT,POST, DELETE

High Level API into:

NSO Service and Device Manager

Python API

Low Level and High Level APIs into:

NSO cDB

NSO Transaction Engine

NSO Service and Device Managers

Java API

Low Level and High Level APIs into:

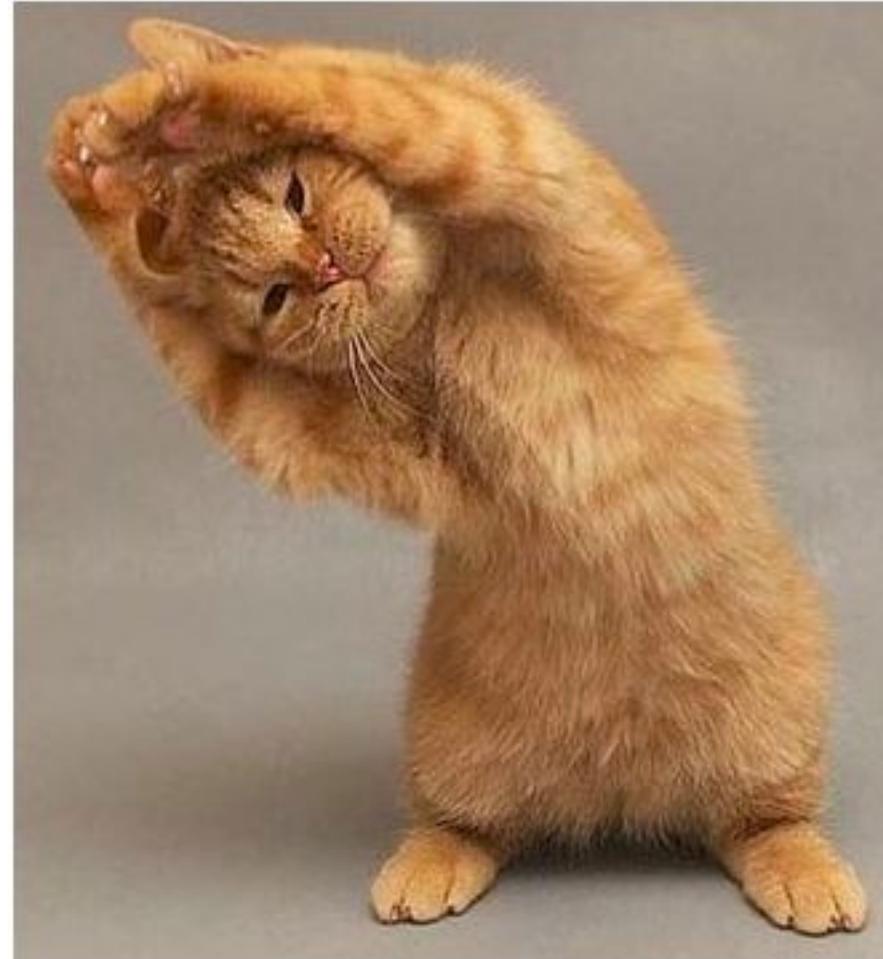
NSO cDB

NSO Transaction Engine

NSO Service and Device Managers

Install Local DevOps VM

Stretch Break!



Lab 0 - Getting Started with Linux and NSO

Building Blocks of NSO Devices

- NSO can connect to any device assuming it has the IP, login credentials, NED (CLI type)
- NSO device names are independent of DNS lookups, the IP is the key piece of info to connect to the device.
- Devices can be in device groups, you can have device groups of device groups
- To make changes to a device NSO needs a local copy of the Config and the device needs to be unlocked. By default a device is locked when added.

Building Blocks of NSO Devices

- Typical flow of adding a device:
 - Add required information for device.
 - Unlock device to be able to pull config (default is admin-state locked)
 - Commit changes to NSO
 - Fetch the ssh keys from the device
 - sync-from the device the config, so NSO has local copy of running-config
 - Admin-state lock the device

Building Blocks of NSO Devices

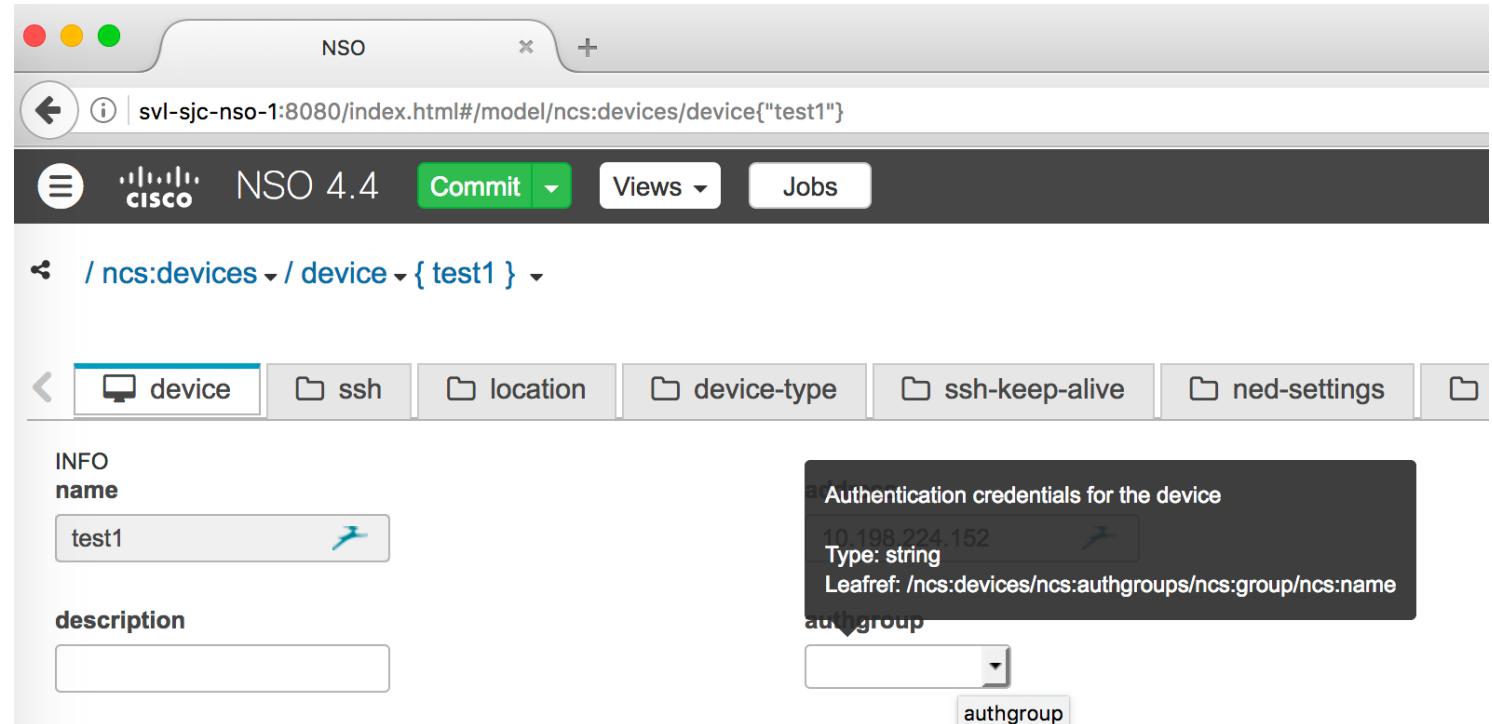
- Typical flow of configuring a device:
 - Enter Config Mode (‘config’)
 - Tell NSO which device or device-group to configure (‘devices device HOSTNAME config’)
 - Use the **NED prefix** (‘ios:’) for commands, **use ‘?’ to verify options**. Some commands have prefix, some don’t depending on how the data is structured in the XML. (‘devices device HOSTNAME config ios:interface GigabitEthernet 1/47’)
 - **Don’t use shorthand commands** like gig 1/47 instead of GigabitEthernet 1/47. Use Tab completion instead.

Building Blocks of NSO CLI

- Everything is within a hierarchy
- Most common commands are under the ‘devices device’ or ‘devices ...’ options.
- Use ‘?’ to explore options
- Use ‘l’ to change output format, debug, or to save output to file. File will be saved to ncs-run directory.
- All changes in config mode need to be committed, all changes can be previewed with commit dry-run

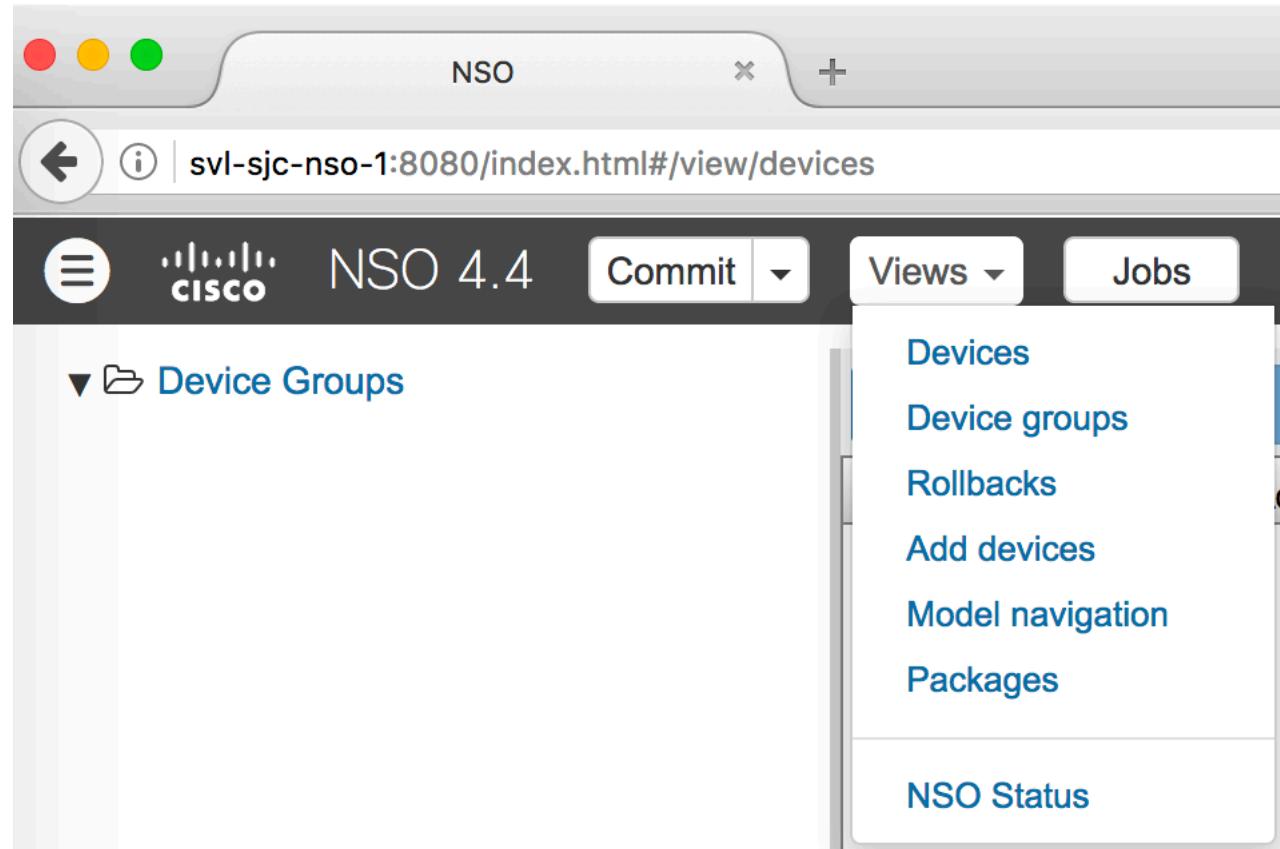
Demo NSO GUI

- Devices View
- Adding a Device
- Commit
- Commit Dry Run
- Alarms
- Rollback
- Xpath Mouse Hover



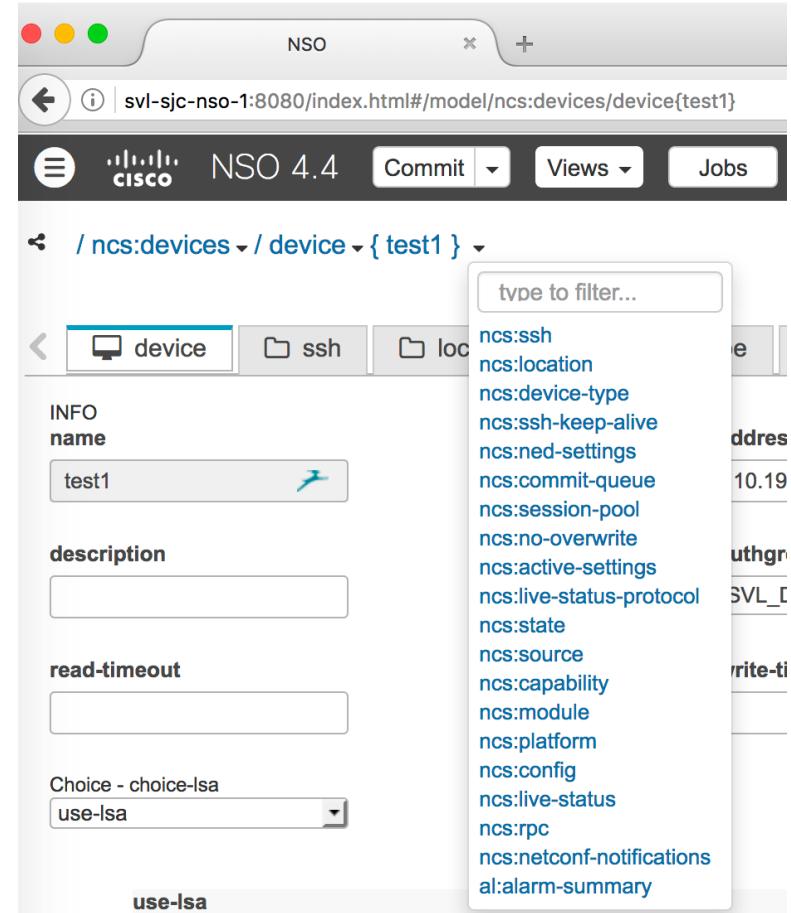
Building Blocks of NSO GUI

- Everything is within a hierarchy
- Views are good entry points to explore.
- Add devices View allows netsim devices to be added



NSO GUI Tips

- Use GUI for visualizing the CLI and understanding data model better
- Use arrows at end of breadcrumbs for more options
- Go to Devices, then click on the device name as a hyperlink to go to details



Lab 1 - Creating Your First Network

BRACE YOURSELF

LUNCH IS COMING



NSO Components: Device Manager

admin@ncs# show devices list					
NAME	ADDRESS	DESCRIPTION	NED ID	ADMIN STATE	
aaaaaaaaaNETSIM	127.0.0.1	-	cisco-ios	unlocked	
aaaaaaaaaa3850LABBY	rtp5-itnlab-3850-sw1.cisco.com	-	cisco-ios	unlocked	
aar02-lab-gw1.cisco.com	10.49.95.153	-	cisco-ios	locked	
aar02-sw1.cisco.com	10.49.95.2	-	cisco-ios	locked	
aar02-wan-gw1.cisco.com	10.49.68.17	-	cisco-ios	locked	

#	Name	Address	Port	Description	Authgroup	Oper State	Admin State
<input type="checkbox"/>	aaaaaaaaaNETSIM	127.0.0.1	10022		default	unknown	unlocked
<input type="checkbox"/>	aaaaaaaaaa3850LABBY	rtp5-itnlab-3850-sw1.cisco.com	22		jabelk.web.auth	unknown	unlocked
<input type="checkbox"/>	aar02-lab-gw1.cisco.com	10.49.95.153	22		jabelk.web.auth	unknown	locked
<input type="checkbox"/>	aar02-sw1.cisco.com	10.49.95.2	22		jabelk.web.auth	unknown	locked
<input type="checkbox"/>	aar02-wan-gw1.cisco.com	10.49.68.17	22		jabelk.web.auth	unknown	locked

NSO Components: Config Management

The screenshot shows the NSO 4.4 interface with the following navigation path: /ncs:devices / device { aar02-lab-gw1.cisco.com } / config / ios:ip / access-list / standard. The page title is "Standard Access List". A table lists six entries under the heading "std-named-acl".

#	
1	ise-snmp-acl
2	lab_nets
3	match-all
4	multicast_ssm_range
5	npc-snmp-acl
6	servicenow-snmp-acl

```
admin@ncs# show running-config devices device aar02-lab-gw1.cisco.com config ios:ip access-list standard | include ios:  
ios:ip access-list standard ise-snmp-acl  
ios:ip access-list standard lab_nets  
ios:ip access-list standard match-all  
ios:ip access-list standard multicast_ssm_range  
ios:ip access-list standard npc-snmp-acl  
ios:ip access-list standard servicenow-snmp-acl  
admin@ncs#
```

NSO Components: Config Management

The screenshot shows the NSO 4.4 interface with the following navigation path: / ncs:devices / device { aar02-lab-gw1.cisco.com } / config / ios:ip / access-list / standard / std-named-acl { lab_nets }.

The interface displays a table of access list rules:

#	rule (k)
<input type="checkbox"/>	permit 10.49.95.153
<input type="checkbox"/>	permit 10.49.96.0 0.0.0.255
<input type="checkbox"/>	permit 10.49.95.164 0.0.0.3

Below the interface, a terminal window shows the running configuration for the device:

```
admin@ncs# show running-config devices device aar02-lab-gw1.cisco.com config ios:ip access-list standard lab_nets
devices device aar02-lab-gw1.cisco.com
config
  ios:ip access-list standard lab_nets
    permit 10.49.95.153
    permit 10.49.96.0 0.0.0.255
    permit 10.49.95.164 0.0.0.3
!
!
!
admin@ncs#
```

Individual or Group device Configuration

```
admin@ncs(config)# devices device aar02-lab-gw1.cisco.com config
admin@ncs(config-config)# ios:ip access-list standard lab_nets
admin@ncs(config-std-nacl)# permit 127.0.0.1
admin@ncs(config-std-nacl)# permit 172.0.0.2
admin@ncs(config-std-nacl)# commit dry-run outformat native
native {
    device {
        name aar02-lab-gw1.cisco.com
        data ip access-list standard lab_nets
            permit 127.0.0.1
            permit 172.0.0.2
        !
    }
}
admin@ncs(config-std-nacl)#
```

Static Templates

```
admin@ncs(config)# devices template "lab_nets template example"
admin@ncs(config-template-lab_nets template example)# config
admin@ncs(config-config)# ios:ip access-list standard std-named-acl lab_nets
admin@ncs(config-std-named-acl-lab_nets)# std-access-list-rule "permit 127.0.0.1"
admin@ncs(config-std-access-list-rule-permit 127.0.0.1)# exit
admin@ncs(config-std-named-acl-lab_nets)# std-access-list-rule "permit 127.0.0.2"
admin@ncs(config-std-access-list-rule-permit 127.0.0.2)# commit dry-run outformat native
native {
}
```

Static template creation

```
admin@ncs(config-std-access-list-rule-permit 127.0.0.2)# commit dry-run
cli {
  local-node {
    data devices {
      + template "lab_nets template example" {
        + config {
          + ios:ip {
            access-list {
              + standard {
                std-named-acl lab_nets {
                  std-access-list-rule "permit 127.0.0.1";
                  std-access-list-rule "permit 127.0.0.2";
                }
              }
            }
          }
        }
      }
    }
  }
}
admin@ncs(config-std-access-list-rule-permit 127.0.0.2)#

```

Static template applied to a device

```
admin@ncs(config)# devices device aar02-lab-gw1.cisco.com apply-template template-name ?
Possible completions:
  lab_nets template example
admin@ncs(config)# devices device aar02-lab-gw1.cisco.com apply-template template-name lab_nets\ template\ example
apply-template-result {
    device aar02-lab-gw1.cisco.com
    result ok
}
admin@ncs(config)# commit dry-run outformat native
native {
    device {
        name aar02-lab-gw1.cisco.com
        data ip access-list standard lab_nets
            permit 127.0.0.1
            permit 127.0.0.2
        !
    }
}
admin@ncs(config)#
```

Template with Variables

Template with variables

```
admin@ncs(config)# devices template template_with_vars
admin@ncs(config-template-template_with_vars)# config
admin@ncs(config-config)# ios:ip access-list standard std-named-acl {$LIST-NAME}
admin@ncs(config-std-named-acl-{$LIST-NAME})# std-access-list-rule {$ACL-RULE}
admin@ncs(config-std-access-list-rule-{$ACL-RULE})# exit
admin@ncs(config-std-named-acl-{$LIST-NAME})# std-access-list-rule {$ACL-RULE-2}
admin@ncs(config-std-access-list-rule-{$ACL-RULE-2})# commit
Commit complete.
admin@ncs(config-std-access-list-rule-{$ACL-RULE-2})# █
```

Original Static template without variables

```
admin@ncs(config)# devices template "lab_nets template example"
admin@ncs(config-template-lab_nets template example)# config
admin@ncs(config-config)# ios:ip access-list standard std-named-acl lab_nets
admin@ncs(config-std-named-acl-lab_nets)# std-access-list-rule "permit 127.0.0.1"
admin@ncs(config-std-access-list-rule-permit 127.0.0.1)# exit
admin@ncs(config-std-named-acl-lab_nets)# std-access-list-rule "permit 127.0.0.2"
admin@ncs(config-std-access-list-rule-permit 127.0.0.2)# commit dry-run outformat native
native {  
}
```

- Variable using syntax {\$somename} (upper or lower case)
- In this case list-name replaces lab_nets
- Acl-rule replaces “permit 127.0.0.1”
- Acl-rule-2 replaces “permit 127.0.0.2”
- Note that NSO represents the ACL syntax as an entire string unit (includes permit + rule within the variable)

Template with variables application

```
admin@ncs(config-device-aar02-lab-gw1.cisco.com)# apply-template template-name template_with_var
variable { name LIST-NAME value 'lab_nets' } variable { name ACL-RULE value 'permit\ 127.0.0.1'
variable { name ACL-RULE-2 value 'permit\ 127.0.0.2' }
apply-template-result {
    device aar02-lab-gw1.cisco.com
    result ok
}
admin@ncs(config-device-aar02-lab-gw1.cisco.com)# commit dry-run outformat native
native {
    device {
        name aar02-lab-gw1.cisco.com
        data ip access-list standard lab_nets
            permit 127.0.0.1
            permit 127.0.0.2
        !
    }
}
admin@ncs(config-device-aar02-lab-gw1.cisco.com)#[
```

- Note that the variables are inclosed in single quotes and words with spaces need an escape character '\'

Check Sync and out of band changes

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com check-sync  
result in-sync
```

Now I logged in a separate ssh session and made a change (without NSO knowing it)

```
rtp5-itnlab-dt-sw1(config-if)#int gigabitEthernet 1/5  
rtp5-itnlab-dt-sw1(config-if)#desc  
rtp5-itnlab-dt-sw1(config-if)#description automation_test  
rtp5-itnlab-dt-sw1(config-if)#end  
rtp5-itnlab-dt-sw1#wr  
Building configuration...  
Compressed configuration from 138460 bytes to 37538 bytes[OK]  
rtp5-itnlab-dt-sw1#
```

Check-Sync/Compare-Config with difference

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com check-sync
result out-of-sync
info got: 9effb7a4329ebeb30f95cb91f46d7026 expected: 710cd4880d4214b48de34f937ff5ec02

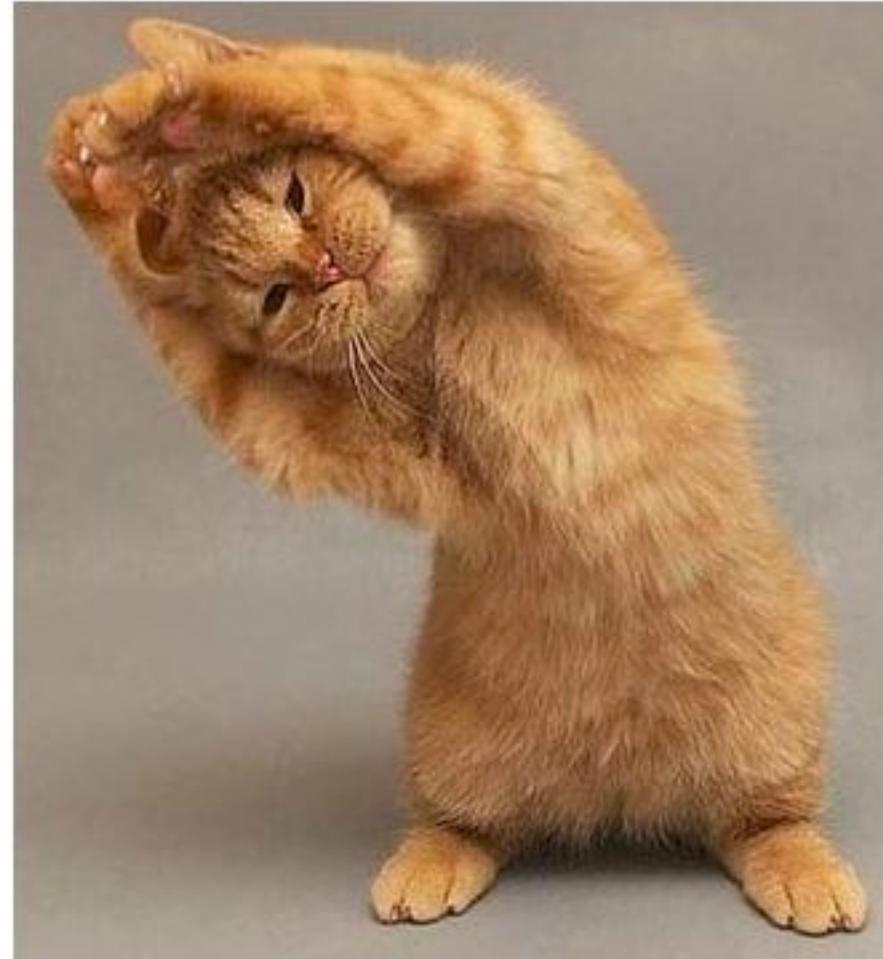
admin@ncs# *** ALARM out-of-sync: got: 9effb7a4329ebeb30f95cb91f46d7026 expected: 710cd4880d4214b48de34f937ff5ec02

admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com compare-config
diff
devices {
    device rtp5-itnlab-dt-sw1.cisco.com {
        config {
            ios:interface {
                GigabitEthernet 1/5 {
                    -           description "applying demosite acl";
                    +           description automation_test;
                }
            }
        }
    }
}
```

Use Sync-from to make local NSO DB in sync

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com sync-from  
result true  
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com check-sync  
result in-sync  
admin@ncs#
```

Stretch Break!



Device Groups, Templates and Rollback

- A device can be added to one or many device group(s), a device group can be nested into another device group
- Templates can either have static configs enforced every time, or variables put in to be declared at the time of usage. Service templates will be covered later.
- You can rollback any change, or part of a change, in NSO through the Rollback in GUI or on CLI (config -> rollback)

Lab 2 - Configuring Device IOS Config & Templates & Device Groups

Show Commands

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com live-status ios-stats:exec ?
Possible completions:
any          Execute any command on device
clear        Reset functions
copy         Copy from one file to another
license      Smart licensing Commands
ping         Send echo messages
reload       Halt and perform a cold restart
show         Execute show commands
traceroute   Trace route to destination
verify       Verify a file
```

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com live-status ios-stats:exec any ?
Possible completions:
WORD  any "<cmd> [option(s)]", e.g: any "show ppp summary"
|     Output modifiers
<CR>
```

```
admin@ncs# devices device rtp5-itnlab-dt-sw1.cisco.com live-status ios-stats:exec any "show ip int br"
result
> show ip int br
Interface          IP-Address      OK? Method Status      Protocol
FastEthernet1      unassigned    YES unset  down       down
GigabitEthernet1/1 unassigned    YES unset  down       down
GigabitEthernet1/2 unassigned    YES unset  down       down
GigabitEthernet1/3 unassigned    YES unset  down       down
```

Lab

Open Discussion –
Feedback, Demos, Deep
Dives?

Feedback - Survey



WELCOME TO DAY TWO!

What is a Service in your PIN? (not one we have used as an example)



What does NCS Stand For?



Agenda

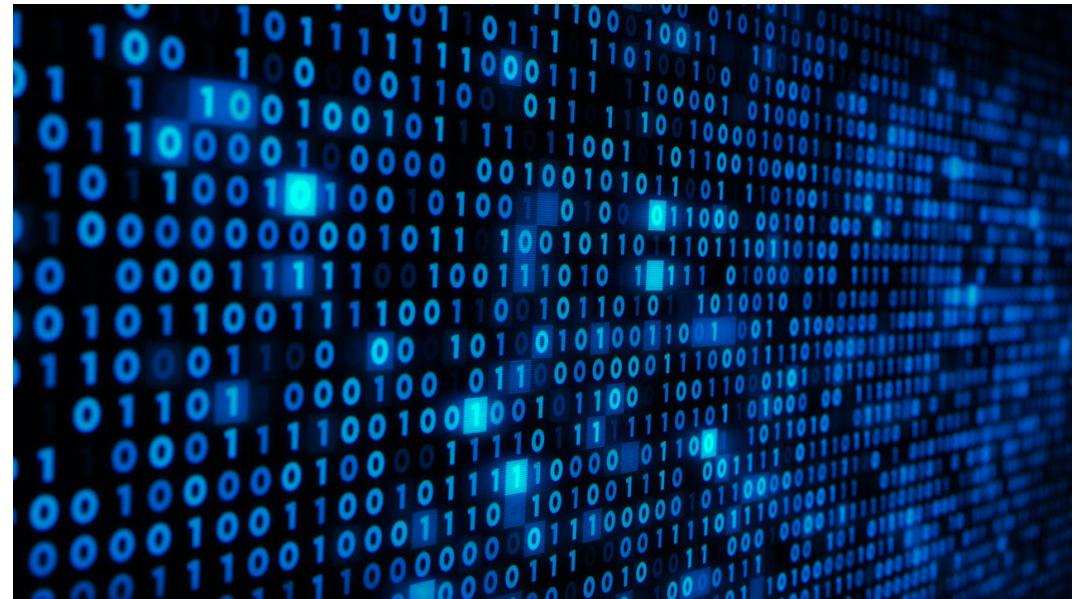
- Programmatic Interfaces
- YANG
- XML
- REST

Interfaces that are good for humans...



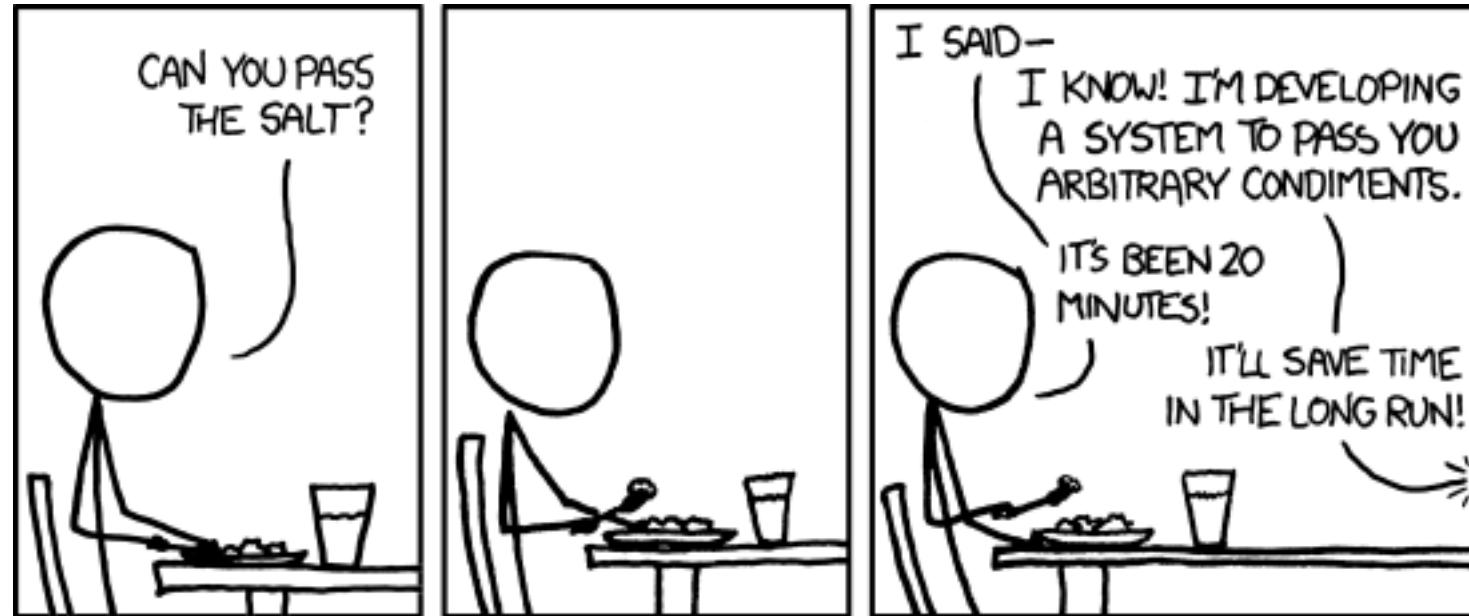
Aren't very good for machines

Interfaces that are good for machines



Aren't very good for machines

But it's easier to make machine interfaces friendlier to people



Than people interfaces friendlier to machines

NSO CLI is a human-oriented interface

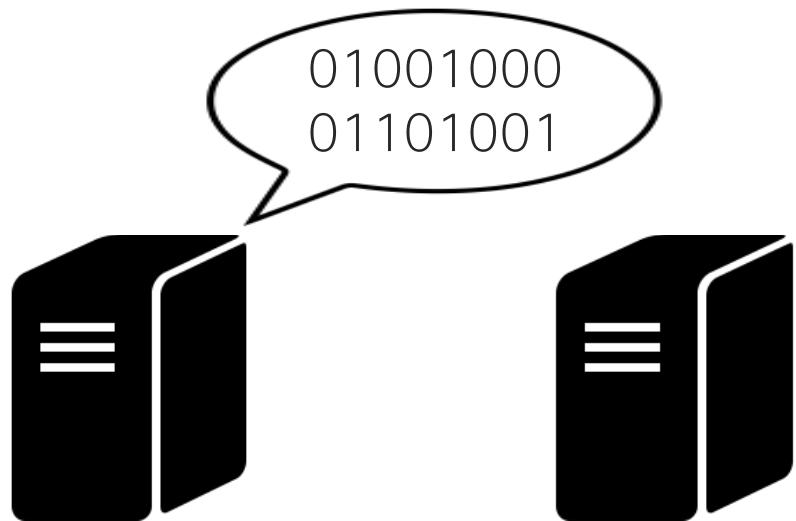


```
[patrhuyn@ncs# devices device sjc12-31-sw1.cisco.com ?  
Possible completions:  
  check-sync          Check if the NCS config is in sync with the device  
  check-yang-modules  Check if NCS and the device have compatible YANG  
                      modules  
  compare-config      Compare the actual device config with the NCS copy  
  config              NCS copy of the device configuration  
  connect             Connect to the device  
  delete-config       Delete the config in NCS without deleting it in the  
                      device  
  disconnect          Close all sessions to the device  
  live-status          Status data fetched from the device  
  live-status-protocol Additional protocols for the live-tree (read-only)  
  netconf-notifications NETCONF notifications from the device  
  ping                ICMP ping the device  
  pioneer             Secure copy file to the device  
  scp-from            Secure copy file to the device  
  scp-to              SSH connection configuration  
  ssh                Synchronize the config by pulling from the device  
  sync-from           Synchronize the config by pushing to the device  
  sync-to             patrhuyn@ncs# devices device sjc12-31-sw1.cisco.com ]
```



A human-oriented interface provides a way for a person to interact and communicate with a machine.

Machine-Oriented Interfaces



- A machine-oriented interface provides a way for one machine to interact and communicate with another machine.
- We obtain maximum value from our automation when they use machine-oriented interfaces.
- But what is the fundamental difference between human-oriented and machine-oriented interfaces?

Example - HTML

Unstructured vs Structured Data

Wayne Rooney Forward 31 David de
Gea Goalkeeper 26

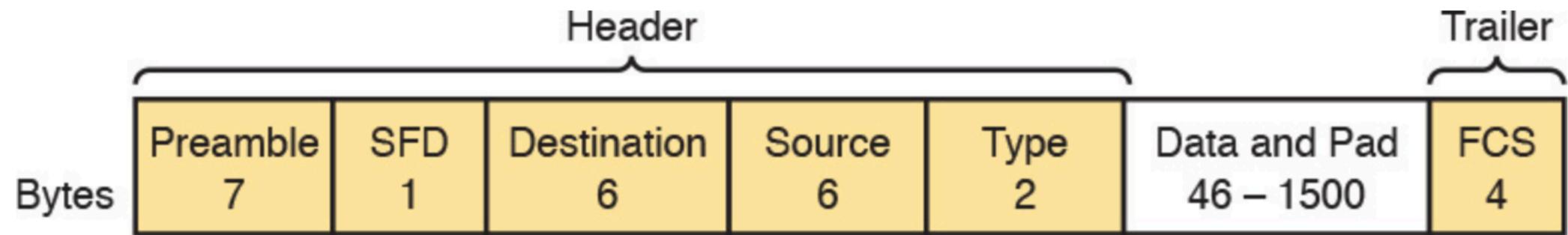
Is there a hierarchy?

What are the possible fields?

Where does it begin and where does
it end?

```
<Player>
  <Name>Wayne Rooney</Name>
  <Position>Forward</Position>
  <Age>31</Age>
</Player>
<Player>
  <Name>David de Gea</Name>
  <Position>Goalkeeper</Position>
  <Age>26</Age>
</Player>
```

Unstructured vs Structured Data cont.



Ethernet is designed with a machine-oriented interface in mind

Problem with Cisco CLI commands

```
Cts sxp connection peer 10.10.10.10 password default mode local  
listener hold-time 3600 36000
```

- What fields correspond to what? What fields are allowed? How do I compare fields?
- When making network automation, the burden of knowing structure and syntax of the commands is placed on the developer.

Current network automation is this:



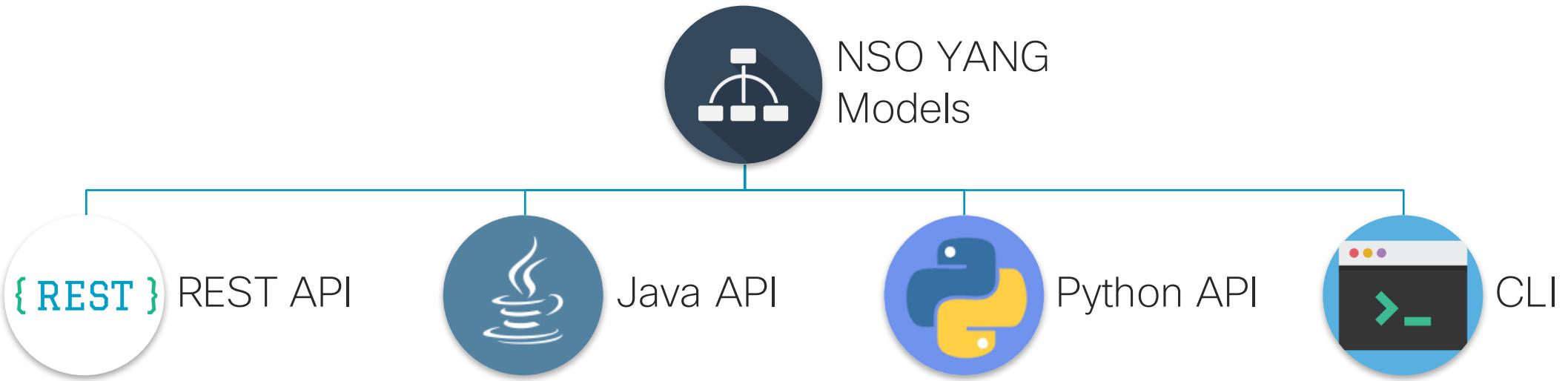
Making a machine use a human-oriented interface.

So how do we interact
with Cisco CLI in a machine
oriented way?

First, we need a way to
translate CLI into a
structured Data Form...

Introduction to Yang

Everything in NSO is YANG



All interfaces are defined by the NSO Yang Model

Example – Looking at NSO YANG Files

So what is Yang?

It is not the opposite of
Yin

Stands for:

Yet
Another
Next
Generation
(Data Modeling Language)

YANG is a data
modeling language
used to model
configuration and state
data manipulated by
the Network
Configuration Protocol
(NETCONF)

That's nice...
But what *IS* it
really?

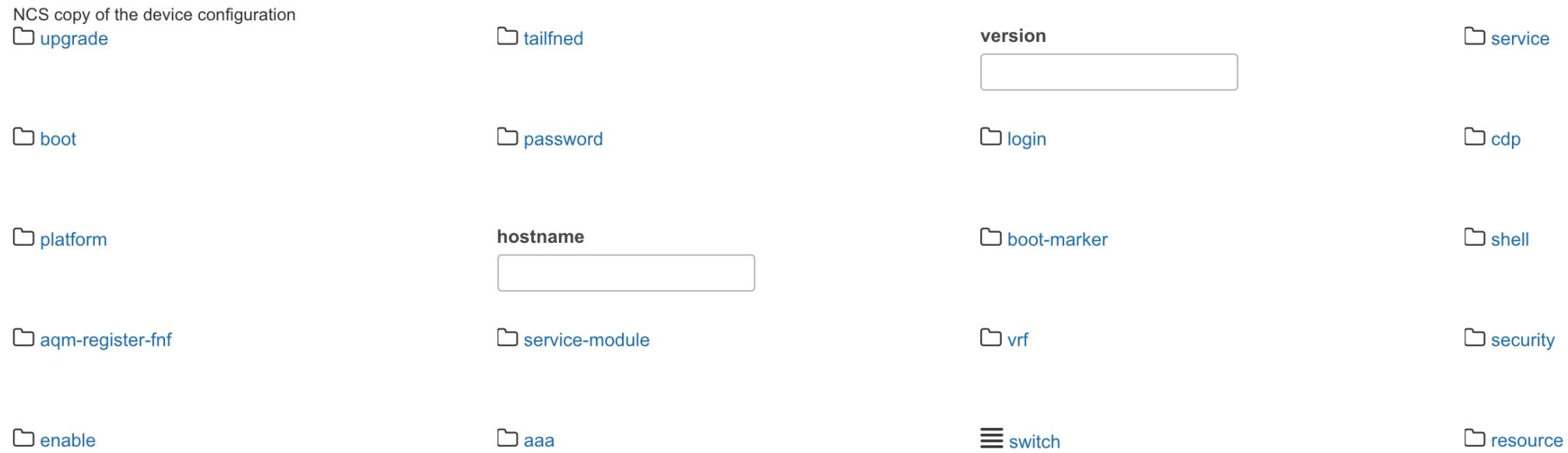
Yang allows for abstractions to be
defined within the Yang language and
then mapped into a markup language.
In our case, XML

Yang types and a way to think about them

- Container
 - Groups things together
- List
 - A collection of leafs
- Leaf
 - A end node of data
- Leaf-List
 - A list of single items



The NSO UI uses this idea as well



*This is a UI representation of the Cisco-IOS Config Yang Model

Modeling a Football team in Yang

- Team should have a name.
- Has multiple players.
- Players have names.
- They have specific positions.
- They have an age.

Yang

```
Container FootballTeam {  
    Leaf TeamName {type string;}  
    List Player  
        Leaf PlayerName {type string;}  
        Leaf Position {type string;}  
        Leaf Age {type uint8;}  
}
```

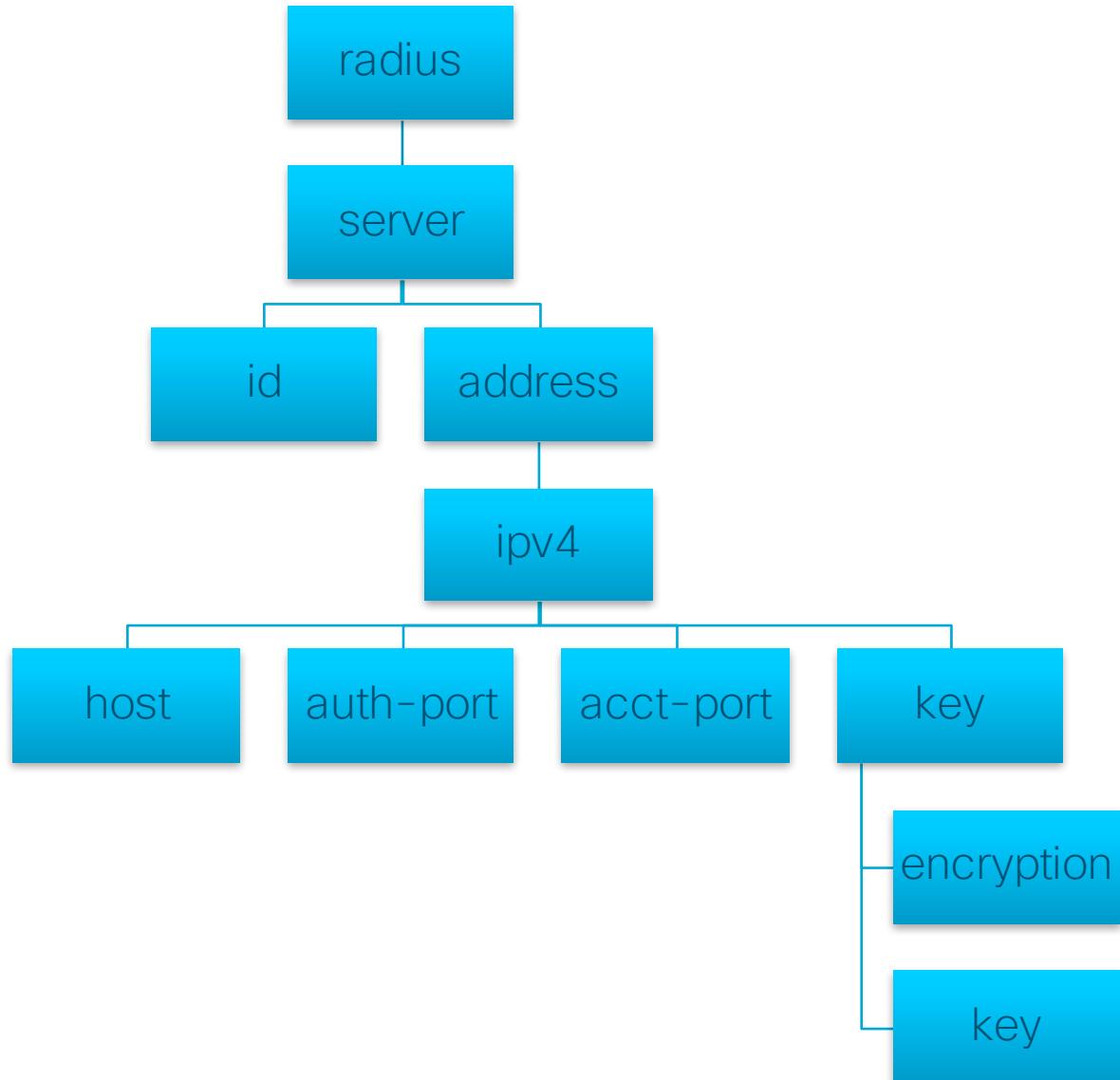
Modeling Cisco IOS commands in Yang

```
radius server <AAA Server>
```

```
  address ipv4 <IP Address> auth-port 1812 acct-port 1813
```

```
  key 7 <Encrypted Key>
```

```
container radius {  
    list server {  
        leaf id {type string;}  
        container address {  
            container ipv4 {  
                leaf host {type string;}  
                leaf auth-port {type uint16;}  
                leaf acct-port {type uint16;}  
                container key {  
                    leaf encryption {type enumeration;}  
                    leaf key {type string;} } } } } }
```



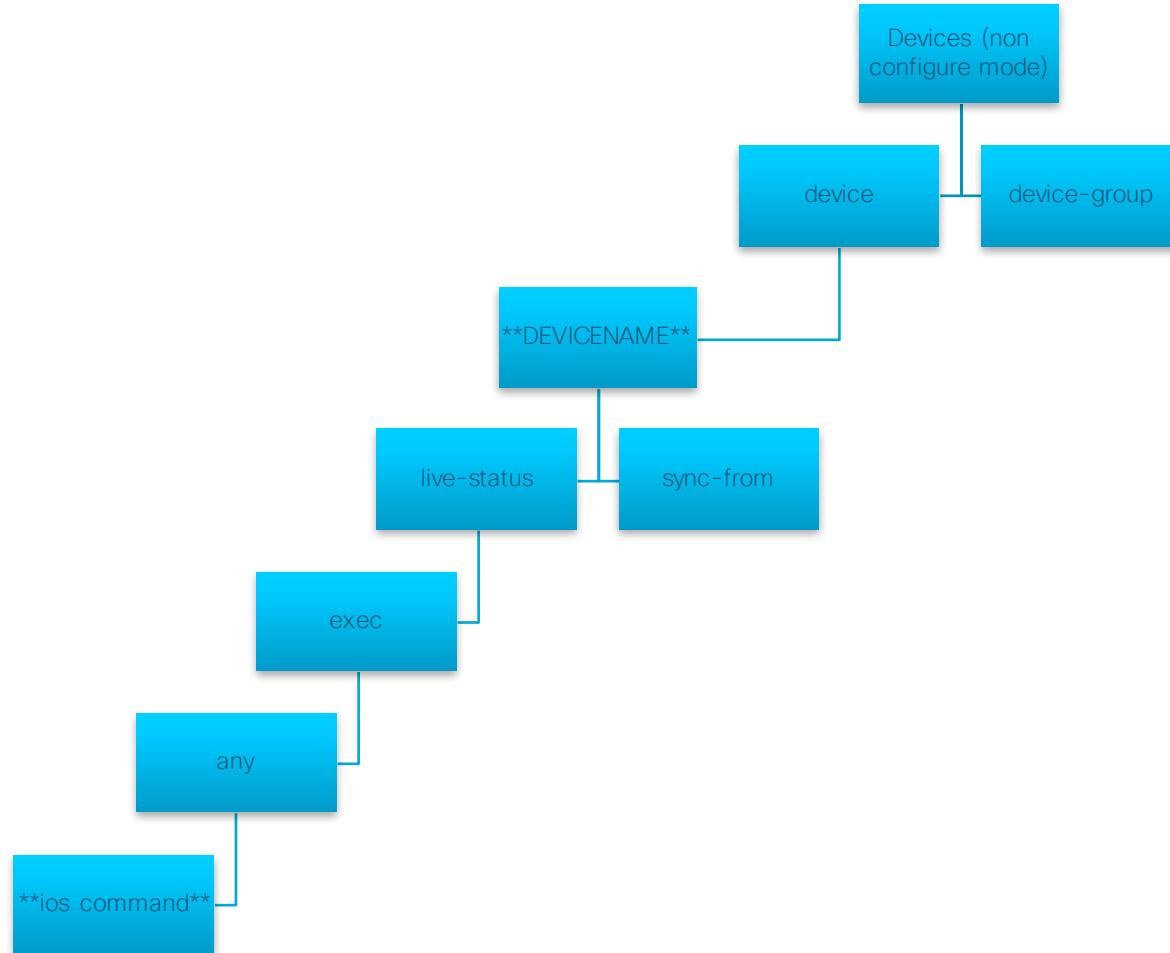
Examples of YANG

```
container access {
    description "Set access mode characteristics of the interface";
    leaf vlan {
        description "VLAN ID of the VLAN when this port is in access mode";
        type uint16 {
            range "1..4094";
        }
    }
}
container voice {
    description "Voice appliance attributes";
    leaf vlan {
        description "Vlan for voice traffic";
        type uint16 {
            range "1..4094";
        }
    }
}
```

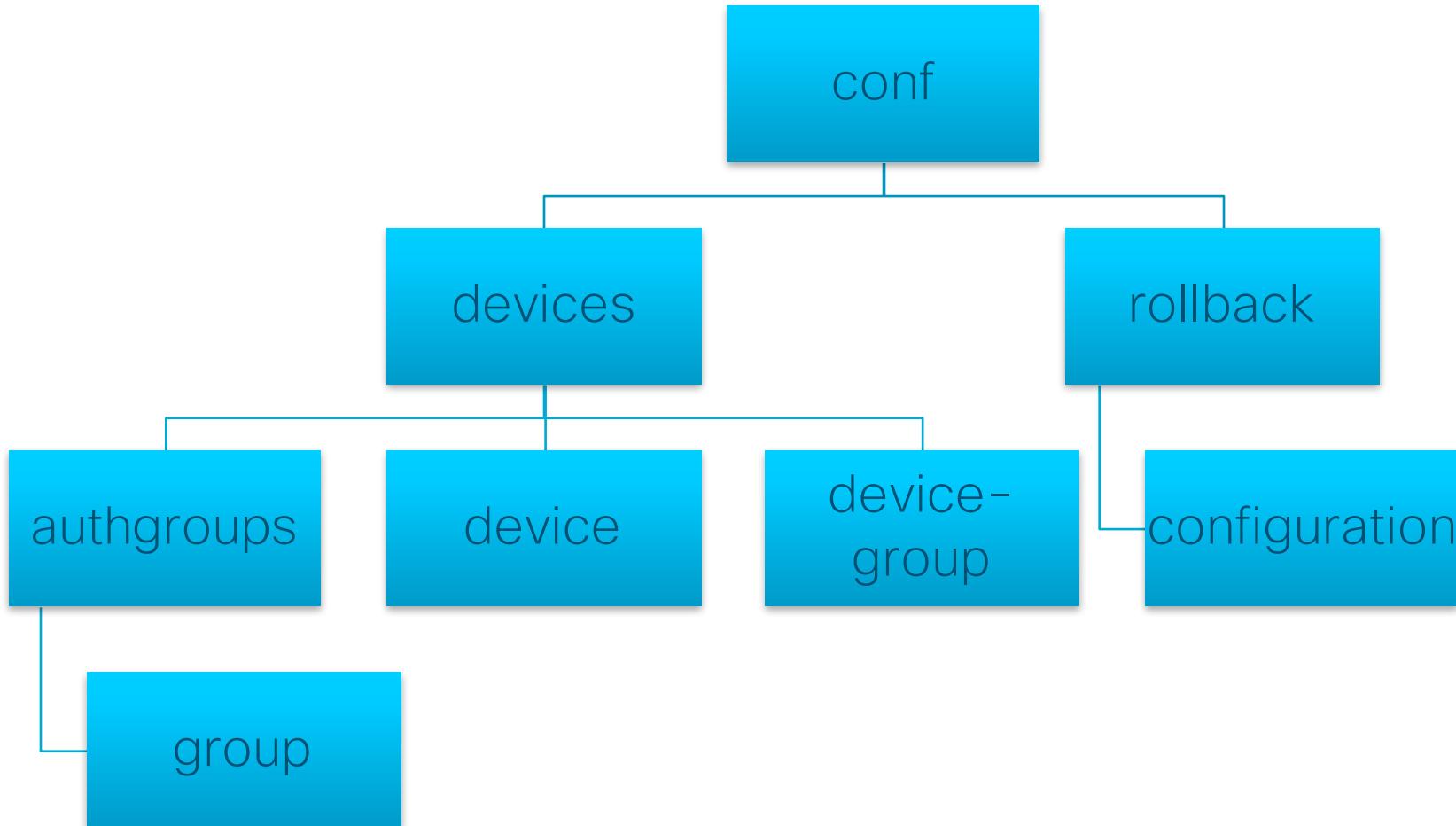
```
//ip access-list resequence
container resequence {
    description
        "Resequence Access List";
    leaf numbers {
        type union {
            type ios-types:std-acl-type;
            type ios-types:ext-acl-type;
        }
    }
    leaf start-seq-no {
        type uint64 {
            range "1..2147483647";
        }
    }
    leaf step-seq-no {
        type uint64 {
            range "1..2147483647";
        }
    }
}
```

Hierarchical Data

NSO data tree (non configure mode)



NSO data tree (configure mode)

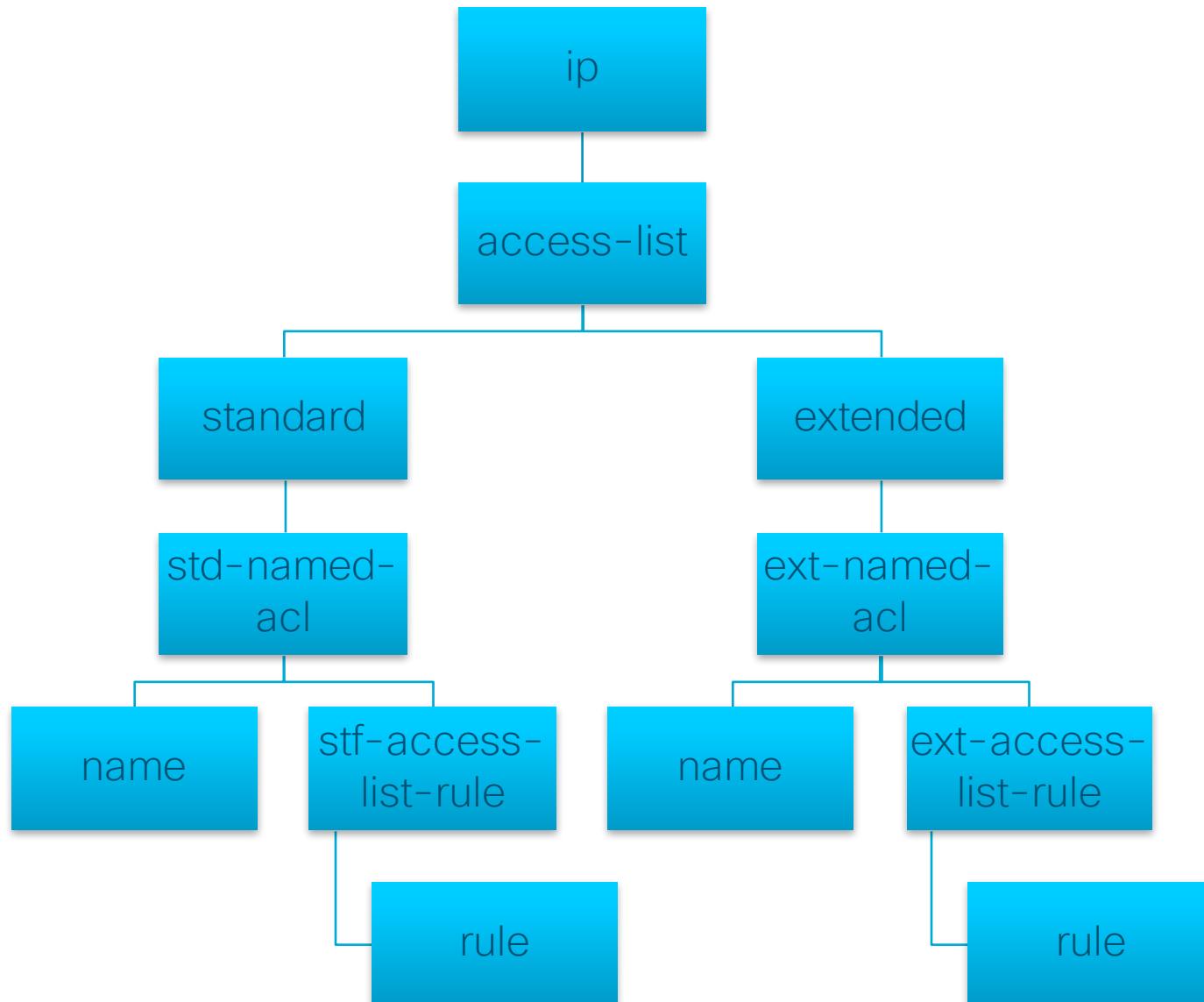


Lab - Navigating Data

Modeling Cisco IOS commands in Yang again

```
ip access-list standard <NAME>
  permit <IP address 1>
  permit <IP address 2>
ip access-list extended <Name>
  permit <rule>
  deny <rule>
```

```
Container ip
  container access-list {
    container standard
      list std-named-acl {
        leaf name {type std-acl-type;}
      }
      list std-access-list-rule {
        leaf rule {type string;}}}
    container extended {
      list ext-named-acl {
        leaf name {type string;}
      }
      list ext-access-list-rule {
        leaf rule {type string;}}}
```



So far we have been
navigating the model.

But how is it built?

Demo – Structure of YANG module

Structure of a YANG Model

- YANG files are called modules
- Have a namespace and prefix
- Can import other YANG models
- Allows for meta data about the model
 - Description
 - Revision
 - Author

```
module ubvpn {  
  
    namespace "http://example.com/ubvpn";  
    prefix ubvpn;  
  
    import ietf-inet-types {  
        prefix inet;  
    }  
    import tailf-common {  
        prefix tailf;  
    }  
    import tailf-ncs {  
        prefix ncs;  
    }  
  
    description  
        "This YANG Module defines service parameters and meta data for the UBVPN Service."  
  
    revision 2017-03-17 {  
        description  
            "  
                First Edition. Beginning of development.  
                Requirements to be flushed out and documented."  
    }  
}
```

YANG Types

Node Structure

- A node in a YANG Model follows a structure
- First, we define the node type and name

```
leaf mask {  
    type inet:ipv4-address;  
    tailf:info "A.B.C.D;;OSPF wild card bits";  
}
```

Node Structure

- A node in a YANG Model follows a structure
- Second, we define data type of the node
- Standard types are:
 - String
 - Uint64
 - Enumeration
- NSO enables custom types

```
leaf mask {  
    type inet:ipv4-address;  
    tailf:info "A.B.C.D;;OSPF wild card bits";  
}
```

Node Structure

- A node in a YANG Model follows a structure
- Last, we define node modifiers
- These are extra pieces of info that tells NSO about the node or other things to do
- Common are:
 - Info
 - description
 - hidden

```
leaf mask {  
    type inet:ipv4-address;  
    tailf:info "A.B.C.D;;OSPF wild card bits";  
}
```

Leaf

- A single thing with no sub-attributes
- You can specify the type of the Leaf
 - string
 - uint16
 - enumeration
 - custom types like “inet:ipv4-address”
- Add descriptions for NSO use
 - Use tailf:info

```
leaf mask {  
    type inet:ipv4-address;  
    tailf:info "A.B.C.D;;OSPF wild card bits";  
}
```

Leaf-List

- A list of simple things
 - Each simple thing has the same type
 - Example:
 - switchport trunk allowed vlan 10,30,240,330,430,550,555,1079
- Different from List
 - “List” is a list of complex things

```
leaf-list vlans {  
    type uint16 {  
        tailf:info "WORD;;VLAN IDs of the allowed VLANs when this port is in trunking mode";  
    }  
}
```

Container

- A single complex thing
 - Has multiple attributes
 - Can have leaves, leaf-lists, lists, or more containers

```
container ip {  
    list vrf {...}  
    container mcr-conf {...}  
    container access-list {...}  
    ...  
}
```

List

- Multiple same, complex things
 - Each have multiple attributes
- Uses keys to determine the correct element
- Different from leaf-list
 - Leaf-lists are all simple things
 - Lists are much more common than leaf-lists

```
// aaa authentication login *
list login {
    key name;
    leaf name {type string;}
    leaf group;
    leaf local;
    leaf none;
}
```

YANG Types

- Once we define a node and its name we need to define its type
- YANG enables us to do this with the type statement
- Common types:
 - string
 - uint64
 - enumeration
 - boolean
 - leafref

```
leaf Hub {  
    tailf:info "The UBVPN Hub the site is in. To be mapped to ASA for configuration";  
    mandatory true;  
    type enumeration{  
        enum "SJC";  
        enum "BGL";  
        enum "TYO";  
        enum "AER";  
    }  
}
```

YANG Type Modifiers

- When we define a leaf in NSO, we can control the data allowed
- YANG Type Modifiers enable us to create coarse or finely defined models
- Enables us to limit and control what and how data is entered into the model

```
leaf lab_id {  
    tailf:info "Lab ID";  
    mandatory true;  
    type uint32 {  
        range "1000..99999";  
    }  
}
```

YANG Type Modifiers

- When we define a leaf in NSO, we can control the data allowed
- Node types have specific modifiers
- Lists
 - Key, limit
- Container
 - Presence
- Leaf
 - Mandatory, pattern, range

Building Our Own Types

- NSO enables us to define our Own Types for re-use
- We do this through the `typedef` statement
- Can be as simple or complex as we want

```
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}]+)?';
    }
}
```

Exercise as a team

- As a small team (3-4 People) create your own type
- Create a type to represent one of the following:
 - PIN in the Network
 - Common Interface Type (FastEthernet, GigabitEthernet, etc..)
 - Lab-ID
- Needs:
 - Node statement
 - Type statement
 - Type Modifiers

Groupings

- NSO enables use to build re-usable models
- Groupings are groupings of node types that can be referenced
- Referenced through “uses group-name”

```
uses ncs:service-data;
ncs:servicepoint ubvpn-servicepoint;
```

```
grouping interface-name-grouping {
    choice interface-choice {

        leaf Null {
            tailf:info "Null interface";
            tailf:cli-allow-join-with-value {
                tailf:cli-display-joined;
            }
            tailf:non-strict-leafref {
                path "/ios:interface/Null/name";
            }
            type uint8 {
                tailf:info "<0-0>;Null interface number";
                range "0";
            }
        }
    }
}
```

Exercise as a team

- As a small team (3-4 People) create your own grouping
- Create a grouping to represent one of the following:
 - ACL
 - Interface (type, number, description)
 - Lab-ID
- Needs:
 - Grouping statement
 - Node statements
 - Type statements

Lunch

Lab – YANG Modeling

What does YANG Stand For?



What does XML Stand For?



Demo –
Show Running Config in XML

XML

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Tove from Jani, stored as XML:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body.

Visualize the Data

Someone must write a piece of software to send, receive, store, or display it:

Note

To: Tove

From: Jani

Reminder

Don't forget me this weekend!

How Can XML be Used?

XML Separates Data from Presentation

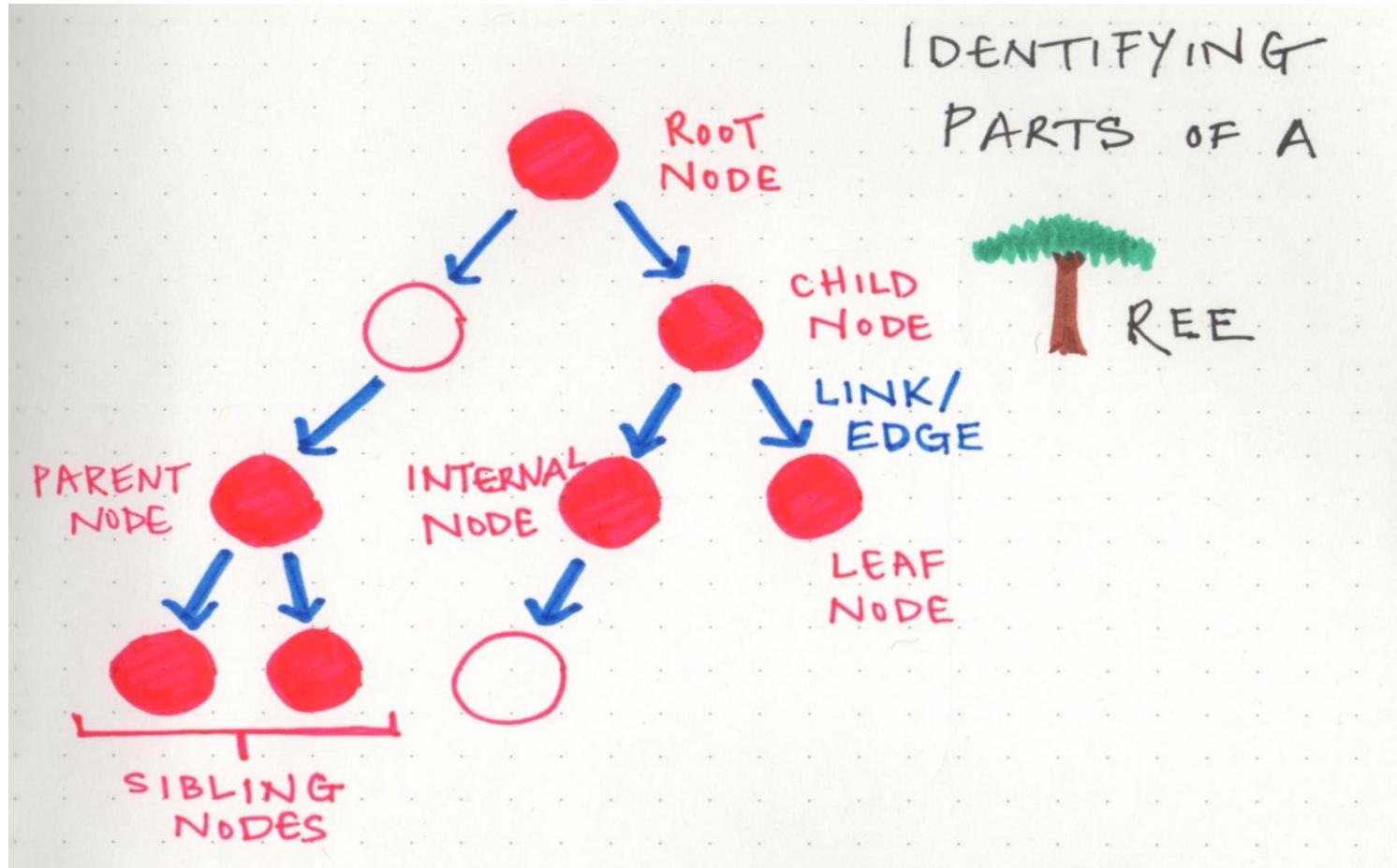
XML does not carry any information about how to be displayed. The same XML data can be used in many different presentation scenarios.

Because of this, with XML, there is a full separation between data and presentation.

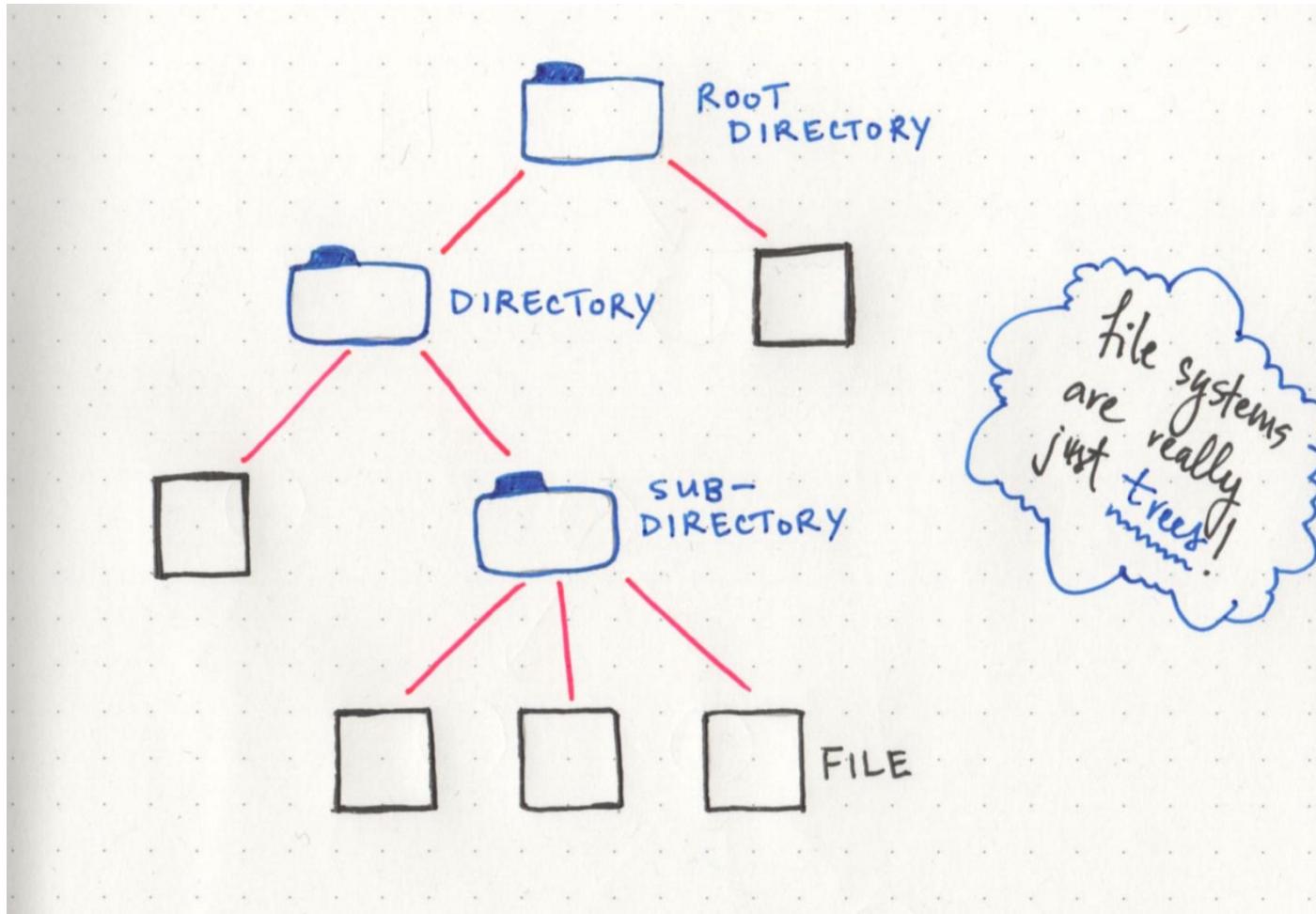
XML Tree Structure



XML Tree Structure



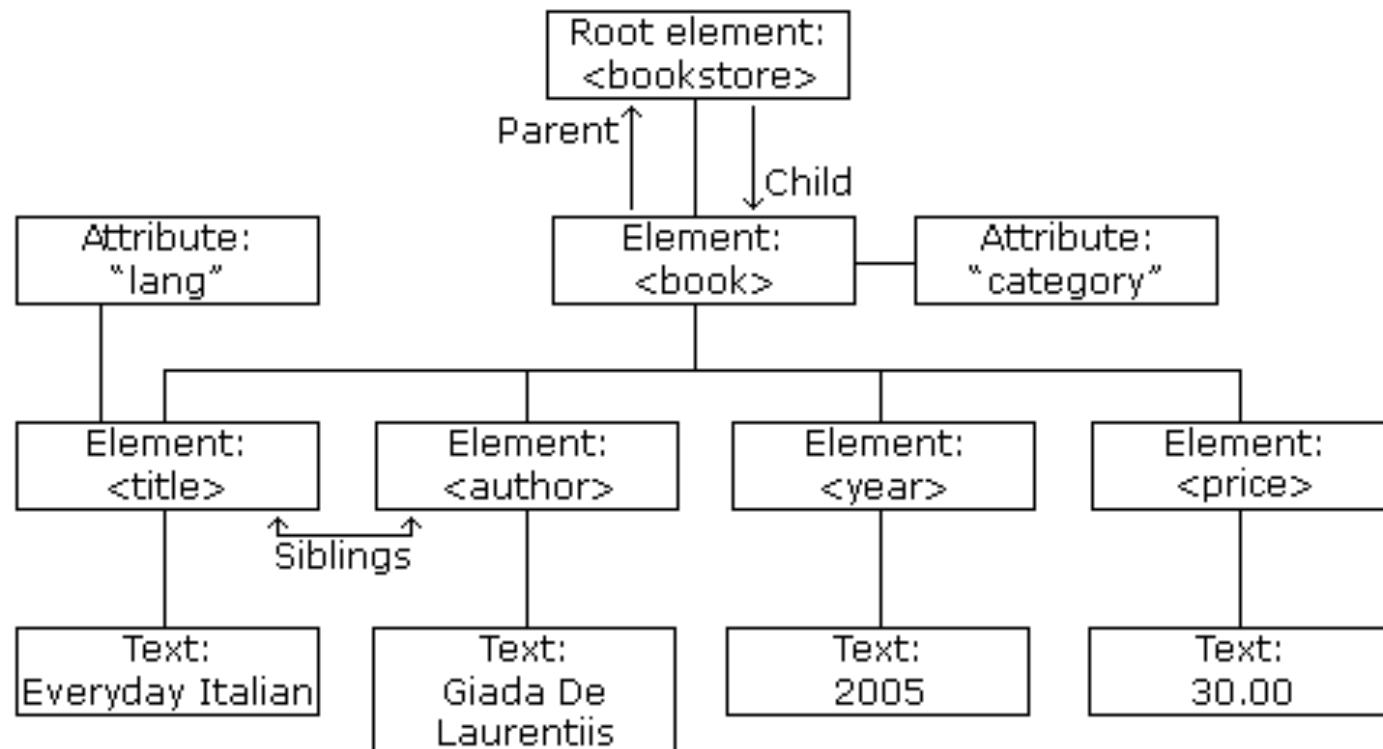
XML Tree Structure - Example



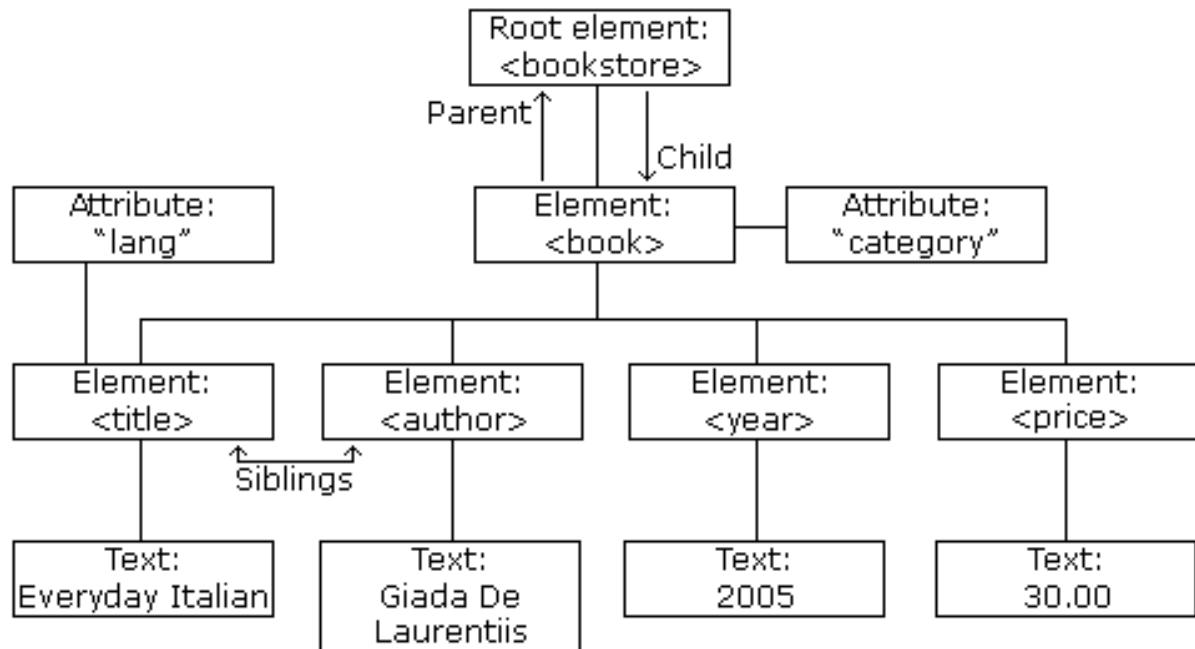
file systems
are really
just trees!

XML Tree Structure

XML documents form a tree structure that starts at "the root" and branches to "the leaves".



An Example XML Document



```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Quiz: Write out an xml tree using tags for a cricket team



Cisco IOS in XML

Cutsheet

radius server <AAA Server>

 address ipv4 <IP Address> auth-port <Port #> acct-port <Port #>

 key <Level> <Encrypted Key>

Actual Configuration

radius server primary-trustsec-radius

 address ipv4 173.36.131.232 auth-port 1812 acct-port 1813

 key 7 1107160A1F021218

```
<radius xmlns="urn:ios">  
    <server>  
        <id>primary-trustsec-radius</id>  
        <address>  
            <ipv4>  
                <acct-port>1813</acct-port>  
                <auth-port>1812</auth-port>  
                <host>173.38.200.166</host>  
            </ipv4>  
        </address>  
        <key>  
            <type>7</type>  
            <secret>1107160A1F021218</secret>  
        </key>  
    </server>  
</radius>
```

XML Tree Structure

XML documents are formed as **element trees**.

An XML tree starts at a **root element** and branches from the root to **child elements**.

All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms **parent**, **child**, and **sibling** are used to describe the relationships between elements. Parent have children. Children have parents. Siblings are children on the same level (brothers and sisters).

All elements can have text content (Harry Potter) and attributes (category="cooking").

XML Syntax Rules

XML Documents Must Have a Root Element

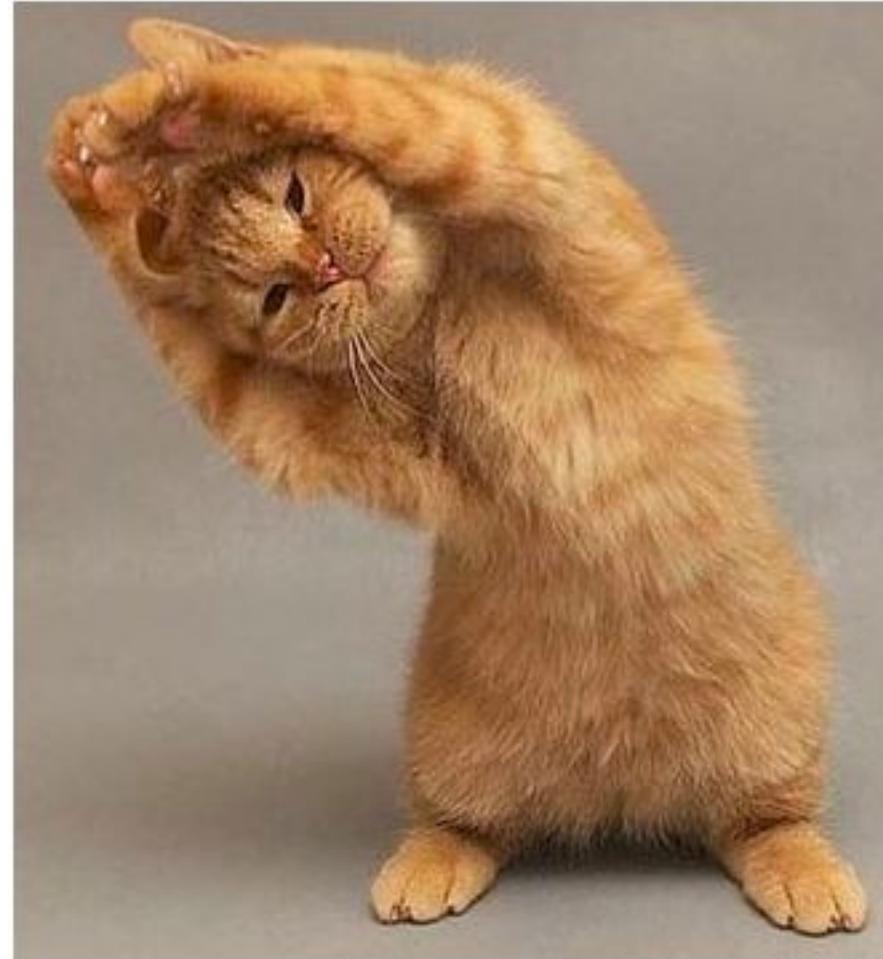
XML documents must contain one **root** element that is the **parent** of all other elements:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

In this example **<note>** is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Stretch Break!



XML Elements (actual data)

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

```
<element></element>
```

You can also use a so called self-closing tag:

```
<element />
```

XML Attributes (meta-data)

Attributes are designed to contain data related to a specific element. (meta-data)

Attribute values must always be quoted. Either single or double quotes can be used.

For example, a person's gender, the <person> element can be written like this:

```
<person gender="female">
```

Some things to consider when using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99).

<bookstore> and <book> have **element contents**, because they contain elements.

<book> has an **attribute** (category="children").

XML Namespaces (using prefixes)

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

Name conflicts in XML can easily be avoided using a name prefix.

When using prefixes in XML, a **namespace** for the prefix must be defined. The namespace can be defined by an **xmlns** attribute in the start tag of an element. The namespace declaration has the following syntax. `xmlns:prefix="URI"`.

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN).

NSO Namespaces

XML Template

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>/Devices</name>
      <config>
        <ip xmlns="http://cisco.com/ned/asa">
          <local>
            <pool>
              <id>{$Partner_site_code}-{$Country_Code}-pool</id>
              <address>{$Address_pool_start}-{$Address_pool_end}</address>
              <mask>{$Address_pool_mask}</mask>
            </pool>
          </local>
        </ip>
      </config>
    </device>
  </devices>
</config-template>
```

CLI Template

```
svl-gm-joe-asa-fw1
  ip local pool site-partner-pool 10.0.0.0-10.0.0.255 mask 255.255.255.0
  group-policy site-partner internal
  group-policy site-partner attributes
  address-pools value site-partner-pool
  vpn-simultaneous-logins 1
exit
```

QoS Example

```
<policy-map xmlns="urn:ios">
  <name>classify</name>
  <description>QoS 2.3.2-</description>
  <class>
    <name>qos-scavenger</name>
    <set>
      <dscp>
        <value>8</value>
      </dscp>
    </set>
  </class>
  <class>
    <name>qos-medium-priority</name>
    <set>
      <dscp>
        <value>16</value>
      </dscp>
    </set>
  </class>
```

```
<interface xmlns="urn:ios">
  <GigabitEthernet>
    <name>1/0/1</name>
    <service-policy>
      <input>TRUST-MARKING</input>
    </service-policy>
  </GigabitEthernet>
  <TenGigabitEthernet>
    <name>1/1/1</name>
    <service-policy>
      <input>TRUST-MARKING</input>
    </service-policy>
  </TenGigabitEthernet>
  <Vlan>
    <name>10</name>
    <service-policy>
      <input>classify</input>
    </service-policy>
  </Vlan>
```

Back to NSO

The NSO Database is an XML database.

All devices' in NSO have their configuration stored in the XML Database

All Data is stored in XML, and is in a hierarchy (see previous slides)

You can see the XML and save it to a file, use it as a template, or use it to import the data into NSO

```
admin@ncs# show running-config devices device rtp5-itnlab-dt-sw1.cisco.com | display xml
```

NSO XML device data

```
<config xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>rtp5-itnlab-dt-sw1.cisco.com</name>
      <address>rtp5-itnlab-dt-sw1.cisco.com</address>
      <port>22</port>
      <ssh>
        <host-key>
          <algorithm>ssh-rsa</algorithm>
          <key-data>AAAAB3NzaC1yc2EAAAQABAAAAgQC0uw3/fNscjQqUQhxLEh4Y82vSZB7K5yR2h+XFaB0c
432kN16CC5QukBBDn06o80N4E009EaBFz7VVuNYDa5dz1SoeJJtGXmP4/mwxUNmRYIUEA0dF
5CU9YqlcGLawSnj+ndBYoxkSbV0IA3wHy6JYwEB/Rw6CH4dGWW+lju7ujw==</key-data>
        </host-key>
      </ssh>
      <authgroup>jabek.web.auth</authgroup>
      <device-type>
        <cli>
          <ned-id xmlns:ios-id="urn:ios:ios-id">ios-id:cisco-ios</ned-id>
        </cli>
      </device-type>
      <state>
        <admin-state>unlocked</admin-state>
      </state>
    </device>
  </devices>
</config>
```

Config data for boot vars

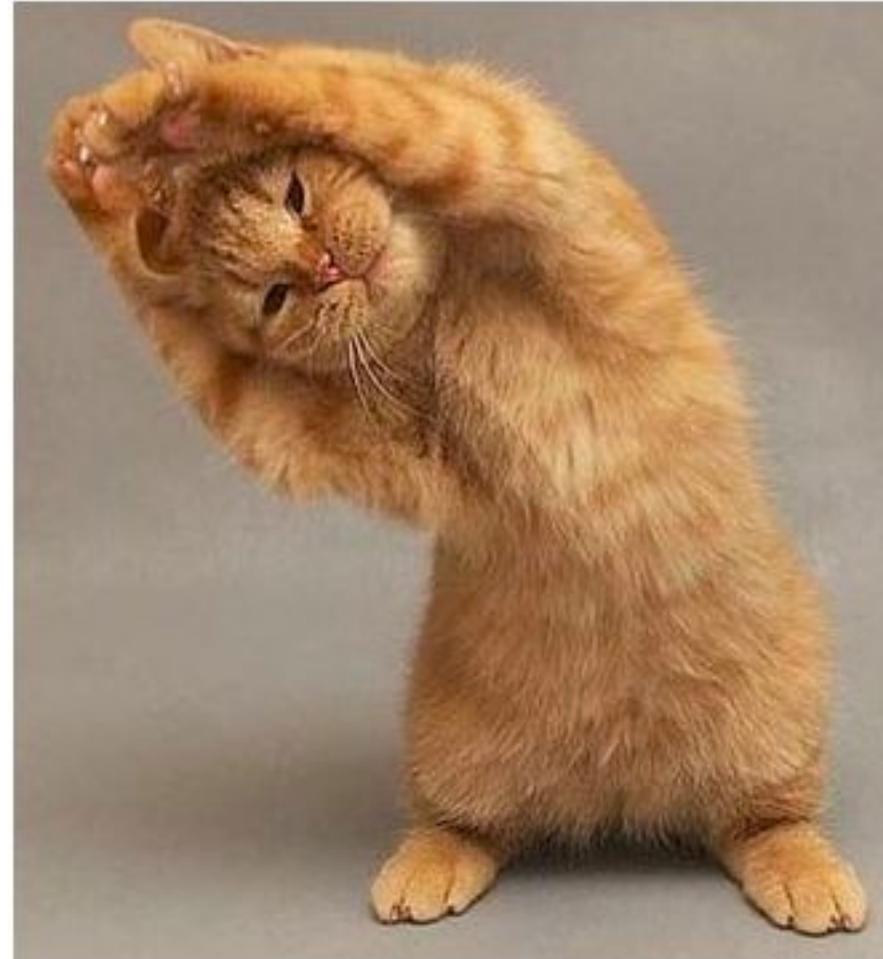
```
<config>
    <hostname xmlns="urn:ios">rtp5-itnlab-dt-sw1</hostname>
    <boot-marker xmlns="urn:ios">
        <boot>
            <system>
                <entry>flash bootflash:cat4500es8-universalk9.SPA.03.06.05.E.152-2.E5.bin</entry>
            </system>
            <system>
                <entry>flash slot0:cat4500es8-universalk9.SPA.03.06.05.E.152-2.E5.bin</entry>
            </system>
            <system>
                <entry>flash bootflash:cat4500es8-universalk9.SPA.03.08.02.E.152-4.E2.bin</entry>
            </system>
            <system>
                <entry>flash slot0:cat4500es8-universalk9.SPA.03.08.02.E.152-4.E2.bin</entry>
            </system>
        </boot>
    </boot-marker>
```

So why do we need this?

XML Summary

- NSO Stores and represents its Database in XML
- XML is NSO's 'Native' Language
- While we rarely interact directly with XML in NSO, it's very important to understand how NSO is representing and navigating the data
- Our service packages will leverage XML configuration templates
- We pass variables into XML to generate config for our designs
- Knowing XML makes troubleshooting much easier

Stretch Break!



REST APIs

Pre-Quiz: What is a REST API?



Building Blocks of NSO REST API

- Everything is within a hierarchy, beginning with 'LINUX_HOSTNAME:8080/api'
- Typical flow:
 - Choose REST API client (Postman, Bash curl, python requests, etc)
 - Choose a REST API URI (API URL path), see first bullet point
 - Choose a REST API Operation (Post, Get, Put, Delete, etc.)
 - Add authentication to the request (default is Basic Auth, admin/admin)
 - Send request and check status (200, 400, 401 etc) and output



REST API Key Components

- A working URI path (it looks like a website address)
- Necessary headers (like login credentials for auth, if want JSON response, add it)
- A way to deliver the rest call (use postman, curl, python requests)

REST API

- POSTMAN: Free Chrome App - <https://www.getpostman.com/apps>
- Examples:
 - GET servername:8080/api/running/devices
 - GET servername:8080/api/running/devices/device/(name)/config?deep
 - POST servername:8080/api/running/devices/device/(name)/config/hostname
- For more info see NSO Docs:
 - nso_northbound-4.4.pdf – Chapter 3

URI in Postman

Example:

svl-sjc-nso-1:8080/api/running/devices

A screenshot of the Postman application interface. At the top, there are two input fields: the first contains 'https://devnetapi.cisco.com' and the second contains 'svl-sjc-nso-1:8080/api...'. To the right of these fields is a '+' button and a dropdown menu labeled 'No Environment'. Below this, the main interface shows a 'GET' method selected on the left, followed by the full URL 'svl-sjc-nso-1:8080/api/...'. To the right of the URL are three buttons: 'Params', 'Send' (which is highlighted in blue), and a dropdown menu.

Use localhost instead of svl-sjc-nso-1 if you are on a VM

You Forgot Authentication!

The screenshot shows the Postman application interface with the following details:

- Header Bar:** Shows the URL `https://devnetapi.cisco.com`, a tab for `svl-sjc-nso-1:8080/api/`, another tab for `svl-sjc-nso-1:8080/api.` (with a close button), and a plus icon for creating new environments. To the right are buttons for "No Environment" (with a dropdown arrow), and icons for "eye" and "gear".
- Request Section:** Method is set to "GET" and the URL is `svl-sjc-nso-1:8080/api/running/devices...`. Below the URL are buttons for "Params", "Send" (highlighted in blue), and "Save".
- Authorization Tab:** Active tab, showing "Type" set to "No Auth".
- Body Tab:** Active tab, showing the response body. Headers section indicates 7 items. Status: **401 Unauthorized**, Time: **432 ms**.
- Body Content:** The response body is displayed in "Pretty" format:

```
1 <html>
2   <body>
3     <h1>401 authentication needed</h1>
4   </body>
5 </html>
```

Adding REST Authentication in Postman

The screenshot shows the Postman interface for adding REST authentication. The top navigation bar includes tabs for Authorization (selected), Headers (1), Body, Pre-request Script, and Tests. The main area shows the 'Authorization' tab with fields for Type, Username, and Password. A dropdown menu under 'Type' is open, showing options: No Auth (disabled), Basic Auth (selected), Digest Auth, OAuth 1.0, OAuth 2.0, Hawk Authentication, and AWS Signature. To the right of the dropdown are the filled-in fields: Username: admin and Password: admin. A checked checkbox labeled 'Show Password' is also visible.

Authorization • Headers (1) Body Pre-request Script Tests

Type

Basic Auth

No Auth

Basic Auth

Digest Auth

OAuth 1.0

OAuth 2.0

Hawk Authentication

AWS Signature

Username

admin

Password

admin

Show Password

Body Cookies Headers (7) Test

https://devnetapi.cisco.com svl-sjc-nso-1:8080/api/ svl-sjc-nso-1:8080/api. + No Environment

GET svl-sjc-nso-1:8080/api/running/devices... Params Send Save

Type Basic Auth Clear Update Request

Username admin The authorization header will be generated and added as a custom header

Password Save helper data to request Show Password

Body Cookies Headers (8) Tests Status: 200 OK Time: 544 ms

Pretty Raw Preview XML

```
1 <devices xmlns="http://tail-f.com/ns/ncs" xmlns:y="http://tail-f.com/ns/rest" xmlns:ncs="http://tail-f.com/ns/ncs">
2   <global-settings>
3     <trace-dir>./logs</trace-dir>
4   </global-settings>
5   <authgroups>
6     <group>
7       <name>SVL_Devices</name>
8     </group>
9     <group>
10      <name>default</name>
11    </group>
12    <snmp-group>
13      <name>default</name>
14    </snmp-group>
15  </authgroups>
16  <mib-group>
17    <name>snmp</name>
18  </mib-group>
```

Now You Try

- Try using the REST API to:
 - Get a specific devices configuration
 - Get just the hostname for a device
- Advanced:
 - Change the devices hostname via a POST call

NSO Query Rest API

NSO Query API

- One of the most powerful things about a database is the ability to query it
- NSO REST API gives us a way to query and get information out of the NSO cDB



NSO Query API

- The NSO REST API also exposes a query functionality for quickly retrieving information from the cDB via xPath expressions.
- Works by POST of a payload to `http:server/api/query`
- A query handle ID is returned
- Results returned via POST of the handle back to the API end point
- Can send in JSON or XML
- Requires knowledge of xPath

Payload Structure

- Payload has two required options. Foreach and select
- Foreach is the node to iterate over
- Select is the attributes to select from it
- XPATH for both

The screenshot shows the Postman Builder interface. On the left, the History tab displays a list of API requests, including several GET and POST requests to localhost:5000/mine and localhost:8080/api/query. On the right, the main area shows a POST request to localhost:8080/api/query. The Body tab is selected, displaying the following XML payload:

```
1 <start-query xmlns="http://tail-f.com/ns/tailf-rest-query">
2 <foreach>
3 /ncs:devices/device-group/customer_gws/
4 </foreach>
5 <select>
6 <label>Host name</label>
7 <expression>device-name</expression>
8 <result-type>string</result-type>
9 </select>
10 <sort-by>name</sort-by>
11 <limit>100</limit>
12 <offset>1</offset>
13 </start-query>
```

Below the payload, a "Response" section is visible with the placeholder text "Hit the Send button to get a response."

NSO Query API Wrapper

- We have been working on simplifying the experience via a python wrapper class for ‘SQL-Like’ input

```
from nso_query_tool import NsoServer, NsoQuery

server = NsoServer('server', "user", "pass")
select = ['name', "config/ios:ip/http/server", "platform/model", "platform/version"]
_from = [{"device-group":"acc1-pl"}, {"device":"acc3-cn-sw1"}]
where = ["config/ios:tacacs/timeout='3'", "and", "config/ios:cdp/run ='true'"]
query = NsoQuery(server, _from=_from, select=select, where=where)
for item in query.results:
    print item
print query.results.length()
```

sample results:

```
{u'platform/version': u'15.2(4)M1', u'platform/model': u'CISCO2901/K9', u'name': u'acc1-pl-cs1', u'config/ios:ip/http/server': u'false'}
{u'platform/version': u'03.06.05E', u'platform/model': u'WS-C3850-48P', u'name': u'acc1-pl-sw1', u'config/ios:ip/http/server': u'true'}
{u'platform/version': u'15.5(3)S2', u'platform/model': u'ISR4451-X/K9', u'name': u'acc1-pl-wan-gw1', u'config/ios:ip/http/server': u'false'}
{u'platform/version': u'15.5(3)S2', u'platform/model': u'ISR4451-X/K9', u'name': u'acc1-pl-wan-gw2', u'config/ios:ip/http/server': u'false'}
```

4

Feedback - Survey



WELCOME TO DAY THREE!

Agenda

- Services
- Building a Service
- YANG + XML

What are NSO Services?

- Abstractions of Network Configurations across devices
- Think programmatic Design Documents/Cookbooks
- Stores “Service”/“Design” instance input data into the NSO cDB
- Provides a database of what service/design is deployed where and what the configuration is.
- Single entry point for updating global standards. Update the service design and deploy the update to all devices it applies to.

The “Service” in Network Service Orchestrator

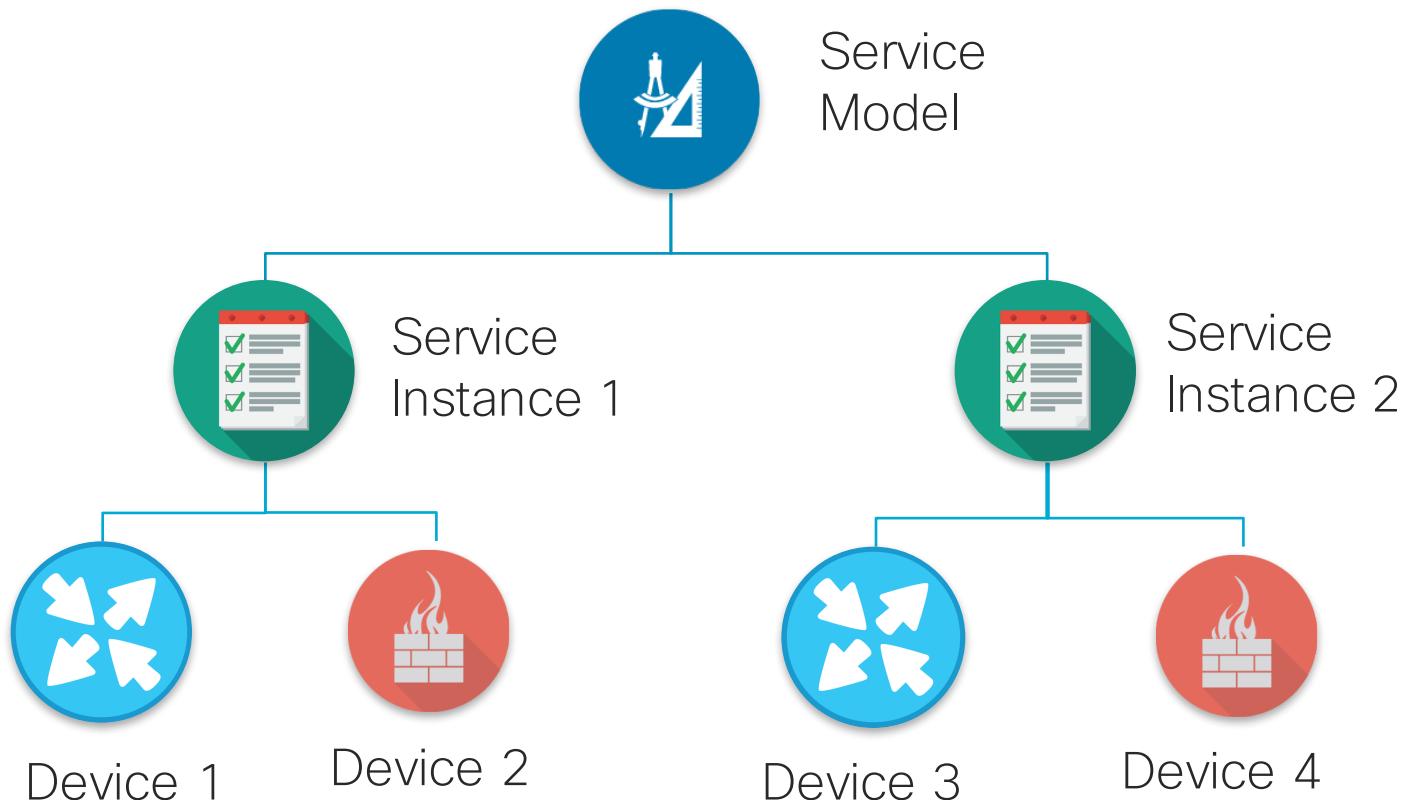
A Network Service is a collection of configurations across one or multiple devices and servers that enable a capability.

An example is Basic Wireless Service:



NSO Service Manager & Models

NSO Manages Network Services through the Service Model Construct:



Designed in YANG and modeled with XML templates, code or both

Instantiated in NSO with input parameters
Service's lifecycle is managed by NSO
(Deployment, Compliance and Removal)

NSO Pushes and tracks configuration based on service input parameters to the devices.

Demo

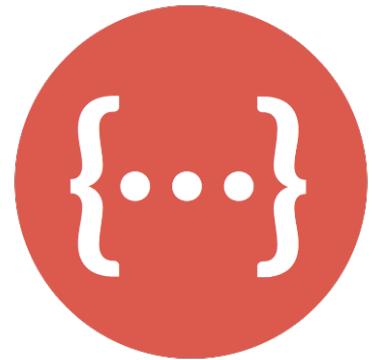
Service Features & Capabilities

- Service Database
- Compliance Reporting
- Compliance Remediation
- Global Updates
- APIs

NSO Service Package

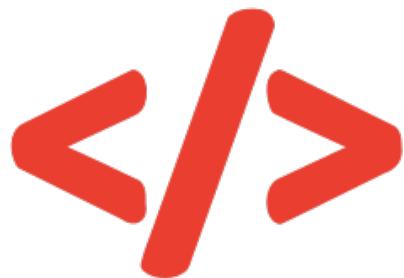
Service Packages are how NSO Implements the Service Construct, Packages include YANG, XML & Code and are loaded into NSO

[YANG](#)



Service Model

[XML](#)



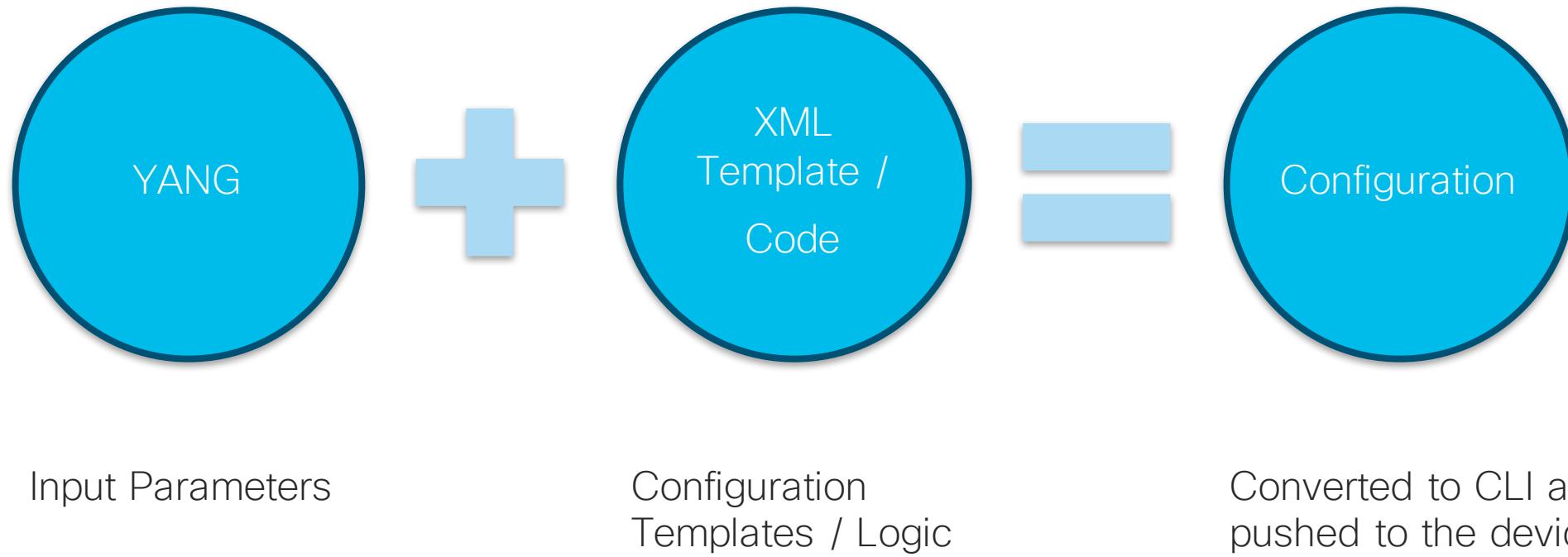
Service Template

[Python and Java](#)



Service Logic

NSO Service Packages



Service Model YANG

YANG Modules in the service package defines the Service **input parameters** and the **meta-data** to be stored

YANG attributes are stored inside the cDB for reference and used in **determining configuration** to be applied.

Building a Service

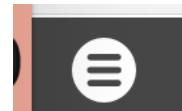
Demo building a service package

- ncs-make-package --service-skeleton template bgl-test
- Took out some dummy values, change into directory, remove xml template file for now, run make in src folder, and then packages reload

```
module bgl-test {
    namespace "http://com/example/bgltest";
    prefix bgl-test;

    import ietf-inet-types {
        prefix inet;
    }
    import tailf-ncs {
        prefix ncs;
    }
    |
    leaf a_yang_input {
        type string;
    }
}
```

```
JABELK-M-D3BK:yang jabelk$ cd ..
JABELK-M-D3BK:src jabelk$ make
mkdir -p ../load-dir
/Users/jabelk/ncs-all/ncs-4.4/bin/ncsc `ls bgl-test-ann.yang > /dev/null 2>&1 && echo "-a bgl-test-ann.yang"` \
-c -o ../load-dir/bgl-test.fxs yang/bgl-test.yang
JABELK-M-D3BK:src jabelk$ kickofffns
```



```
admin@ncs# packages reload
reload-result {
    package bgl-test
    result true
}
```

Models

```
▼ Services
  (No services)
▼ Modules
  ▼ bgl-test
    my_tab_around_this
```

GUI
or
CLI

```
admin@ncs(config)# my_tab_around_this ?
Possible completions:
  a_yang_input
admin@ncs(config)# my_tab_around_this a_yang_input ?
Possible completions:
  <string>
admin@ncs(config)# my_tab_around_this a_yang_input
```

CLI to GUI

```
admin@ncs(config)# my_tab_around_this a_yang_input "hello there"  
admin@ncs(config)# commit  
Commit complete.
```



NSO 4.4 Commit

/ bgl-test:my_tab_around_this

my_tab_around_this

INFO
a_yang_input
hello there

Or GUI to CLI

NSO 4.4 Commit

/ bgl-test:my_tab_around_this

my_tab_around_this

INFO
a_yang_input
I want candy



```
admin@ncs#  
System message at 2017-10-18 18:48:18...  
Commit performed by admin via http using webui.
```



```
admin@ncs# show running-config my_tab_around_this a_yang_input  
my_tab_around_this a_yang_input "I want candy"  
admin@ncs#
```

Lab - bgl test service

Developing Services

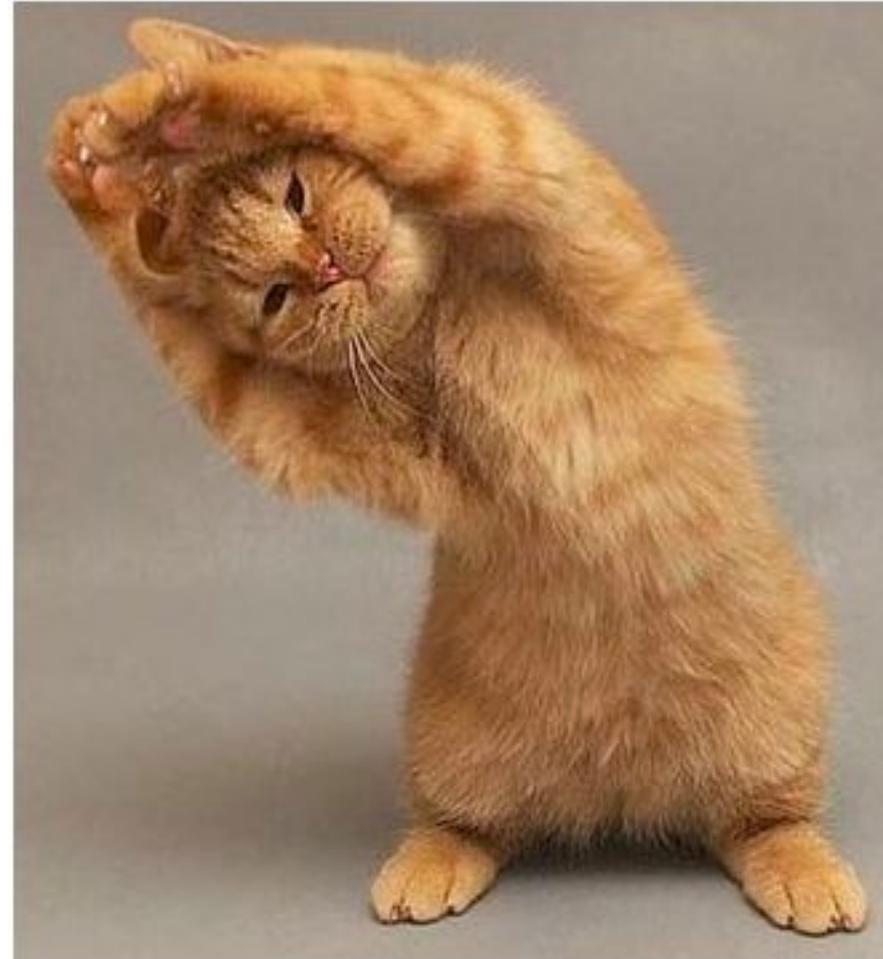
High-level development flow

- Define input parameters (YANG)
- Develop Cutsheet (CLI)
- Map Input Parameters to cutsheet (XML)
- Load into NSO
- Test

Demo

Lab - Simple Service

Stretch Break!



Service Database

- Service model inputs are defined in YANG
- Data is stored in the cDB following the services YANG model

/ncs:services/lab_gateway_ACL:lab_gateway_ACL

The screenshot shows a table with a header row containing a checkbox and the column title "name". Below the header are six rows, each containing a checkbox and a host name: abj01-lab-gw1.cisco.com, abq-lab-gw1.cisco.com, adl-lab-gw1.cisco.com, akl02-lab-gw1.cisco.com, alb-lab-gw1.cisco.com, and alln01-lab-gw1.cisco.com.

	name
1	abj01-lab-gw1.cisco.com
2	abq-lab-gw1.cisco.com
3	adl-lab-gw1.cisco.com
4	akl02-lab-gw1.cisco.com
5	alb-lab-gw1.cisco.com
6	alln01-lab-gw1.cisco.com

◀ / ncs:services ▶ / lab_gateway_ACL:lab_gateway_ACL ▶ { bdlk09-00-lab-gw1.cisco.com } ▶ / Uplink_Interfaces ▶ / GigabitEthernet ▶

GigabitEthernet

The screenshot shows a table with a header row containing a plus sign (+), a minus sign (-), an "Actions" button, and a filter icon. Below the header are two rows, each containing a checkbox and an interface number: 1/1 and 1/2.

+	-	Actions	Filter
		#	interfaceNumber (k)
		<input type="checkbox"/>	1/1
		<input type="checkbox"/>	1/2

“Querying” the Database

- Through the Python API or REST API we can write code to get information from the cDB
- Query: What are the uplink interfaces for lab-gateway bdlk09-00-lab-gw1.cisco.com ?
- REST API Call:

GET

https://nwsnsoprd-1.cisco.com:8888/api/services/lab_gateway_ACL/bdlk09-00-lab-gw1.cisco.com

```
<lab_gateway_ACL xmlns="http://com/example/labgatewayACL
  <name>bdlk09-00-lab-gw1.cisco.com</name>
  <device>bdlk09-00-lab-gw1.cisco.com</device>
  <Uplink_Interfaces>
    <GigabitEthernet>
      <interfaceNumber>1/1</interfaceNumber>
    </GigabitEthernet>
    <GigabitEthernet>
      <interfaceNumber>1/2</interfaceNumber>
    </GigabitEthernet>
  </Uplink_Interfaces>
```

Can use XML or JSON.

Parse the output and use it as you wish!

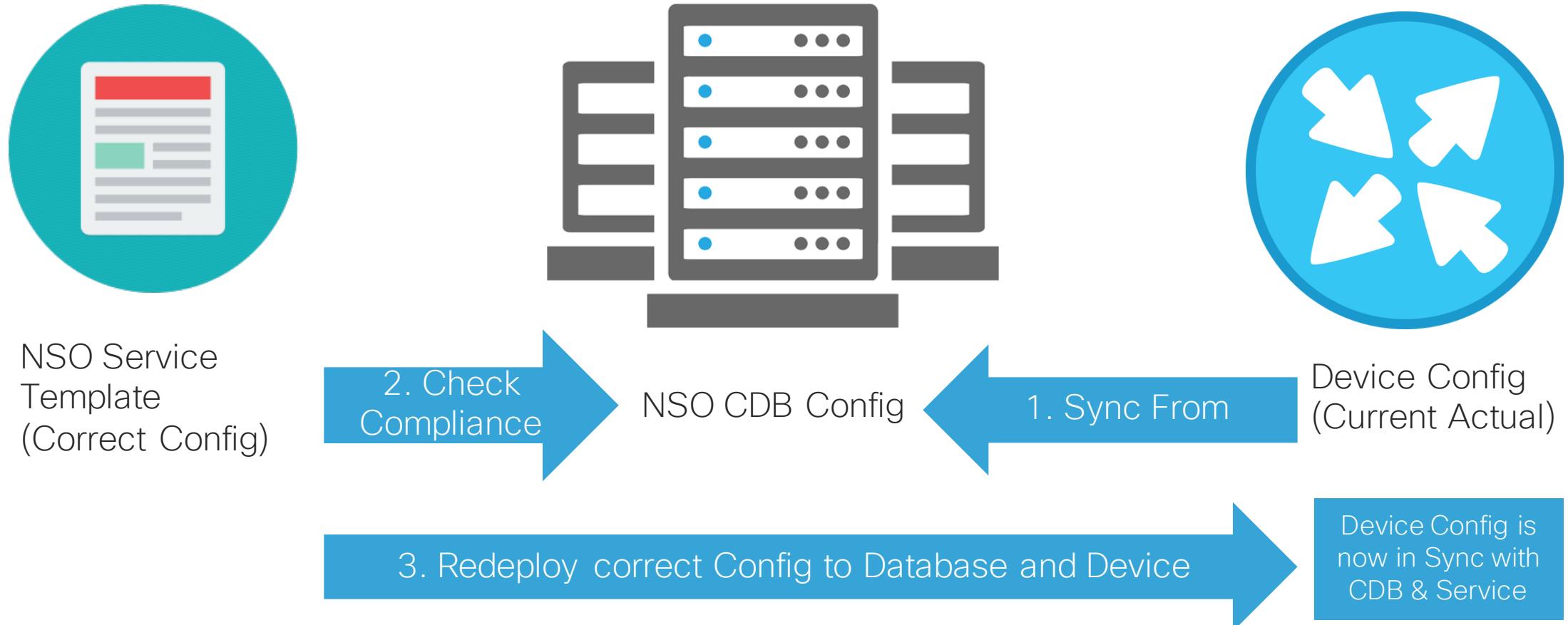
Compliance Reporting

Demo – Lab lenient

Compliance Reporting

- Once we have deployed our services into the network we may want to periodically check if the configuration is still in place
- Since NSO knows what each instance of a service is *suppose to* be and *what it currently is* we can compare the two

Compliance Process



Service Instance Check-Sync

- A service instance check-sync will check if all the devices for a specific service instance are “in-sync” or compliant with the service model.
- Uses the saved input parameters for the service models YANG for that instance to determine what the **config should be**, and checks against what the **device's config is**

Lab - QoS

Lunch

XML Template Methods

- We can actually build logic into our XML templates
- The most common methods are a “For” Lop and “If” statement
- “If” logic is handled through a “when” statement
- “For” loop logic is handled through a “foreach” statement
- Both leverage YANG paths for their operations

XML – When statement

- We use a when statement to replicate a programming “If”
- References our service nodes
- Only applies nested XML if/when the specified information is true

```
1 <config-template xmlns="http://tail-f.com/ns/config/1.0"
2   servicepoint="radius_example">
3   <devices xmlns="http://tail-f.com/ns/ncs">
4     <device>
5       <name>{/device}</name>
6       <config>
7         <radius xmlns="urn:ios" when="{/Region='Amer'}">
8           <server>
9             <id>one</id>
10            <address>
11              <ipv4>
12                <host>10.0.0.1</host>
13                <auth-port>1812</auth-port>
14                <acct-port>1813</acct-port>
15              </ipv4>
16            </address>
17
18            <backoff>
19              <exponential></exponential>
20            </backoff>
21          </server>
22        </radius>
23        <radius xmlns="urn:ios" when="{/Region='APJC'}">
24          <server>
25            <id>one</id>
26            <address>
27              <ipv4>
28                <host>10.0.0.2</host>
29                <auth-port>1812</auth-port>
30                <acct-port>1813</acct-port>
31              </ipv4>
32            </address>
```

XML – Foreach statement

- We use a when statement to replicate a programming “for” loop
- References our service nodes and applies the xml template ‘foreach’ of that node (think foreach item in a list)

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                  servicepoint="acl_lab">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device foreach="{/devices}">
      <name>{device_name}</name>
      <config>
        <interface xmlns="urn:ios" foreach="{interfaces}" >
          </interface>
        </config>
      </device>
    </devices>
</config-template>
```

Demo –
foreach and when in action

Lab –
Build your Own Use Case!
Or follow our lab

Feedback - Survey

WELCOME TO DAY FOUR!

Agenda

- SERVICES INTRODUCTION
- SERVICE FEATURES
- SERVICE DEVELOPMENT
- LAB: TEMPLATE SERVICE
- LUNCH
- SERVICES WITH CODE
- LAB: PYTHON SERVICE
- RECAP and Q&A

Compliance Remediation

Compliance Remediation

- To remediate a service that is non-compliant or “out-of-sync” NSO needs to re-run the service logic, determine the configuration and re-deploy it to the network
- NSO Service Instances have a pre-built function to execute these tasks called “re-deploy”
- NSO re-calculates the configuration needed based upon what the devices should be and what they are and corrects the differences

Service Instance Re-Deploy

Navigate to a service instance and enter the “re-deploy...” action

The screenshot shows the NSO 4.3 interface with the following details:

- Header:** NSO 4.3, Commit ▾, Views ▾, Jobs, Alarms 5078, branblac ▾.
- Breadcrumbs:** / ncs:services ▾ / lab_gateway_ACL:lab_gateway_ACL ▾ { ams5-lab-gw1.cisco.com } ▾.
- Toolbar:** lab_gateway_ACL (selected), modified, directly-modified, commit-queue, log, Uplink_Interfaces.
- INFO section:** name: ams5-lab-gw1.cisco.com.
- device-list:** ams5-lab-gw1.cisco.com.
- used-by-customer-service:** No items to display.
- device:** ams5-lab-gw1.cisco.com (with a + icon).
- Action Buttons:** check-sync..., deep-check-sync..., re-deploy..., reactive-re-deploy..., touch..., get-modifications..., un-deploy... (the re-deploy... button is highlighted).

Service Instance Re-Deploy

Inside we have several key options that can be set before “re-deploying” the service configuration

- Dry-run
 - Simulate what the deployment will be
- No-networking
 - Only re-deploy to the cDB NOT the network

The screenshot shows the NSO 4.4 interface with the following details:

- Header:** NSO 4.4, Commit, Views, Jobs, Alarms (4), admin.
- Breadcrumbs:** / ncs:services / demo:demo { demo } / re-deploy
- Page Title:** Run/Dryrun the service logic again
- Buttons:** re-deploy (highlighted), Invoke re-deploy
- Form Fields:**
 - dry-run:** no-revision-drop (checkbox), no-networking (checkbox)
 - commit-queue:** Choice - choice-sync-check (dropdown)
 - reconcile:** Choice - choice-lsa (dropdown), Choice - depth (dropdown), Choice - outformat (dropdown)
- Table:** Name Value (empty)
- Message:** No items to display

A note about Re-Deploys

- NSO only has the native ability for a user to re-deploy one service instance at time
- Each re-deploy is its own transaction with a rollback file
- Re-deploys should also be-used when the service model (design) changes or is updated. NSO will then re-calculate what the "correct" config is based upon the update and push the delta

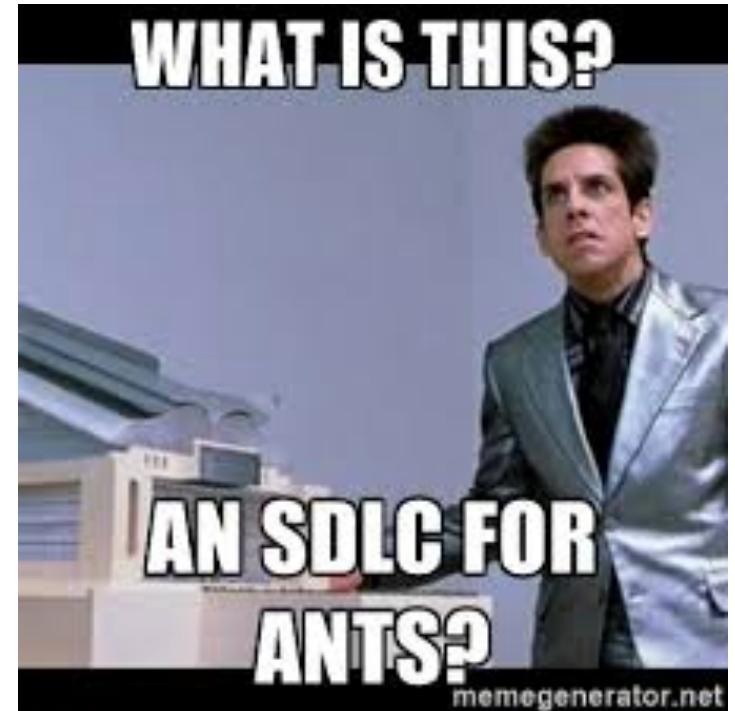
Service Development

Learn by doing!

- You will have labs, but the following slides will build a simple service package for a simple radius config that has 3 regional servers

Service Development Lifecycle

- Gather Requirements
- Determine appropriate input parameters
- Determine Configuration (CLI or XML)
- Map input parameters to configuration
- Make YANG Model
- Make XML Template
- Test!



Gather Requirements

- Does the service/design apply to one device or many?
- Should it accept single devices or groups of devices?
- Are there regional / site differences?
- Is the config dependent upon any other factors?
- Is there relevant meta-data we want to store with an instance?
- Is there cross OS dependencies?
- Etc....

Gather Requirements - Example

- Service instance should accept multiple devices, perhaps even an NSO defined device group
- Each instance of the service should represent one of the regions
- Each regional instance will apply a different ipv4 radius server
- Just for IOS/IOS-XE devices
- Meta-Data, lets store the data-center name for the radius server (just for fun)

Determine appropriate input parameters

- What is variable inside the configuration?
- What aspects of the configuration are dependent upon an inputted factor?
- Where would we get the input parameters?

EXAMPLE:

For our use case, the input parameters are pretty straightforward. We want to input which region the instance is for, the DC for the server, and a list of devices that it applies to.

Determine Configuration (CLI or XML)

- What configuration are we applying?
- What is the CLI that is applied?
- Is it applied to multiple interfaces?
- Is there interdependencies?
- Get CLI examples to use as a starting point for generating XML templates

Determine Configuration (CLI or XML) -- Example

For our example we want to apply the following CLI:

radius server one

 address ipv4 10.0.0.1 auth-port 1812 acct-port 1813

 backoff exponential

!

Within this config, the ipv4 address will change per region:

AMER: 10.0.1.1

APJC: 10.0.2.1

EMEAR: 10.0.3.1

Map input parameters to configuration

- Now lets parameterize our CLI example
- Determine where the config changes based upon the input parameters (Like the region in our example)
 - Determine what the different configs are per permutations
- Determine where the input parameters go directly into the config (for example ip-address, or AS# maybe an put parameter)
- Determine if any loops or iterations over config elements are needed (like applying a config per interface)

Map input parameters to configuration --Example

- Lets map how the config changes per theater:

radius server one

```
address ipv4 (IP_ADDRESS) auth-port 1812 acct-port 1813  
backoff exponential
```

!

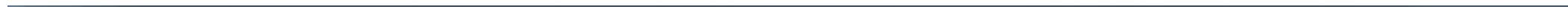
- If Region = AMER then IP_ADDRESS = 10.0.1.1
 - If Region = APJC then IP_ADDRESS = 10.0.2.1
 - If Region = EMEAR then IP_ADDRESS = 10.0.3.1
-
- Then we want to apply this to each inputted device

Make YANG Model

- Now that we know our input parameters and meta-data we need to model that information in YANG
- We start by making a new template-service framework with the NSO-CLI
 - `ncs-make-package --service-skeleton template radius`
- Then inside radius /src/yang we can open up the `radius.yang` file
- Modify the file with the appropriate YANG data-types to meet your inputs (it comes with examples inside)
- `pyang` is a command line tool to help with yang file debugging and validation

Make YANG Model -- Example

```
[BRANBLAC-M-W0WN:packages branblac$ ncs-make-package --service-skeleton template dhcp  
BRANBLAC-M-W0WN:packages branblac$
```



```
Project — ~/ncs-run/netsim-dmeo/packages/Audit-and-Remediation-Framework
```

Project	main.py	audit.yang	ipv6.py	input_output.yang	dhcp.yang
Audit-and-Remediation-Framework					
NSO-ipv6-cleanup					
packages					
Audit-and-Remediation-Framework					
cisco-ios					
demo					
dhcp					
src					
yang					
dhcp.yang					
Makefile					
templates					
test					
package-meta-data.xml					
lab_gateway_ACL					
lab_gateway_Cross_Connect					
lab_gateway_Master					
lab_gateway_QoS					
Object_group_cleaner					
.DS_Store					
Audit_CLI					

```
1 module dhcp {  
2   namespace "http://com/example/dhcp";  
3   prefix dhcp;  
4  
5   import ietf-inet-types {  
6     prefix inet;  
7   }  
8   import tailf-ncs {  
9     prefix ncs;  
10  }  
11  
12  list dhcp {  
13    key name;  
14  
15    uses ncs:service-data;  
16    ncs:servicepoint "dhcp";  
17  
18    leaf name {  
19      type string;  
20    }  
21  
22    // may replace this with other ways of referring to the devices.  
23    leaf-list device {  
24      type leafref {  
25        path "/ncs:devices/ncs:device/ncs:name";  
26      }  
27    }  
28  }  
29  
30  // may replace this with other ways of referring to the devices.  
31  leaf-list device {  
32    type leafref {  
33      path "/ncs:devices/ncs:device/ncs:name";  
34    }  
35  }  
36  
37  // may replace this with other ways of referring to the devices.  
38  leaf-list device {  
39    type leafref {  
40      path "/ncs:devices/ncs:device/ncs:name";  
41    }  
42  }  
43  
44  // may replace this with other ways of referring to the devices.  
45  leaf-list device {  
46    type leafref {  
47      path "/ncs:devices/ncs:device/ncs:name";  
48    }  
49  }  
50  
51  // may replace this with other ways of referring to the devices.  
52  leaf-list device {  
53    type leafref {  
54      path "/ncs:devices/ncs:device/ncs:name";  
55    }  
56  }  
57  
58  // may replace this with other ways of referring to the devices.  
59  leaf-list device {  
60    type leafref {  
61      path "/ncs:devices/ncs:device/ncs:name";  
62    }  
63  }  
64  
65  // may replace this with other ways of referring to the devices.  
66  leaf-list device {  
67    type leafref {  
68      path "/ncs:devices/ncs:device/ncs:name";  
69    }  
70  }  
71  
72  // may replace this with other ways of referring to the devices.  
73  leaf-list device {  
74    type leafref {  
75      path "/ncs:devices/ncs:device/ncs:name";  
76    }  
77  }  
78  
79  // may replace this with other ways of referring to the devices.  
80  leaf-list device {  
81    type leafref {  
82      path "/ncs:devices/ncs:device/ncs:name";  
83    }  
84  }  
85  
86  // may replace this with other ways of referring to the devices.  
87  leaf-list device {  
88    type leafref {  
89      path "/ncs:devices/ncs:device/ncs:name";  
90    }  
91  }  
92  
93  // may replace this with other ways of referring to the devices.  
94  leaf-list device {  
95    type leafref {  
96      path "/ncs:devices/ncs:device/ncs:name";  
97    }  
98  }  
99  
100 // may replace this with other ways of referring to the devices.  
101 leaf-list device {  
102   type leafref {  
103     path "/ncs:devices/ncs:device/ncs:name";  
104   }  
105 }  
106  
107 // may replace this with other ways of referring to the devices.  
108 leaf-list device {  
109   type leafref {  
110     path "/ncs:devices/ncs:device/ncs:name";  
111   }  
112 }  
113  
114 // may replace this with other ways of referring to the devices.  
115 leaf-list device {  
116   type leafref {  
117     path "/ncs:devices/ncs:device/ncs:name";  
118   }  
119 }  
120  
121 // may replace this with other ways of referring to the devices.  
122 leaf-list device {  
123   type leafref {  
124     path "/ncs:devices/ncs:device/ncs:name";  
125   }  
126 }  
127  
128 // may replace this with other ways of referring to the devices.  
129 leaf-list device {  
130   type leafref {  
131     path "/ncs:devices/ncs:device/ncs:name";  
132   }  
133 }  
134  
135 // may replace this with other ways of referring to the devices.  
136 leaf-list device {  
137   type leafref {  
138     path "/ncs:devices/ncs:device/ncs:name";  
139   }  
140 }  
141  
142 // may replace this with other ways of referring to the devices.  
143 leaf-list device {  
144   type leafref {  
145     path "/ncs:devices/ncs:device/ncs:name";  
146   }  
147 }  
148  
149 // may replace this with other ways of referring to the devices.  
150 leaf-list device {  
151   type leafref {  
152     path "/ncs:devices/ncs:device/ncs:name";  
153   }  
154 }  
155  
156 // may replace this with other ways of referring to the devices.  
157 leaf-list device {  
158   type leafref {  
159     path "/ncs:devices/ncs:device/ncs:name";  
160   }  
161 }  
162  
163 // may replace this with other ways of referring to the devices.  
164 leaf-list device {  
165   type leafref {  
166     path "/ncs:devices/ncs:device/ncs:name";  
167   }  
168 }  
169  
170 // may replace this with other ways of referring to the devices.  
171 leaf-list device {  
172   type leafref {  
173     path "/ncs:devices/ncs:device/ncs:name";  
174   }  
175 }  
176  
177 // may replace this with other ways of referring to the devices.  
178 leaf-list device {  
179   type leafref {  
180     path "/ncs:devices/ncs:device/ncs:name";  
181   }  
182 }  
183  
184 // may replace this with other ways of referring to the devices.  
185 leaf-list device {  
186   type leafref {  
187     path "/ncs:devices/ncs:device/ncs:name";  
188   }  
189 }  
190  
191 // may replace this with other ways of referring to the devices.  
192 leaf-list device {  
193   type leafref {  
194     path "/ncs:devices/ncs:device/ncs:name";  
195   }  
196 }  
197  
198 // may replace this with other ways of referring to the devices.  
199 leaf-list device {  
200   type leafref {  
201     path "/ncs:devices/ncs:device/ncs:name";  
202   }  
203 }  
204  
205 // may replace this with other ways of referring to the devices.  
206 leaf-list device {  
207   type leafref {  
208     path "/ncs:devices/ncs:device/ncs:name";  
209   }  
210 }  
211  
212 // may replace this with other ways of referring to the devices.  
213 leaf-list device {  
214   type leafref {  
215     path "/ncs:devices/ncs:device/ncs:name";  
216   }  
217 }  
218  
219 // may replace this with other ways of referring to the devices.  
220 leaf-list device {  
221   type leafref {  
222     path "/ncs:devices/ncs:device/ncs:name";  
223   }  
224 }  
225  
226 // may replace this with other ways of referring to the devices.  
227 leaf-list device {  
228   type leafref {  
229     path "/ncs:devices/ncs:device/ncs:name";  
230   }  
231 }  
232  
233 // may replace this with other ways of referring to the devices.  
234 leaf-list device {  
235   type leafref {  
236     path "/ncs:devices/ncs:device/ncs:name";  
237   }  
238 }  
239  
240 // may replace this with other ways of referring to the devices.  
241 leaf-list device {  
242   type leafref {  
243     path "/ncs:devices/ncs:device/ncs:name";  
244   }  
245 }  
246  
247 // may replace this with other ways of referring to the devices.  
248 leaf-list device {  
249   type leafref {  
250     path "/ncs:devices/ncs:device/ncs:name";  
251   }  
252 }  
253  
254 // may replace this with other ways of referring to the devices.  
255 leaf-list device {  
256   type leafref {  
257     path "/ncs:devices/ncs:device/ncs:name";  
258   }  
259 }  
260  
261 // may replace this with other ways of referring to the devices.  
262 leaf-list device {  
263   type leafref {  
264     path "/ncs:devices/ncs:device/ncs:name";  
265   }  
266 }  
267  
268 // may replace this with other ways of referring to the devices.  
269 leaf-list device {  
270   type leafref {  
271     path "/ncs:devices/ncs:device/ncs:name";  
272   }  
273 }  
274  
275 // may replace this with other ways of referring to the devices.  
276 leaf-list device {  
277   type leafref {  
278     path "/ncs:devices/ncs:device/ncs:name";  
279   }  
280 }  
281  
282 // may replace this with other ways of referring to the devices.  
283 leaf-list device {  
284   type leafref {  
285     path "/ncs:devices/ncs:device/ncs:name";  
286   }  
287 }  
288  
289 // may replace this with other ways of referring to the devices.  
290 leaf-list device {  
291   type leafref {  
292     path "/ncs:devices/ncs:device/ncs:name";  
293   }  
294 }  
295  
296 // may replace this with other ways of referring to the devices.  
297 leaf-list device {  
298   type leafref {  
299     path "/ncs:devices/ncs:device/ncs:name";  
300   }  
301 }  
302  
303 // may replace this with other ways of referring to the devices.  
304 leaf-list device {  
305   type leafref {  
306     path "/ncs:devices/ncs:device/ncs:name";  
307   }  
308 }  
309  
310 // may replace this with other ways of referring to the devices.  
311 leaf-list device {  
312   type leafref {  
313     path "/ncs:devices/ncs:device/ncs:name";  
314   }  
315 }  
316  
317 // may replace this with other ways of referring to the devices.  
318 leaf-list device {  
319   type leafref {  
320     path "/ncs:devices/ncs:device/ncs:name";  
321   }  
322 }  
323  
324 // may replace this with other ways of referring to the devices.  
325 leaf-list device {  
326   type leafref {  
327     path "/ncs:devices/ncs:device/ncs:name";  
328   }  
329 }  
330  
331 // may replace this with other ways of referring to the devices.  
332 leaf-list device {  
333   type leafref {  
334     path "/ncs:devices/ncs:device/ncs:name";  
335   }  
336 }  
337  
338 // may replace this with other ways of referring to the devices.  
339 leaf-list device {  
340   type leafref {  
341     path "/ncs:devices/ncs:device/ncs:name";  
342   }  
343 }  
344  
345 // may replace this with other ways of referring to the devices.  
346 leaf-list device {  
347   type leafref {  
348     path "/ncs:devices/ncs:device/ncs:name";  
349   }  
350 }  
351  
352 // may replace this with other ways of referring to the devices.  
353 leaf-list device {  
354   type leafref {  
355     path "/ncs:devices/ncs:device/ncs:name";  
356   }  
357 }  
358  
359 // may replace this with other ways of referring to the devices.  
360 leaf-list device {  
361   type leafref {  
362     path "/ncs:devices/ncs:device/ncs:name";  
363   }  
364 }  
365  
366 // may replace this with other ways of referring to the devices.  
367 leaf-list device {  
368   type leafref {  
369     path "/ncs:devices/ncs:device/ncs:name";  
370   }  
371 }  
372  
373 // may replace this with other ways of referring to the devices.  
374 leaf-list device {  
375   type leafref {  
376     path "/ncs:devices/ncs:device/ncs:name";  
377   }  
378 }  
379  
380 // may replace this with other ways of referring to the devices.  
381 leaf-list device {  
382   type leafref {  
383     path "/ncs:devices/ncs:device/ncs:name";  
384   }  
385 }  
386  
387 // may replace this with other ways of referring to the devices.  
388 leaf-list device {  
389   type leafref {  
390     path "/ncs:devices/ncs:device/ncs:name";  
391   }  
392 }  
393  
394 // may replace this with other ways of referring to the devices.  
395 leaf-list device {  
396   type leafref {  
397     path "/ncs:devices/ncs:device/ncs:name";  
398   }  
399 }  
400  
401 // may replace this with other ways of referring to the devices.  
402 leaf-list device {  
403   type leafref {  
404     path "/ncs:devices/ncs:device/ncs:name";  
405   }  
406 }  
407  
408 // may replace this with other ways of referring to the devices.  
409 leaf-list device {  
410   type leafref {  
411     path "/ncs:devices/ncs:device/ncs:name";  
412   }  
413 }  
414  
415 // may replace this with other ways of referring to the devices.  
416 leaf-list device {  
417   type leafref {  
418     path "/ncs:devices/ncs:device/ncs:name";  
419   }  
420 }  
421  
422 // may replace this with other ways of referring to the devices.  
423 leaf-list device {  
424   type leafref {  
425     path "/ncs:devices/ncs:device/ncs:name";  
426   }  
427 }  
428  
429 // may replace this with other ways of referring to the devices.  
430 leaf-list device {  
431   type leafref {  
432     path "/ncs:devices/ncs:device/ncs:name";  
433   }  
434 }  
435  
436 // may replace this with other ways of referring to the devices.  
437 leaf-list device {  
438   type leafref {  
439     path "/ncs:devices/ncs:device/ncs:name";  
440   }  
441 }  
442  
443 // may replace this with other ways of referring to the devices.  
444 leaf-list device {  
445   type leafref {  
446     path "/ncs:devices/ncs:device/ncs:name";  
447   }  
448 }  
449  
450 // may replace this with other ways of referring to the devices.  
451 leaf-list device {  
452   type leafref {  
453     path "/ncs:devices/ncs:device/ncs:name";  
454   }  
455 }  
456  
457 // may replace this with other ways of referring to the devices.  
458 leaf-list device {  
459   type leafref {  
460     path "/ncs:devices/ncs:device/ncs:name";  
461   }  
462 }  
463  
464 // may replace this with other ways of referring to the devices.  
465 leaf-list device {  
466   type leafref {  
467     path "/ncs:devices/ncs:device/ncs:name";  
468   }  
469 }  
470  
471 // may replace this with other ways of referring to the devices.  
472 leaf-list device {  
473   type leafref {  
474     path "/ncs:devices/ncs:device/ncs:name";  
475   }  
476 }  
477  
478 // may replace this with other ways of referring to the devices.  
479 leaf-list device {  
480   type leafref {  
481     path "/ncs:devices/ncs:device/ncs:name";  
482   }  
483 }  
484  
485 // may replace this with other ways of referring to the devices.  
486 leaf-list device {  
487   type leafref {  
488     path "/ncs:devices/ncs:device/ncs:name";  
489   }  
490 }  
491  
492 // may replace this with other ways of referring to the devices.  
493 leaf-list device {  
494   type leafref {  
495     path "/ncs:devices/ncs:device/ncs:name";  
496   }  
497 }  
498  
499 // may replace this with other ways of referring to the devices.  
500 leaf-list device {  
501   type leafref {  
502     path "/ncs:devices/ncs:device/ncs:name";  
503   }  
504 }
```

Make YANG Model -- Example

Inside the YANG model, we want to add nodes to represent our input parameters.

We have:

- Device list

- Region (3 Options)

- Data-center for the server

The Device list is actually created for us by the skeleton:

It is a YANG “leaf-list” ie, a list of single value leafs of type “leafref”

A leafref, is a leaf that references a separate YANG node in NSO, in this case our NSO devices!

This means we can pick from a list of available devices inside the NSO

```
// may replace this with other ways of referring to the devices.  
leaf-list device {  
    type leafref {  
        path "/ncs:devices/ncs:device/ncs:name";  
    }  
}
```

Make YANG Model -- Example

We now need to add a node for our Region input parameters.

We want it to only allow 3 possible options, AMER, APJC & EMEAR.

We can use the YANG enumeration type to do this:

It is a YANG leaf of type enumeration, with our 3 possible options!

```
// replace with your own stuff here
leaf Region {
    type enumeration{
        enum "AMER";
        enum "APJC";
        enum "EMEAR";
    }
}
```

Make YANG Model -- Example

Finally we want a simple node to store what DC the server is in.
Lets just have it as an open string.

```
leaf Data-Center{  
    type string;  
}
```

Make YANG Model -- Example

An annoyance / step due to a bug...

The skeleton creation script for NSO 4.4 includes a bug where it is missing “augment /ncs:services { list your_service_name {} }

This is crucial to add for your service to work.

```
augment /ncs:services {  
    list dhcp {  
        key name;
```

Make YANG Model -- Example

Now, lets validate the model with pyang:

```
[BRANBLAC-M-W0WN:packages branblac$ cd dhcp/src/yang/
[BRANBLAC-M-W0WN:yang branblac$ pyang dhcp.yang
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:22: warning: imported module tailf-ncs-monitoring not used
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1323: error: node tailf-ncs::connect is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1333: error: node tailf-ncs::sync-to is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1343: error: node tailf-ncs::sync-from is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1353: error: node tailf-ncs::disconnect is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1363: error: node tailf-ncs::check-sync is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1373: error: node tailf-ncs::check-yang-modules is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:1384: error: node tailf-ncs::fetch-ssh-host-keys is not found
/Users/branblac/ncs-4.4/src/ncs.yang/tailf-ncs-devices.yang:3278: error: node tailf-ncs::disconnect is not found
BRANBLAC-M-W0WN:yang branblac$ █
```

Ignoring the /tailf-ncs-devices.yang errors (bu issue) we do not see any errors related to our YANG module.

Meaning our model is sound! (at least in syntax)

Make XML Template

- We now need to templatize the configuration we want deployed inside our service model!
- The easiest way to do this is to have NSO do the hard work for us!
 - Of course we can try to convert CLI to XML in our heads if wanted...
- XML from CLI Process:
 - Configure a lab/virtual or get config sample from existing devices
 - Sync the device to NSO
 - Get the Parsed XML config from NSO!

Getting XML from NSO devices

NSO CLI:

```
show running-config devices device  
(device-name) config | display xml
```

Will print the full config to the screen

Pick and choose the config you want!

REST API (My preference):

GET:

```
NsoHostName.cisco.com/api/running/device  
s/device/(device-name)/config?deep
```

Will return the full config (may be big!) so
we can filter it down via rest paths

Pick and choose the config you want!

Postman for getting XML config

The screenshot shows the Postman application interface. The top navigation bar includes 'Runner', 'Import', 'Builder' (which is selected), 'Team Library', and user information. The left sidebar shows a history of requests grouped by date: June 28, June 26, and June 22. The main workspace displays a 'GET' request to 'http://localhost:8080/api/running/devices/device/demo-0?deep'. The 'Headers' tab shows several header fields: Accept (application/vnd.yang.data+json), Content-Type (application/vnd.yang.data+xml), X-Cisco-Meraki-API-Key (9cd412e906cdaa067b59afdc143ce1c625e174c5), and three additional Content-Type entries. The 'Body' tab is selected, showing the XML response in 'Pretty' format. The XML content includes nodes like <radius>, <server>, <id>, <address>, <ipv4>, <host>, <auth-port>, <acct-port>, <backoff>, <exponential>, and <backoff>. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 491 ms'.

localhost:8080/api/running/ http://localhost:8080/ +

GET http://localhost:8080/api/running/devices/device/demo-0?deep Params Send Save

Accept application/vnd.yang.data+json
Content-Type application/vnd.yang.data+xml
X-Cisco-Meraki-API-Key 9cd412e906cdaa067b59afdc143ce1c625e174c5
Content-Type application/json
Content-Type application/vnd.yang.data+json

New key Value Description

Body Cookies Headers (8) Tests Status: 200 OK Time: 491 ms

Pretty Raw Preview XML ↗

```
150      </bpdu>
151      </optimize>
152      </spanning-tree>
153      <radius xmlns="urn:ios">
154          <server>
155              <id>one</id>
156              <address>
157                  <ipv4>
158                      <host>10.0.0.1</host>
159                      <auth-port>1812</auth-port>
160                      <acct-port>1813</acct-port>
161                  </ipv4>
162              </address>
163              <backoff>
164                  <exponential>
165          </exponential>
166          <backoff>
167      </server>
168  </radius>
```

June 28

POST localhost:8080/api/running/Audits/IPv6/igmp/_operations/audit
POST localhost:8080/api/running/Audits/IPv6/igmp/_operations/audit
POST localhost:8080/api/running/Audits/IPv6/igmp/_operations/audit

June 26

GET http://localhost:8080/api/running/devices/device/svl-gem-joe-asa-fw1.cisco.com?deep
GET http://localhost:8080/api/running/devices/device
GET http://localhost:8080/api/running/devices
GET http://localhost:8080/api/running/devices

June 22

POST localhost:8080/api/running/Audits/IPv6/igmp/_operations/audit
POST localhost:8080/api/running/Audits/IPv6/igmp/_operations/audit

© 2017 Cisco

Postman for getting filtered XML config

The screenshot shows the Postman application interface in 'Builder' mode. The top navigation bar includes 'Runner', 'Import', 'Builder' (which is highlighted), 'Team Library', and user information ('Brandon B...'). The left sidebar displays a timeline of requests, categorized by date: June 28, June 26, and June 22. The main workspace shows a 'GET' request to 'http://localhost:8080/api/running/devices/device/demo-0/config/radius?deep'. The 'Headers' tab is selected, showing the following configuration:

Header	Description
Accept	application/vnd.yang.data+json
Content-Type	application/vnd.yang.data+xml
X-Cisco-Meraki-API-Key	9cd412e906cdaa067b59afdc143ce1c625e174c5
Content-Type	application/json
Content-Type	application/vnd.yang.data+json
New key	Value

The 'Body' tab is also visible, showing the XML response structure. The response status is '200 OK' and the time taken is '137 ms'. The XML content is displayed in a code editor-like view with line numbers:

```
1 <radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest" xmlns:ios="urn:ios" xmlns:ncs="http://tail-f.com/ns/ncs">
2   <server>
3     <id>one</id>
4     <address>
5       <ipv4>
6         <host>10.0.0.1</host>
7         <auth-port>1812</auth-port>
8         <acct-port>1813</acct-port>
9       </ipv4>
10      </address>
11      <backoff>
12        <exponential>
13          </exponential>
14        </backoff>
15      </server>
16    </radius>
```

Make XML Template

- Take the XML and Copy Paste it into our service skeleton's xml file!
- Once we have the overall XML from NSO, we need to modify it for the variables.
- Within the XML/YANG we can do a lot of fancy stuff from iterations, dependencies, references, look up etc.
- For our purposes we want to keep the templates are clean and simple as we can.
- If complex logic is desired, Python service logic is recommended

Input Parameters in XML

- Accessing our input parameters in the XML template is easy!
- All types defined in our YANG are available through the structure:
 - {/input_variable_name}
- When we use the variables in the format above, NSO will replace the variable with the value we input into the model for that variable!

Make XML Template -- example

Copy & Paste the XML config
into the template file:

```
dhcp-template.xml •
6      Select the devices from some data structure in the service
7      model. In this skeleton the devices are specified in a leaf-list.
8      Select all devices in that leaf-list:
9
10     <!--
11     <name>{/device}</name>
12     <config>
13         <!--
14         Add device-specific parameters here.
15         In this skeleton the service has a leaf "dummy"; use that
16         to set something on the device e.g.:
17         <ip-address-on-device>{/dummy}</ip-address-on-device>
18     <!--
19     <radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest" xmlns:ios="urn:ios" xmlns:ncs="http://tail-f.com/ns/ncs">
20         <server>
21             <id>one</id>
22             <address>
23                 <ipv4>
24                     <host>10.0.0.1</host>
25                     <auth-port>1812</auth-port>
26                     <acct-port>1813</acct-port>
27                 </ipv4>
28             </address>
29             <backoff>
30                 <exponential>
31                     </exponential>
32             </backoff>
33         </server>
34     </radius>
35     </config>
36 </device>
```

Make XML Template -- example

For this simple example, we will use YANG “When” statements as IF statements to determine Radius server IP by region.

When works by applying the tags underneath the tag it is in, only when a specific condition is met.

In our case, per region!

```
<radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest" xmlns:ios="urn:ios" xmlns:ncs="http://tail-f.com/ns/ncs">
  <server>
    <id>one</id>
    <address when="{/Region='AMER'}">
      <ipv4>
        <host>10.0.0.1</host>
        <auth-port>1812</auth-port>
        <acct-port>1813</acct-port>
      </ipv4>
    </address>
    <backoff>
      <exponential>
    </exponential>
    </backoff>
  </server>
</radius>
```

Make XML Template -- example

And repeat per region!

As you can tell, we could do a lot more, with a lot less work if this mapping was done via Python!

Don't worry, we'll get there!

```
<!-- One -->
<address when="/Region='AMER'>
  <ipv4>
    <host>10.0.0.1</host>
    <auth-port>1812</auth-port>
    <acct-port>1813</acct-port>
  </ipv4>
</address>
<address when="/Region='APJC'>
  <ipv4>
    <host>10.0.2.1</host>
    <auth-port>1812</auth-port>
    <acct-port>1813</acct-port>
  </ipv4>
</address>
<address when="/Region='EMEAR'>
  <ipv4>
    <host>10.0.3s.1</host>
    <auth-port>1812</auth-port>
    <acct-port>1813</acct-port>
  </ipv4>
</address>
```

A quick distraction... CLI to XML

- While it may seem intimidating or a lot, the XML structure is actually almost the same thing as normal CLI! Just structured.
- If we look at our Radius Server CLI vs XML for example

radius server one

 address ipv4 10.0.0.1 auth-port 1812 acct-port 1813

- It follows a pattern. The address belongs to “radius” ID of server “one”
- Then, for that server, we have an address that has attributes of “ipv4” with ip of 10.0.0.1, a auth-port with value of 1812 and an acct-port with a value of 1813
- The XML we have is saying the exact same thing, just in a easier to machine process format!

Side by Side

radius server one

address ipv4 10.0.0.1 auth-port 1812 acct-port
1813

```
<radius>
  <server>
    <id>one</id>
    <address>
      <ipv4>
        <host>10.0.0.1</host>
        <auth-port>1812</auth-port>
        <acct-port>1813</acct-port>
      </ipv4>
    </address>
  </server>
</radius>
```

Test!

- We are now ready to compile and load our package!
- We can then test to ensure the service is deploying properly!
- To compile:
 - On the CLI, cd to the package directory's src folder and type: make
- To reload
 - Enter the nso cli and type “packages reload”
- We're now ready to test it out! (provided it compiled and reloaded ☺)

Test!

- Easiest way to visually see what the service is doing is via the WebUI
- CLI is faster but harder to visualize whats happening (preferences!)
- Pre-defined Python or Bash test scripts are the best and can be auto-run by a CI pipeline for automated testing on build!
- To test, either apply against virtual and/or lab devices and validate that pushes can be made with out issue. Try different corner cases and oddities.
- Get peer review
- Compare the devices CLI/NSO output CLI to your earlier defined CLI

Test! --example

- From UI, navigate to the service and add a device and pick a region

The screenshot shows the NSO 4.4 web interface. The browser tab is titled "localhost:8080/index.html#/model/ncs:services/demo:demo%7Bdummy%7D". The top navigation bar includes links for "Commit", "Views", "Jobs", "Alarms" (with 6 notifications), and "admin". The URL in the address bar is "/ncs:services / demo:demo { dummy }".

The main content area displays the configuration for the "dummy" service. It includes tabs for "demo", "modified", "directly-modified", "commit-queue", and "log".

Configuration details:

- INFO name**: A text input field containing "dummy".
- Region**: A dropdown menu set to "AMER".
- device-list**: A panel showing "No items to display".
- used-by-customer-service**: A panel showing "No items to display".
- device**: A panel containing a list box with "demo-0" selected, and a "+" button to the right.
- Actions**: Buttons for "check-sync...", "deep-check-sync...", "re-deploy...", "reactive-re-deploy...", "touch...", "get-modifications...", and "un-deploy...".

Test! --example

- Now lets “Commit dry-run native” to see what CLI would be pushed
- As we can see, the CLI is the same as what we defined in our requirements for AMER!
- Now try with all regions/permutations ☺

```
✖ Native Commit Dry Run
-----
Search with regular expression...
-----
▼ demo-0
radius server one
address ipv4 10.0.0.1 auth-port 1812 acct-port 1813
backoff exponential
!
```

Summary

- This is a simplistic example, but highlights the starting point of how to develop a service
- Most Service will be significantly more advanced but follow the same development process
- YANG -> Define the input parameters
- XML -> Configuration template
- Map the YANG inputs to the XML variables

Q&A Before Lab?

Feedback - Survey

Agenda

- PYTHON REVIEW
- PYTHON API INTRODUCTION
- MAAGIC (with Labs)
- LUNCH
- Services with Code
- LAB: DMZ Provisioning

Python Review

NSO Python API

Recap: Python Automation with Unstructured Data

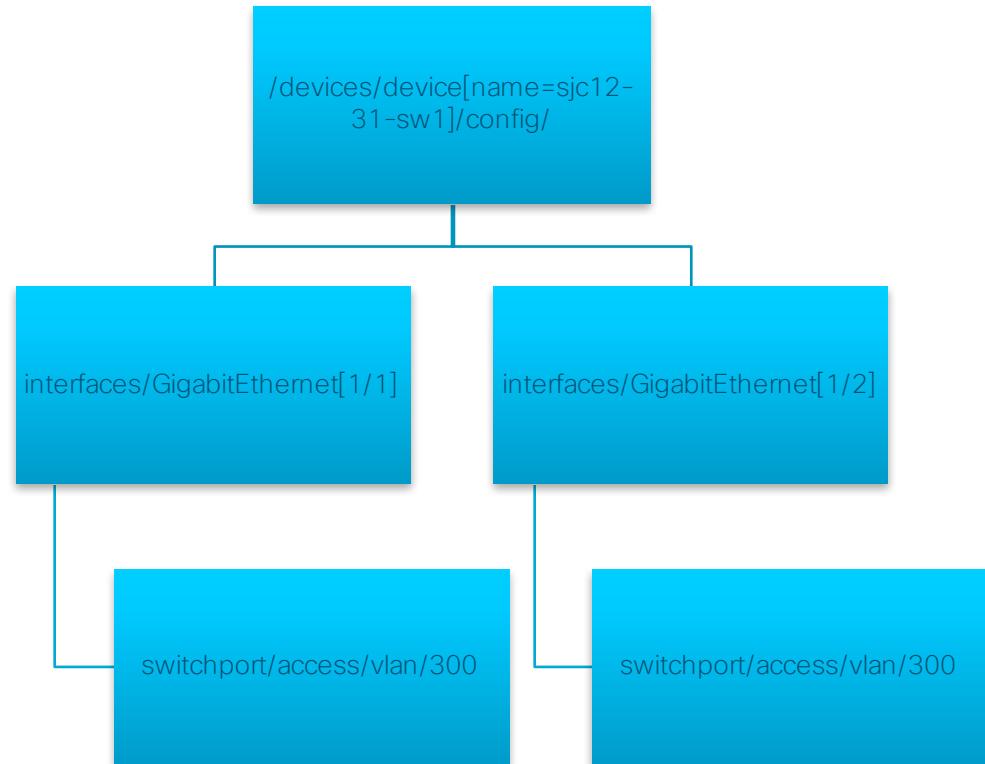
Current network automation within Cisco Network IT involves manipulating unstructured data through the Cisco CLI

```
import netmiko  
  
ssh_con = netmiko.ConnectHandler(device_type='cisco_ios', ip="10.155.8.4", username='user.web',  
password='webpassword'  
ssh_con.find_prompt()  
output = ssh_con.send_command("show running-config")
```

Use ciscoconfparse, TextFSM, or regular expressions to parse running config

Recap: Structured Data

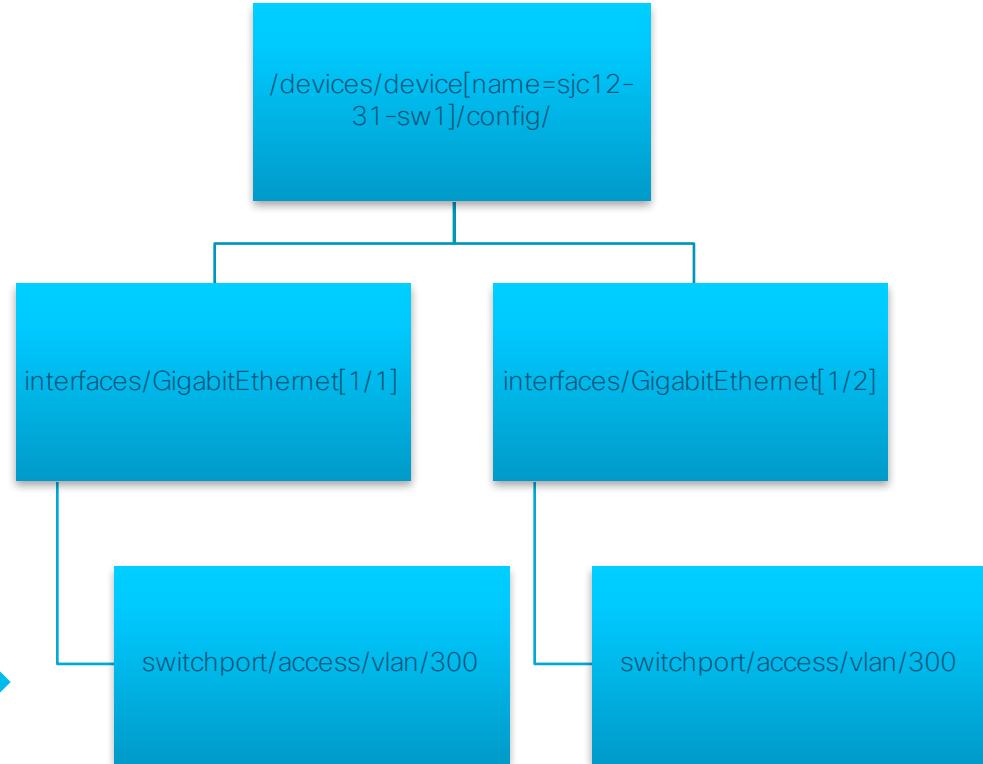
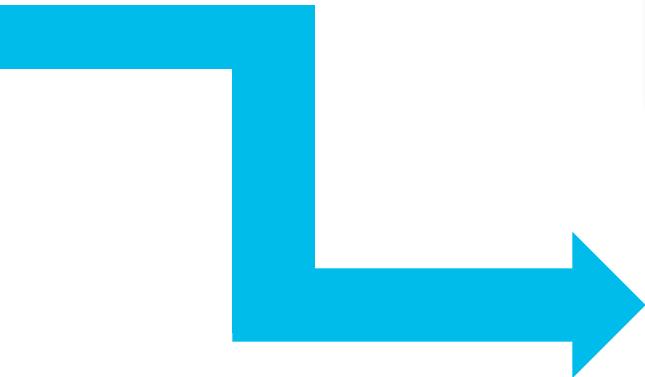
- YANG defines the hierarchy and the available parameters
- XML provides the actual data
- Through YANG and XML, Cisco CLI can be represented in a structured format



Python, YANG, and XML



NSO Python API

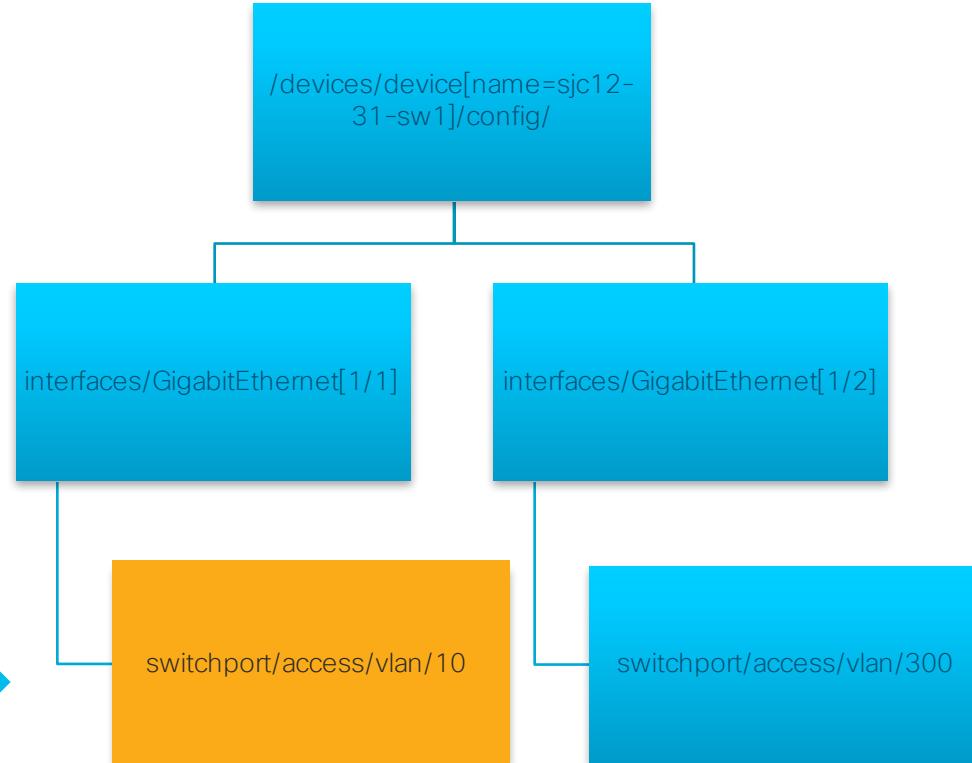
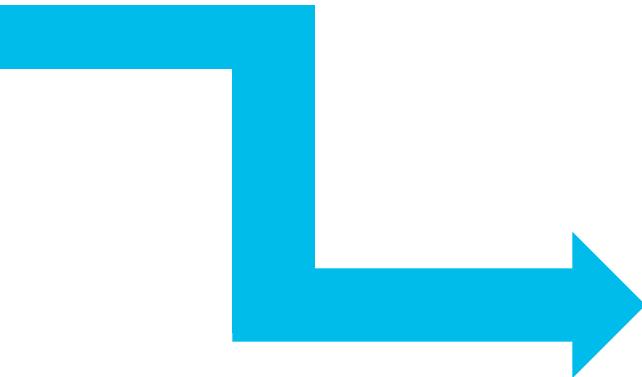


NSO Python API traverses the YANG tree and manipulates the data inside the XML

Python, YANG, and XML



NSO Python API

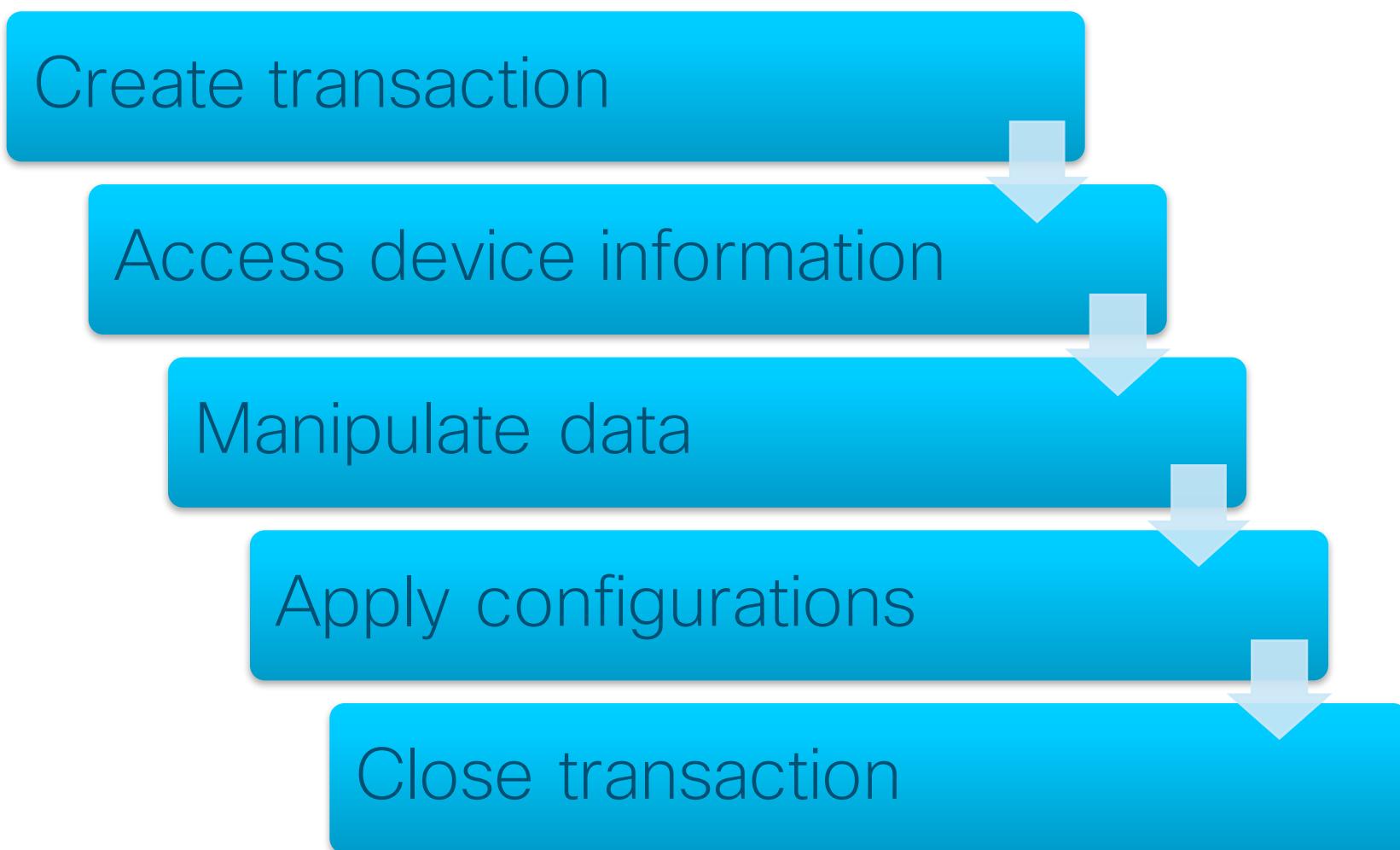


NSO Python API traverses the YANG tree and manipulates the data inside the XML

ncs Library

- When you install NSO local or system install, the ncs library is automatically installed
- Called by doing an import ncs
- Source code is located at
 - Local install: /Users/<username>/ncs-<version-number>/src/ncs/pyapi/pysrc
 - System install: /apps/nso/opt/ncs/<ncs-version-number>/src/ncs/pyapi/pysrc
- Python interpreter reference
 - >>import ncs
 - >>help(ncs.maapi)
 - >>help(ncs.maagic)

NSO Python Transactions



NSO Python Example

```
import ncs

with ncs.maapi.single_write_trans('admin', 'python') as trans:
    root = ncs.maagic.get_root(trans)
    device = root.devices.device["sjc12-31-sw1.cisco.com"]
    interface = device.config.ios_interface.GigabitEthernet["1/13"]
    print(interface.switchport.access.vlan) # 330
    interface.switchport.access.vlan = 240
    trans.apply()
```

NSO Python Example

```
import ncs

with ncs.maapi.single_write_trans('admin', 'python') as trans:

    root = ncs.maagic.get_root(trans)

    device = root.devices.device["sjc12-31-sw1.cisco.com"]

    interface = device.config.ios_interface.GigabitEthernet["1/13"]

    print(interface.switchport.access.vlan) # 330

    interface.switchport.access.vlan = 240

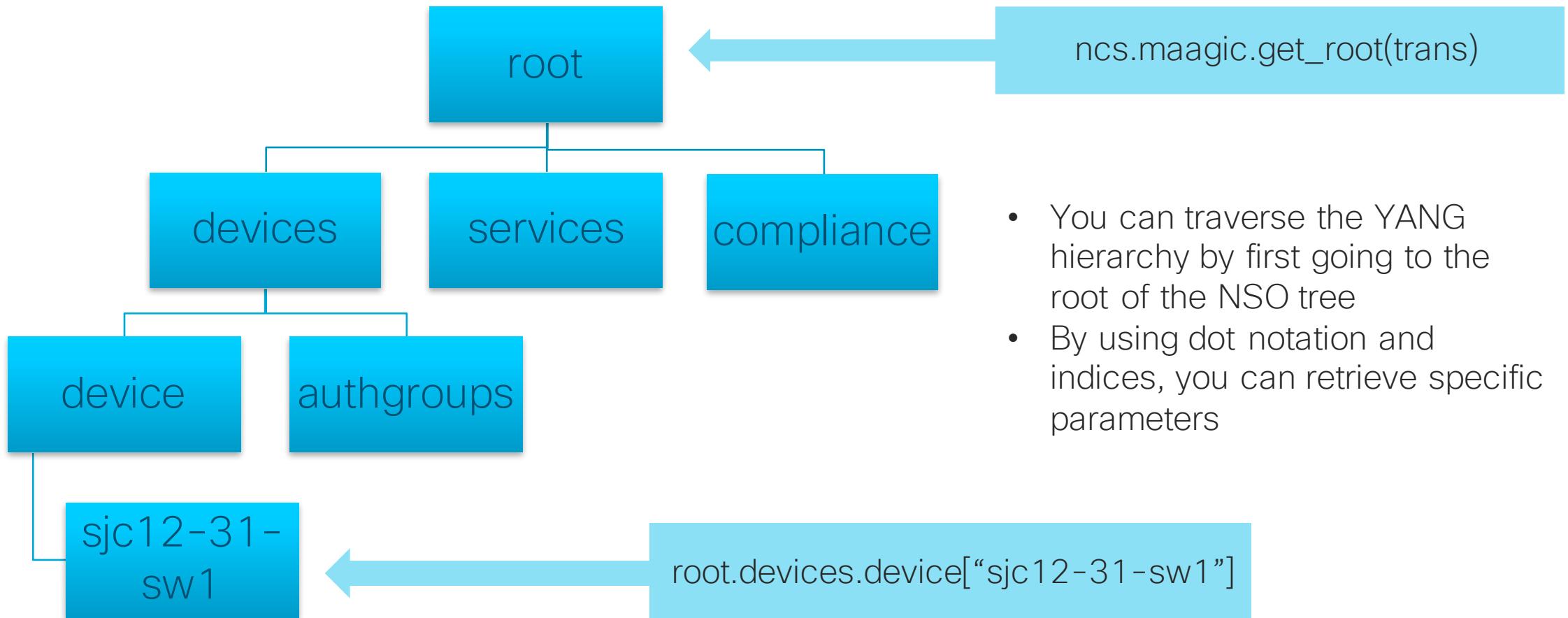
    trans.apply()
```

NSO Python Example

```
import ncs

with ncs.maapi.single_write_trans('admin', 'python') as trans:
    root = ncs.maagic.get_root(trans)
    device = root.devices.device["sjc12-31-sw1.cisco.com"]
    interface = device.config.ios_interface.GigabitEthernet["1/13"]
    print(interface.switchport.access.vlan) # 330
    interface.switchport.access.vlan = 240
    trans.apply()
```

YANG Traversal through Python API



NSO Python Example

```
import ncs

with ncs.maapi.single_write_trans('admin', 'python') as trans:

    root = ncs.maagic.get_root(trans)

    device = root.devices.device["sjc12-31-sw1.cisco.com"]

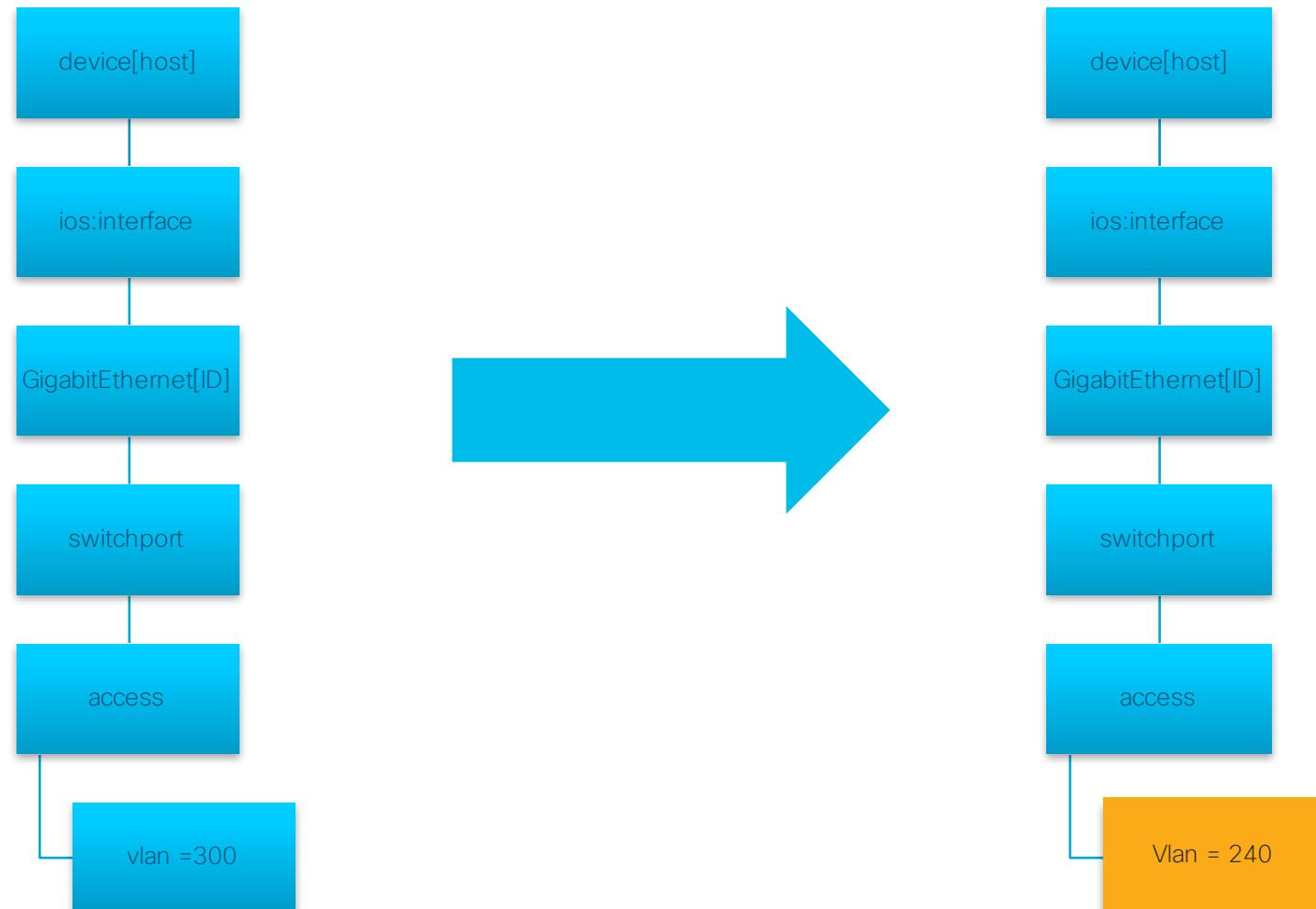
    interface = device.config.ios_interface.GigabitEthernet["1/13"]

    print(interface.switchport.access.vlan) # 330

    interface.switchport.access.vlan = 240

    trans.apply()
```

Configuration Hierarchy



Lab 1

Creating a Session

- This is an example of how to create a session into NSO.
- A sessions allows for reading data from NSO and executing Actions. **It does not create a transaction into NSO.**

with ncs.maapi.Maapi() as m:

```
    with ncs.maapi.Session(m, 'admin', 'python', groups=['ncsadmin']):  
        root = ncs.maagic.get_root(m)
```

Creating a Transaction

- This is an example of how to create a transaction into NSO.
- We can create a transaction using either:
 - with ncs.maapi.Maapi() as m:
 - with ncs.maapi.Session(m, 'admin', 'python'):
 - with m.start_write_trans() as t:
- Or
 - with ncs.maapi.single_write_trans('admin', 'python', groups=['admin']) as t:
- commit the transaction with the apply() method inside the transaction object t, we created above. i.e t.apply()

NSO Python Read-Only

```
import ncs

with ncs.maapi.single_read_trans('admin', 'python') as trans:
    root = ncs.maagic.get_root(trans)
    device = root.devices.device["sjc12-31-sw1.cisco.com"]
    interface = device.config.ios_interface.GigabitEthernet["1/13"]
    print(interface.switchport.access.vlan) # 330
```

Navigating with Maagic

- Example of how to understand and navigate a devices config in the python API.
- This example will show by printing the directory of different levels of the config with ncs.maapi.Maapi() as m:

```
with ncs.maapi.Session(m, 'admin', 'python', groups=['ncsadmin']):  
    root = ncs.maagic.get_root(m)  
    device_config = root.devices.device[device_name].config  
    print(dir(device_config))  
    print(dir(device_config.ip))  
    print(dir(device_config.ip.dhcp))  
    print(dir(device_config.ip.dhcp.snooping))
```

Lab 2

Changing a Value

- If you are just changing a leaf value you can use the assignment operator.
`device.config.hostname = "new_host_name"`
- But if you are changing a list value you may have to delete an old value first and create a new value
`del root.devices.device.config.interface.vlan["200"]`
`root.devices.device.config.interface.vlan.create("200")`

Deleting a Value

- If you want to delete a value using the python api, you simply run del and pass the object you want to delete.

```
del root.devices.device.config.interface.vlan["200"]
```

Lab 3

Iterating through a list

- Using For loops allows the maagic api to iterate through all the objects in a given object path.

```
for acl in root.devices.device[box.name].config.ip.access_list.extended.ext_named_acl:  
    print(acl.name)
```

Lab 4

CLI Commands...

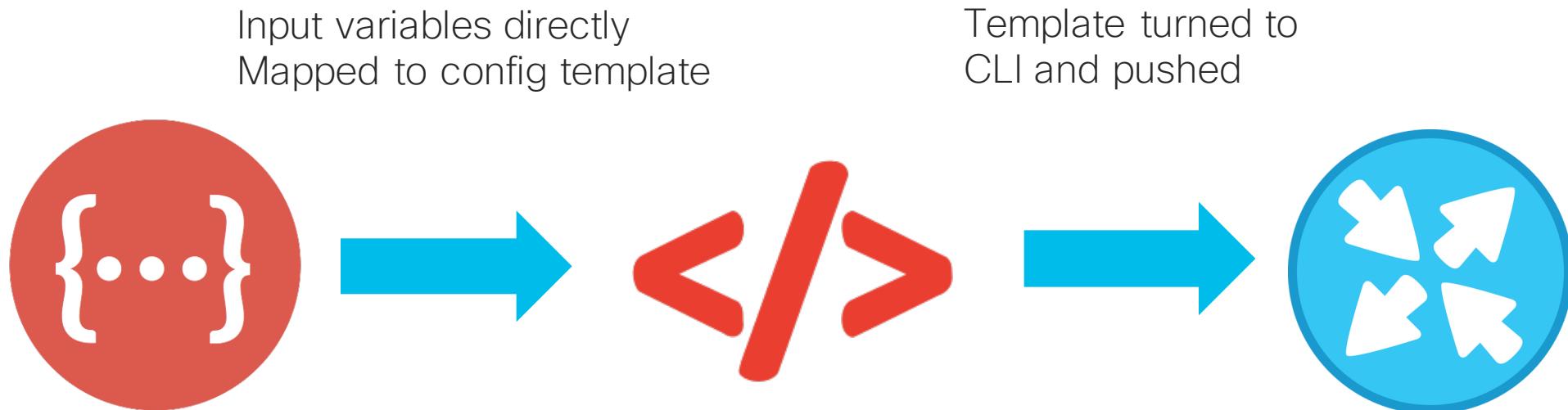
```
with ncs.maapi.Maapi() as m:  
    with ncs.maapi.Session(m, 'admin', 'python'):  
        root = ncs.maagic.get_root(m)  
        device = root.devices.device[device_name]  
        input1 = device.live_status.ios_stats__exec.show.get_input()  
        input1.args = [command]  
        output = device.live_status.ios_stats__exec.show(input1).result  
        output = device.live_status.ios_stats__exec.any(input1).result  
        print(output)
```

Lab 5

Services with Code

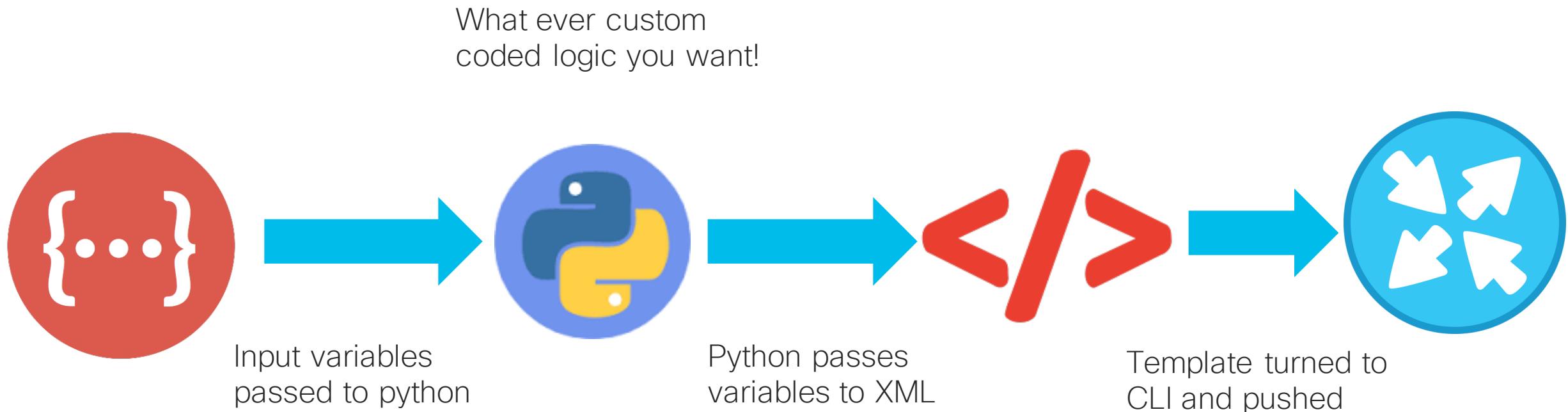
Developing Services with Code

Services with just templates:



Developing Services with Code

Services with code:



Services with Code

- NSO enables us to use several different languages for building packages, however Python is the de-facto standard for network automation programming
- As a result, all packages should be made with Python to allow for easy reading, improvement and maintenance by the rest of the network team

Developing a Python Service

- Gather Requirements
- Determine appropriate input parameters
- Determine Configuration (CLI or XML)
- Map input parameters to configuration
- Make YANG Model
- Make Python code
- Make XML Template
- Test!



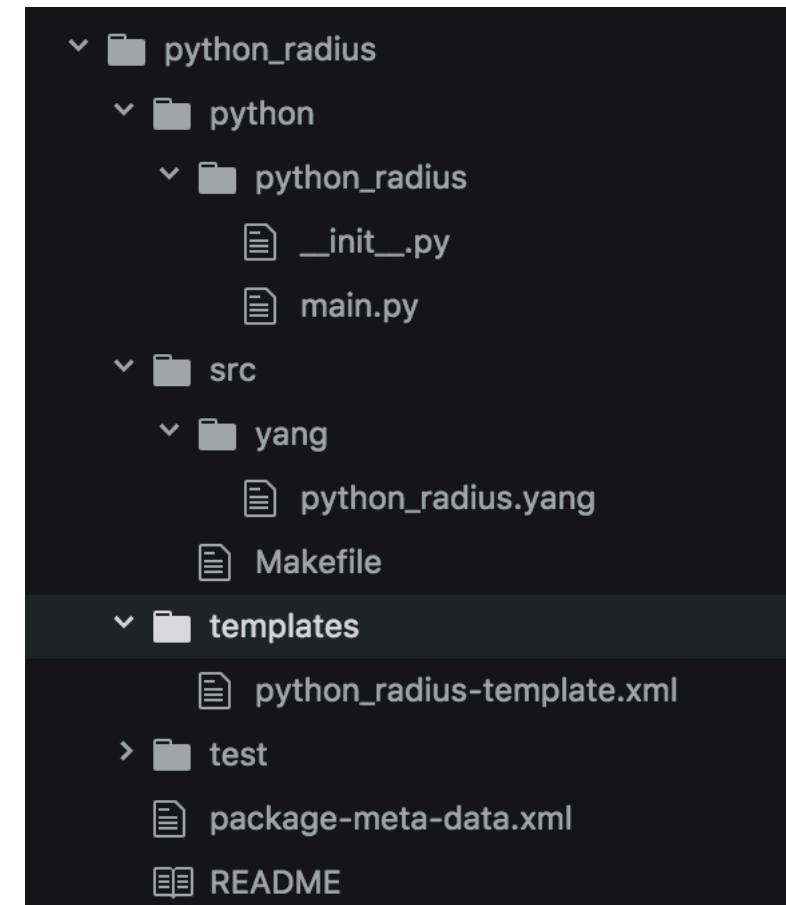
We're gonna jump
in right here!

Learn by doing!

- Lets build upon our earlier example! Lets walk through adding custom logic to our radius service

Python Service Directory Structure

- Same as our template but with a python directory
- We can add other modules and files to this directory for importing into our code as well
- Made using:
- ncs-make-package -- service-skeleton python-and-template python_radius



Make python code

- Open the main.py file in our python directory
- This is a pre-built scaffold for us
- The scaffold provides some simple examples of how it works:

```
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

        vars = ncs.template.Variables()
        vars.add('DUMMY', '127.0.0.1')
        template = ncs.template.Template(service)
        template.apply('python_radius-template', vars)
```

Make python code

- When the service is executed from NSO, the cb_create method in the ServiceCallbacks class is called.
- As a result, all code we put into the cb_create method, will get executed whenever the service is created or modified
- **IMPORTANT!** Remember, this code is always executed! Keep this in mind when developing services with external systems. (beyond the scope of this class)

Make python code

- The cb_create method gets passed a couple useful inputs
 - Root – this is a root object into NSO for interacting with the cDB
 - Service – this is a python object with the information that was passed into the service model. Think of it as an object representing the inputted parameters. It's a python representation of our YANG model

Make python code

- Interacting with the template:
- The scaffold shows us how to use the template
 - We create an object to store of input paramater values (vars = ncs.template.Variables())
 - We add key-value pairs to the vars object (vars.add('DUMMY', '127.0.0.1'))
 - These are the names and values that are passed to the template
 - We then create a template object (template = ncs.template.Template(service))
 - Then specify which template and pass the values into it!
template.apply('python_radius-template', vars)
 - This is then what modifies the device (we can also modify it in the python if desired)

```
def cb_create(self, tctx, root, service, proplist):
    self.log.info('Service create(service=', service._path, ')')

    vars = ncs.template.Variables()
    vars.add('DUMMY', '127.0.0.1')
    template = ncs.template.Template(service)
    template.apply('python_radius-template', vars)
```

Make python code

- Now we need to modify the python for our service
- Determine the logic needed for defining the values to be inputted into the XML template
- This logic maybe things like incrementing Ip addresses, mapping devices to region configs, or getting config based on a specific input
- Then assign the new values to the variable object and pass to the template

Make python code

- A note on de-bugging and record keeping
- Since the code runs on the server by the server, we need to use logs to get troubleshooting, de-bugging and accountability information
- The NSO provides us an API to do this in a consistent fashion.
- The “self” parameter, has a log method for us to use:
 - `self.log.info('our log message here!')`

Make python code --example

- For our radius server example, lets improve the service by determining the ip address needed in Python rather than by mapping it in the template
- To do this, lets use a simple If-else block to map the region to an Ip-address

```
@Service.create
def cb_create(self, tctx, root, service, proplist):
    self.log.info('Service create(service=', service._path, ')')
    vars = ncs.template.Variables()
    if service.Region == "AMER":
        vars.add('ip_address', "10.0.1.1")
    elif service.Region == "APJC":
        vars.add('ip_address', "10.0.2.1")
    elif service.Region == "APJC":|
        vars.add('ip_address', "10.0.3.1")
    template = ncs.template.Template(service)
    template.apply('python_radius-template', vars)
```

Make XML Template

- Now that we have our logic in python we are ready to modify our XML template for the updates
- The key difference in the template (aside from design) will be how the variables are referenced
- Rather than {/variable_from_yang} we use {\$variable_from_python}
- The name being the name we created in the python file

Make XML Template -- example

- We have removed our when statements and mappings. Instead we now just use the ip_address from python!

```
<name>/device</name>
<config>
    <!--
        Add device-specific parameters here.
        In this skeleton the, java code sets a variable DUMMY, use it
        to set something on the device e.g.:
        <ip-address-on-device>{$DUMMY}</ip-address-on-device>
    -->
    <radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest" xmlns:ios="<!--
        <server>
            <id>one</id>
            <address>
                <ipv4>
                    <host>{$ip_address}</host>
                    <auth-port>1812</auth-port>
                    <acct-port>1813</acct-port>
                </ipv4>
            </address>
            <backoff>
                <exponential>
            </exponential>
        </backoff>
    </server>
</radius>
</config>
<!--
```

Test!

- For fun, we can also run pylint on our service code to check our code for standard issues/ best practices
- We are now ready to test our service and make sure everything is working as expected!
- Nothing changes here when compared to a template service

Test! -- example

Issue a commit dry-run native to see the CLI that would be pushed!

The screenshot shows the NSO 4.4 web interface at `localhost:8080/index.html#/model/ncs%3Aservices/python_radius%3Apython_radius%7B%22demo%22%7D`. The top navigation bar includes links for Cisco, Commit (highlighted in green), Views, Jobs, Alarms (with 7 notifications), admin, and a menu icon. The main content area displays a "Native Commit Dry Run" section with a search bar and a tree view under "/ncs:services". The tree shows a node for "ios-0" which contains the following CLI configuration:

```
radius server one
address ipv4 10.0.1.1 auth-port 1812 acct-port 1813
backoff exponential
!
```

On the left side, there are three vertical tabs: "python_radius" (selected), "INFO", and "name". The "INFO" tab shows "demo" as the value. The "Region" tab shows "AMER". The "reactive-re-deploy" tab has a gear icon.

Test! -- example

Pylint score:

```
BRANBLAC-M-W0WN:python_radius branblac$ cd python/python_radius/
BRANBLAC-M-W0WN:python_radius branblac$ ls
__init__.py      main.py
BRANBLAC-M-W0WN:python_radius branblac$ pylint main.py
No config file found, using default configuration
*****
Module python_radius.main
C: 16, 0: Trailing whitespace (trailing-whitespace)
C: 1, 0: Missing module docstring (missing-docstring)
C: 9, 0: Missing class docstring (missing-docstring)
W: 17, 8: Redefining built-in 'vars' (redefined-builtin)
C: 14, 4: Missing method docstring (missing-docstring)
W: 15,49: Access to a protected member _path of a client class (protected-access)
C: 52, 0: Missing class docstring (missing-docstring)

-----
Your code has been rated at 6.50/10
```

Pylint score after updating:

```
BRANBLAC-M-W0WN:python_radius branblac$ pylint main.py
No config file found, using default configuration
*****
Module python_radius.main
C: 1, 0: Missing module docstring (missing-docstring)
W: 18,49: Access to a protected member _path of a client class (protected-access)

-----
Your code has been rated at 9.00/10 (previous run: 6.50/10, +2.50)
```

5 Minute Breather

Services with Code - Lab

Recap

Feedback - Survey

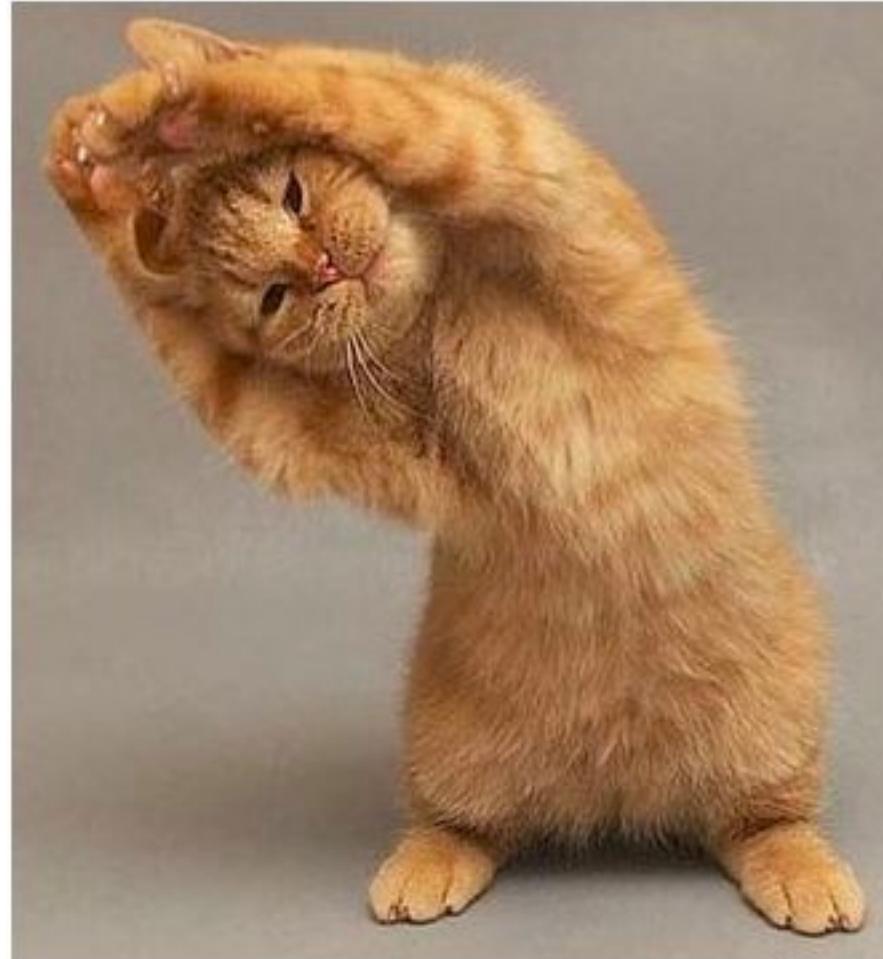
Recap

- Final overall questions
- Survey

THANK YOU!

Lab – Build Your Own Yang
Model and view the outputs
from GUI/CLI/Rest API

Stretch Break!



Review: Modeling Cisco IOS commands in Yang

```
ip access-list standard <NAME>
  permit <IP address 1>
  permit <IP address 2>
ip access-list extended <Name>
  permit <rule>
  deny <rule>
```

```
Container ip
  container access-list {
    container standard
    list std-named-acl {
      leaf name {type std-acl-type;}
    }
    list std-access-list-rule {
      leaf rule {type string;}}}
  container extended {
    list ext-named-acl {
      leaf name {type string;}
    }
    list ext-access-list-rule {
      leaf rule {type string;}}}
```

Wait a minute...

- Yang tells me the hierarchy of my input
- Tells me the possible fields of my data
- But how about where my data begins and ends?
- Or even what are the actual values of my data?

XML

- Extensible Markup Language
- Tells me where fields begin and end.
- Formatting used by Netconf payload
- NSO converts configuration data to XML even if using a non-Netconf deployment method.

Cisco IOS in XML

Cutsheet

radius server <AAA Server>

 address ipv4 <IP Address> auth-port <Port #> acct-port <Port #>

 key <Level> <Encrypted Key>

Actual Configuration

radius server primary-trustsec-radius

 address ipv4 173.36.131.232 auth-port 1812 acct-port 1813

 key 7 1107160A1F021218

```
<radius xmlns="urn:ios">  
    <server>  
        <id>primary-trustsec-radius</id>  
        <address>  
            <ipv4>  
                <acct-port>1813</acct-port>  
                <auth-port>1812</auth-port>  
                <host>173.38.200.166</host>  
            </ipv4>  
        </address>  
        <key>  
            <type>7</type>  
            <secret>1107160A1F021218</secret>  
        </key>  
    </server>  
</radius>
```

Cisco IOS in XML again

Actual Configuration

```
ip access-list extended PREAUTH-ACL  
permit udp any any eq bootps  
permit udp any any eq bootpc
```

```
<ip>  
  <access-list>  
    <extended>  
      <ext-named-acl>  
        <name>PREAUTH-ACL</name>  
        <ext-access-list-rule>  
          <rule>permit udp any any eq bootps</rule>  
        </ext-access-list-rule>  
        <ext-access-list-rule>  
          <rule>permit udp any any eq bootpc</rule>  
        </ext-access-list-rule>  
      </ext-named-acl>  
    </extended>  
  </access-list>  
</ip>
```

Combining YANG and XML

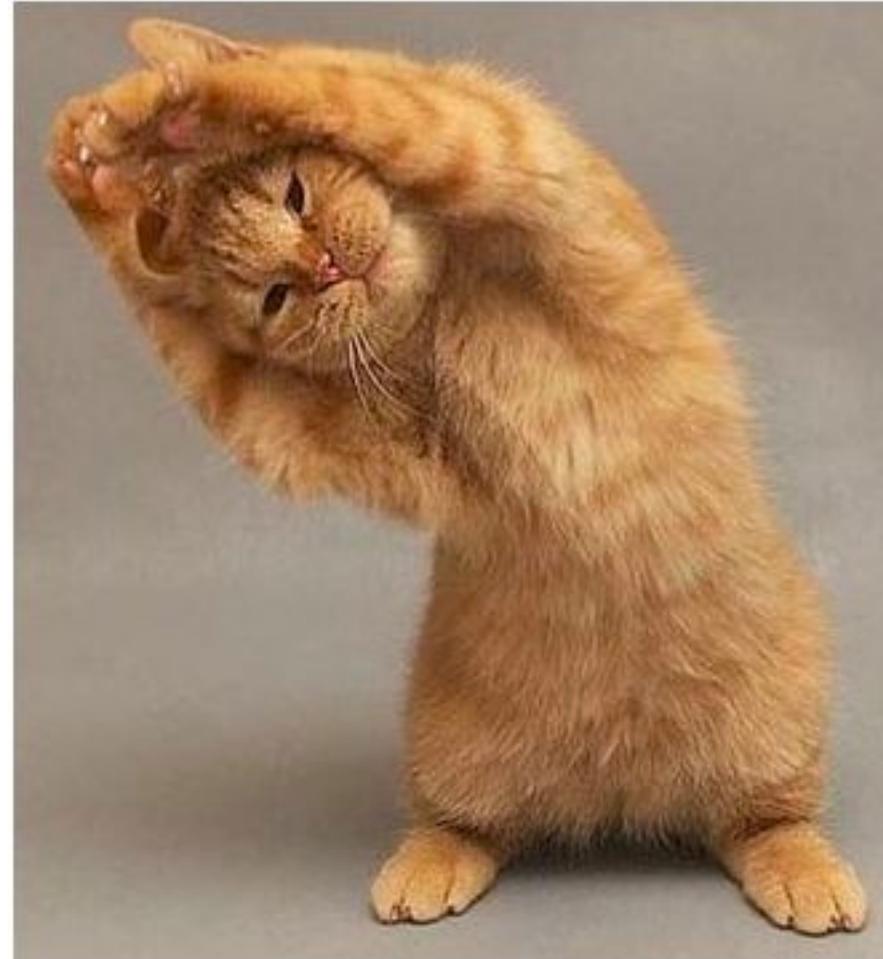
```
container radius {  
    list server {  
        leaf id {type string;}  
        container address {  
            container ipv4 {  
                leaf host {type string;}  
                leaf auth-port {type uint16;}  
                leaf acct-port {type uint16;}  
                container key {  
                    leaf encryption {type enumeration;}  
                    leaf key {type string;}}}}}}
```

```
<radius xmlns="urn:ios">  
    <server>  
        <id>primary-trustsec-radius</id>  
        <address>  
            <ipv4>  
                <acct-port>1813</acct-port>  
                <auth-port>1812</auth-port>  
                <host>173.38.200.166</host>  
            </ipv4>  
        </address>  
        <key>  
            <type>7</type>  
            <secret>1107160A1F021218</secret>  
        </key>  
    </radius>
```

Recap

- Unstructured configuration data becomes structured through YANG and XML
 - YANG defines the hierarchy and what the fields should be
 - XML defines where the field begins and ends, and contains actual data
- Reduces development overhead by using structured data
 - Even if NETCONF is not used as the main delivery method, there is value to be gained from having structured data that can be parsed

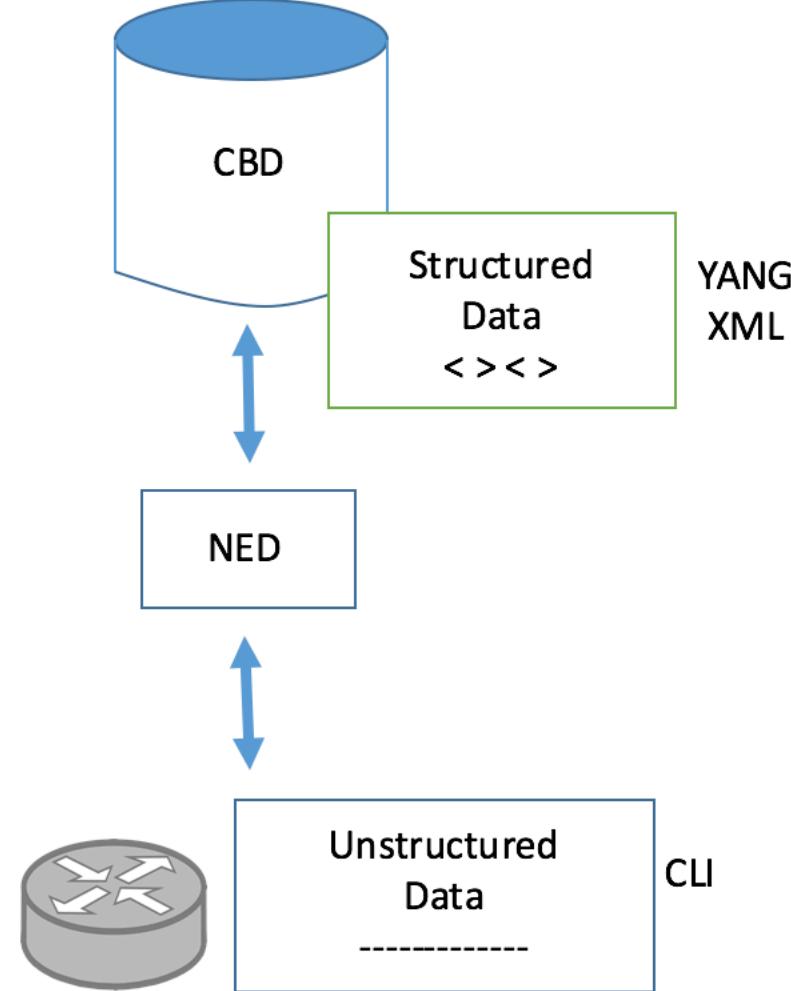
Stretch Break!



Practical YANG and XML

Network Element Drivers

- A translator between the structured data (YANG & XML) of NSO and the unstructured data from a network device.
- Consists of YANG models and Java Expect code
- They are stored within:
 - System Install: /apps/ns0/var/opt/ncs/packages
 - Local Install: /Users/<username>/ncs-run/packages
- Maintained by NSO BU at: <https://earth.tail-f.com:8443/ncs-pkgs/>



How to use NEDs as reference

- Go the appropriate package folder: packages/<platform>/src/yang
- Open up the YANG file and search by command

```
/// =====
/// aaa
/// =====

container aaa {
    tailf:info "Authentication, Authorization and Accounting.";
    tailf:cli-incomplete-command;

    // aaa new-model
    leaf new-model {
        tailf:info "Enable NEW access control commands and functions."+
                    "(Disables OLD commands.)";
        type empty;
    }

    // aaa group
    container group {
        tailf:info "AAA group definitions";
        tailf:cli-diff-dependency "/ios:aaa/new-model";
        // aaa group server
        container server {
            tailf:info "AAA Server group definitions";
        }
    }
}
```

Converting CLI to XML

```
[patruhn@ncs# config t  
Entering configuration mode terminal  
[patruhn@ncs(config)# devices device sjc12-31-sw1.cisco.com config  
[patruhn@ncs(config-config)# ios:ip access-list standard 28  
[patruhn@ncs(config-std-nacl)# permit 10.10.10.10
```



```
[patruhn@ncs(config-std-nacl)# commit dry-run outformat xml  
result-xml {  
    local-node {  
        data <devices xmlns="http://tail-f.com/ns/ncs">  
            <device>  
                <name>sjc12-31-sw1.cisco.com</name>  
                <config>  
                    <ip xmlns="urn:ios">  
                        <access-list>  
                            <standard>  
                                <std-named-acl>  
                                    <name>28</name>  
                                    <std-access-list-rule>  
                                        <rule>permit 10.10.10.10</rule>  
                                    </std-access-list-rule>  
                                </std-named-acl>  
                            </standard>  
                        </access-list>  
                    </ip>  
                </config>  
            </device>  
        </devices>  
    }  
}
```

Running Configuration in XML

```
patruyn@ncs# show running-config devices device sjc12-31-sw1.com | display xml
```

```
<aaa xmlns="urn:ios">
  <new-model/>
  <group>
    <server>
      <radius>
        <name>trustsec-radius</name>
        <server>
          <name>
            <name>primary-trustsec-radius</name>
          </name>
          <name>
            <name>secondary-trustsec-radius</name>
          </name>
        </server>
      </radius>
      <tacacs-plus>
        <name>vty_access</name>
        <server>
          <server-list>
            <name>64.104.123.61</name>
            . . .
        </server-list>
      </tacacs-plus>
    </server>
  </group>
</aaa>
```

How to use xpath as reference

```
patruyn@ncs# show running-config devices device sjc12-31-sw1.com | display xpath
```

```
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/new-model
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/group/server/radius[name='trustsec-radius']/se
rver/name[name='primary-trustsec-radius']
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/group/server/radius[name='trustsec-radius']/se
rver/name[name='secondary-trustsec-radius']
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/group/server/tacacs-plus[name='vty_access']/se
rver/server-list[name='64.104.123.61']
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/group/server/tacacs-plus[name='vty_access']/se
rver/server-list[name='171.70.168.112']
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/group/server/tacacs-plus[name='vty_access']/se
rver/server-list[name='173.38.203.29']
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/authentication/dot1x[name='default']/group tru
stsec-radius
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/authentication/login[name='admin']/group tacac
s+
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/authentication/login[name='admin']/enable
```

How to use xpath as reference

```
patruhn@ncs# show running-config devices device sjc12-31-sw1.cisco.com | include dot1x | display xpath  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/aaa/authentication/dot1x[name='default']/group true  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/ios:class-map-filter-control/class-map[name='DOT1X']  
/match/method/dot1x  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/ios:class-map-filter-control/class-map[name='DOT1X_FAILED']  
/match/method/dot1x  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/ios:class-map-filter-control/class-map[name='DOT1X_FAILED']  
/match/result-type/method/dot1x/authoritative  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/ios:class-map-filter-control/class-map[name='DOT1X_NO_RESP']  
/match/method/dot1x  
/devices/device[name='sjc12-31-sw1.cisco.com']/config/ios:class-map-filter-control/class-map[name='DOT1X_NO_RESP']  
/match/result-type/method/dot1x/agent-not-found
```

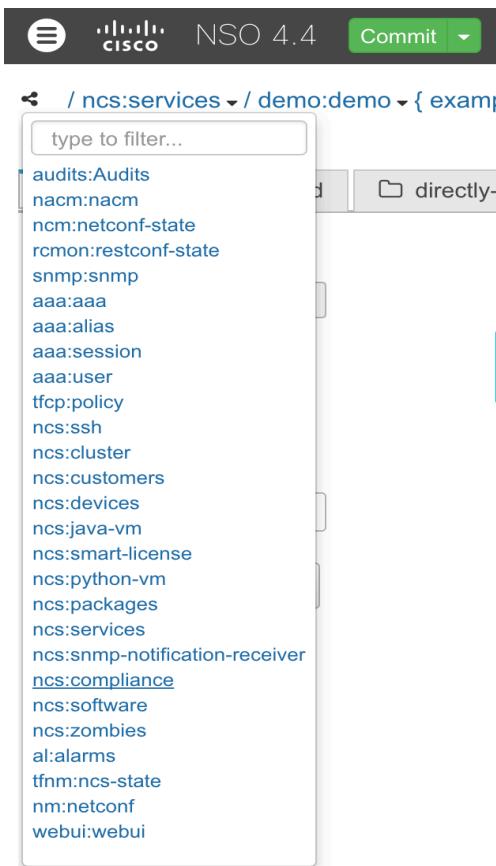
LAB: Yang and XML

LAB: Yang and XML

- Create your own YANG and XML for this command:
 - switchport access vlan 2
 - You can do this for any interface of your choice.
- Run the command through the NSO CLI and do a “commit dry-run outformat xml”
 - Did you expect the output that you got?
- Compare your YANG to the YANG in the NED.
 - Is the format the same?

Native Compliance Report Tool

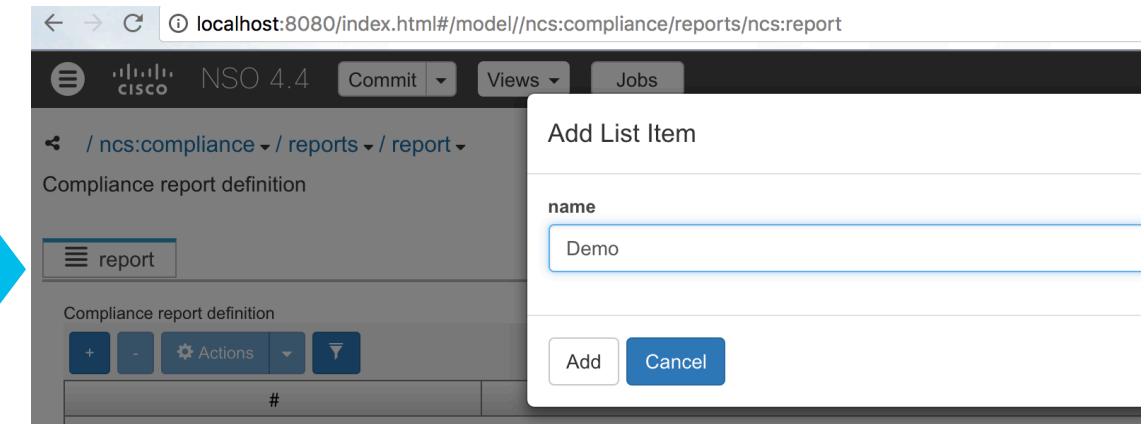
Go to ncs:compliance



Go reports > report



Click [+] to make a new report



Native Compliance Report Tool

Configure the report to audit services

The screenshot shows the NSO 4.4 interface. At the top, there are navigation buttons: Commit (green), Views, and Jobs. Below the header, the URL path is /ncs:compliance / reports / report { Demo }. A tooltip "Report on services out of sync" points to the "service-check" tab, which is highlighted in blue. Other tabs include report, device-check, and compare-template. Under the "INFO" section, there is a "name" field containing "Demo". At the bottom right is a "run..." button with a gear icon.

Select “some-services” and enter the XPATH for the service name

The screenshot shows the NSO 4.3 interface. At the top, there are navigation buttons: Commit (green), Views, and Jobs. Below the header, the URL path is /ncs:compliance / reports / report { Lab_Gateway_ACL_Test }. The "service-check" tab is selected and highlighted in blue. A tooltip "Report on services out of sync" is shown above the tab. Below the tabs, there is a "Choice - service-choice" dropdown set to "some-services". Under the "service-check" section, there is a checked checkbox. In the "select-services" section, the XPATH "/ncs:services/lab_gateway_/" is entered. The "service" section is empty with a plus sign icon and the message "No items to display".

Native Compliance Report Tool

Return to the report container & click run...

The screenshot shows the NSO 4.3 interface. At the top, there's a navigation bar with icons for Cisco, Commit, Views, and Jobs. Below the bar, a URL path is displayed: /ncs:compliance -> /reports -> /report -> { Lab_Gateway_ACL_Test } ->. A modal window titled "Run this compliance report" is open, containing a "run..." button. The background shows a "INFO" section with a "name" field set to "Lab_Gateway_ACL_Test".

Enter a title and outformat then run!

The screenshot shows the NSO 4.3 interface with a modal dialog for running a compliance report. The dialog has a header "Run this compliance report". It includes fields for "title" (set to "demo!"), "from" (Date and Time dropdowns), and "to" (Date and Time dropdowns). To the right, a tooltip explains the "outformat" field: "The format of this report output file, and therefore also the file suffix". The "outformat" dropdown is currently set to "html". Below the main form is a "Invoke run" button and a table with two columns, "Name" and "Value", which is empty. A message at the bottom states "No items to display".

Native Compliance Report Tool

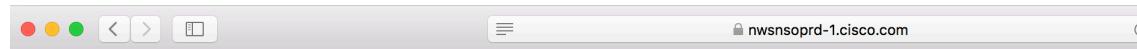
We now have a report! Copy the “/compliance-report...” and append it to the servers hostname (webUI bug)

The screenshot shows the NSO 4.3 web interface. At the top, there is a navigation bar with icons for list, Cisco logo, and NSO 4.3, followed by buttons for Commit, Views, Jobs, and Alarms (with 5078 notifications). Below the navigation bar, the URL is displayed as /ncs:compliance / reports / report { Lab_Gateway_ACL_Test } / run. A link to "Run this compliance report" is present. A "run" button is highlighted with a blue border. Below this, there are input fields for "title" (demo!), "from" (Date and Time dropdowns), "to" (Date and Time dropdowns), and "outformat" (html dropdown). A "Invoke run" button is also visible. At the bottom, a table displays report parameters: Name, Value, including id (59), compliance-status (violations), info (Checking no devices and 478 services), and location (http://localhost:8080/compliance-reports/report_59_branblac_1_2017-6-29T12:38:10.html).

Name	Value
id	59
compliance-status	violations
info	Checking no devices and 478 services
location	http://localhost:8080/compliance-reports/report_59_branblac_1_2017-6-29T12:38:10.html

Native Compliance Report Tool

We now have a report!



demo!

Publication date : 2017-6-29 12:38:1

Produced by user : branblac

Summary

Compliance result titled "demo!" defined by report "Lab_Gateway_ACL_Test"

Resulting in **violations**

Checking no devices and 478 services

Produced 2017-6-29 12:38:1

From : Oldest available information

To : 2017-6-29 12:38:1

Services out of sync

/services/lab_gateway_ACL:lab_gateway_ACL{alln01-lab-gw1.cisco.com}

Service /services/lab_gateway_ACL:lab_gateway_ACL{alln01-lab-gw1.cisco.com} out of sync

/services/lab_gateway_ACL:lab_gateway_ACL{ams5-lab-gw1.cisco.com}

Service /services/lab_gateway_ACL:lab_gateway_ACL{ams5-lab-gw1.cisco.com} out of sync

/services/lab_gateway_ACL:lab_gateway_ACL{argrp4-in-lab-gw1.cisco.com}

Service /services/lab_gateway_ACL:lab_gateway_ACL{argrp4-in-lab-gw1.cisco.com} out of sync

/services/lab_gateway_ACL:lab_gateway_ACL{argrp4-in-lab-gw2.cisco.com}

Service /services/lab_gateway_ACL:lab_gateway_ACL{argrp4-in-lab-gw2.cisco.com} out of sync

/services/lab_gateway_ACL:lab_gateway_ACL{beg02-lab-gw1.cisco.com}

Details for "/services/lab_gateway_ACL:lab_gateway_ACL{ams5-lab-gw1.cisco.com}"

Local node changes

```
devices {
    device ams5-lab-gw1.cisco.com {
        config {
            ios:interface {
                GigabitEthernet 2/1 {
                    ip {
                        access-group out {
                            access-list lab-lenient;
                        }
                    }
                    ipv6 {
                        traffic-filter out {
                            access-list lab-lenient-ipv6;
                        }
                    }
                }
                GigabitEthernet 3/1 {
                    ip {
                        access-group out {
                            access-list lab-lenient;
                        }
                    }
                    ipv6 {
                        traffic-filter out {
                            access-list lab-lenient-ipv6;
                        }
                    }
                }
            }
        }
    }
}
```

Native Compliance Report Tool

- Report Components:
 - Summary – High level info
 - Services-out-of-sync – List of instances that are non-compliant
 - Details / Service differences –
 - Per service instance
 - Per device in the instance
 - Gives the configuration info that is out of sync.
 - Includes what needs to be added and what needs to be removed

Service Instance Check-Sync

- A service instance check-sync will check if all the devices for a specific service instance are “in-sync” or compliant with the service model.
- Uses the saved input parameters for the service models YANG for that instance to determine what the config should be, and checks against what the device's config is

Service Instance Check-Sync

Select the Instance to Audit

The screenshot shows the NSO 4.3 interface with the following elements:

- Top navigation bar: NSO 4.3, Commit, Views, Jobs.
- Current path: /ncs:services/lab_gateway_ACL:lab_gateway_ACL
- Action bar: Edit, Delete, Add, Actions dropdown.
- Table view of service instances:

	<input type="checkbox"/>	name
1	<input type="checkbox"/>	abj01-lab-gw1.cisco.com
2	<input type="checkbox"/>	abq-lab-gw1.cisco.com
3	<input type="checkbox"/>	adl-lab-gw1.cisco.com
4	<input type="checkbox"/>	akl02-lab-gw1.cisco.com
5	<input type="checkbox"/>	alb-lab-gw1.cisco.com
6	<input type="checkbox"/>	alln01-lab-gw1.cisco.com

Service Instance Check-Sync

Click “check-sync...”

The screenshot shows the NSO 4.3 interface with the following details:

- Header:** NSO 4.3, Commit ▾, Views ▾, Jobs, Alarms 5078, branblac ▾.
- Breadcrumbs:** / ncs:services ▾ / lab_gateway_ACL:lab_gateway_ACL ▾ { ams5-lab-gw1.cisco.com } ▾.
- Toolbar:** lab_gateway_ACL (selected), modified, directly-modified, commit-queue, log, Uplink_Interfaces.
- INFO section:** name: ams5-lab-gw1.cisco.com.
- device-list:** ams5-lab-gw1.cisco.com.
- used-by-customer-service:** No items to display.
- device:** ams5-lab-gw1.cisco.com (+).
- Action Buttons:** check-sync..., deep-check-sync..., re-deploy..., reactive-re-deploy..., touch..., get-modifications..., un-deploy....

Service Instance Check-Sync

The screenshot shows the NSO 4.3 interface with the following details:

- Header:** NSO 4.3, Commit, Views, Jobs.
- Breadcrumbs:** / ncs:services > / lab_gateway_ACL:lab_gateway_ACL > { ams5-lab-gw1.cisco.com } > / check-sync.
- Page Title:** Check if device config is according to the service
- Form Fields:**
 - check-sync** (button)
 - Check if device config is according to the service
 - outformat**: native (selected)
 - Choice - depth**: - (dropdown)
 - suppress-positive-result**:
 - Choice - choice-lsa**: - (dropdown)
 - Choice - outformat**: - (dropdown)
- Buttons:** Invoke check-sync (button)

Select the format for the output
(Native means CLI)

And click “Invoke check-sync”!

Service Instance Check-Sync

Our audit results are at the bottom!

This process is available via all API as well

⚙️ Invoke check-sync	
Name	Value
native/device/name	ams5-lab-gw1.cisco.com
native/device/data	<pre>interface GigabitEthernet2/1 no switchport ip access-group lab-lenient out ipv6 traffic-filter lab-lenient-ipv6 out no shutdown exit interface GigabitEthernet3/1 no switchport ip access-group lab-lenient out ipv6 traffic-filter lab-lenient-ipv6 out</pre>