



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:

Gateway Configuration

Wireless Service

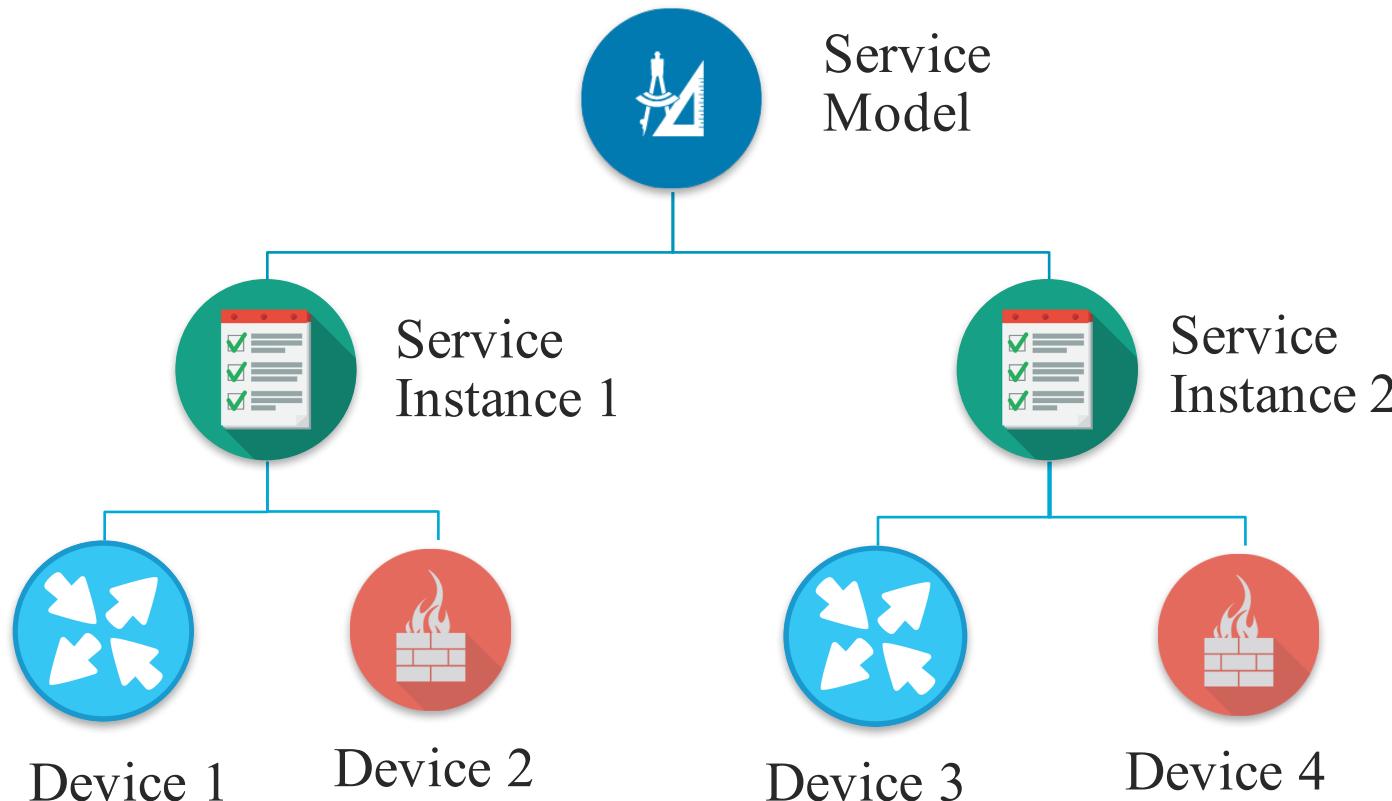
WLC Configuration

Switch Configuration



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

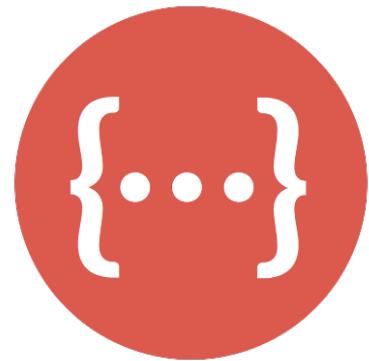
Service Features & Capabilities

- 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](#)



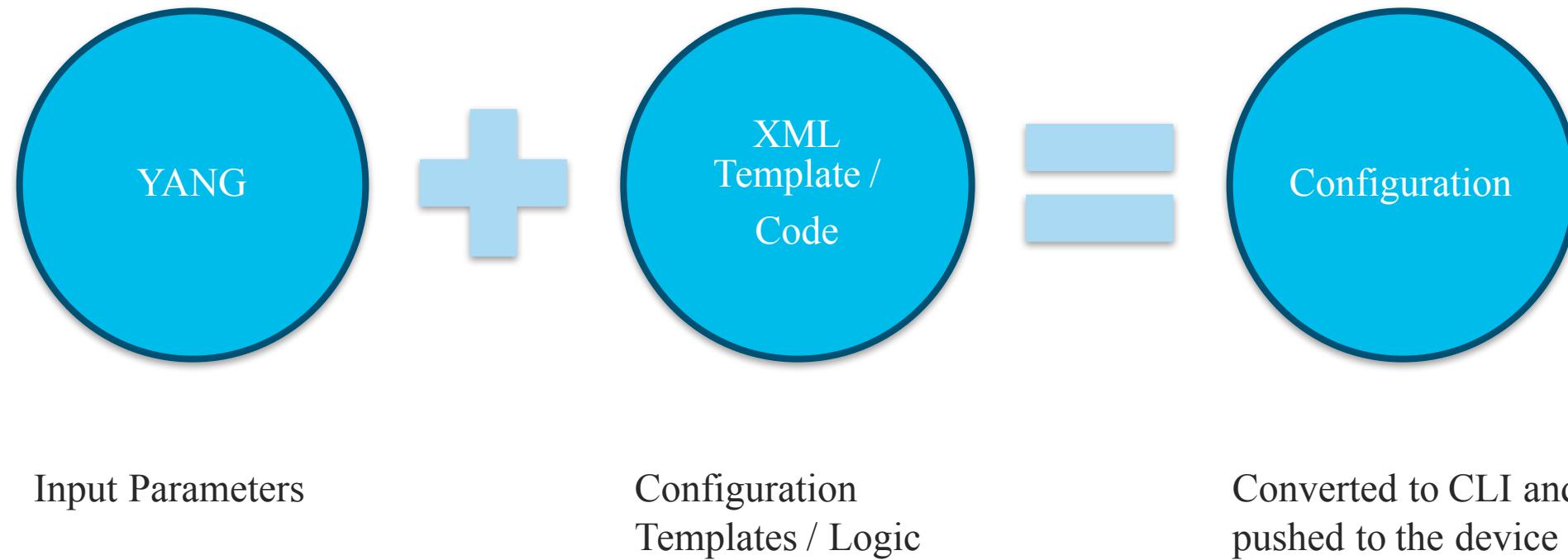
Service Template

[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 input meta-data are stored inside the cDB for reference
and used in **determining configuration** to be applied.

Building a Service

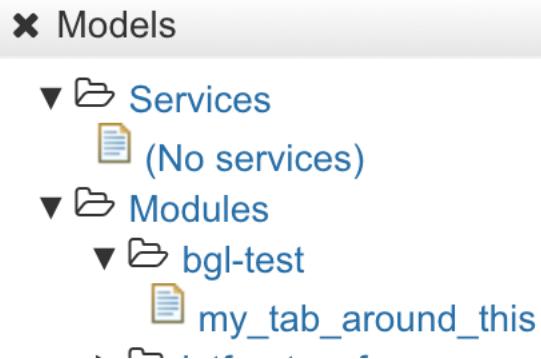
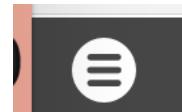
Demo building a service package

- `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
}
```

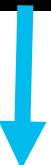
Models

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

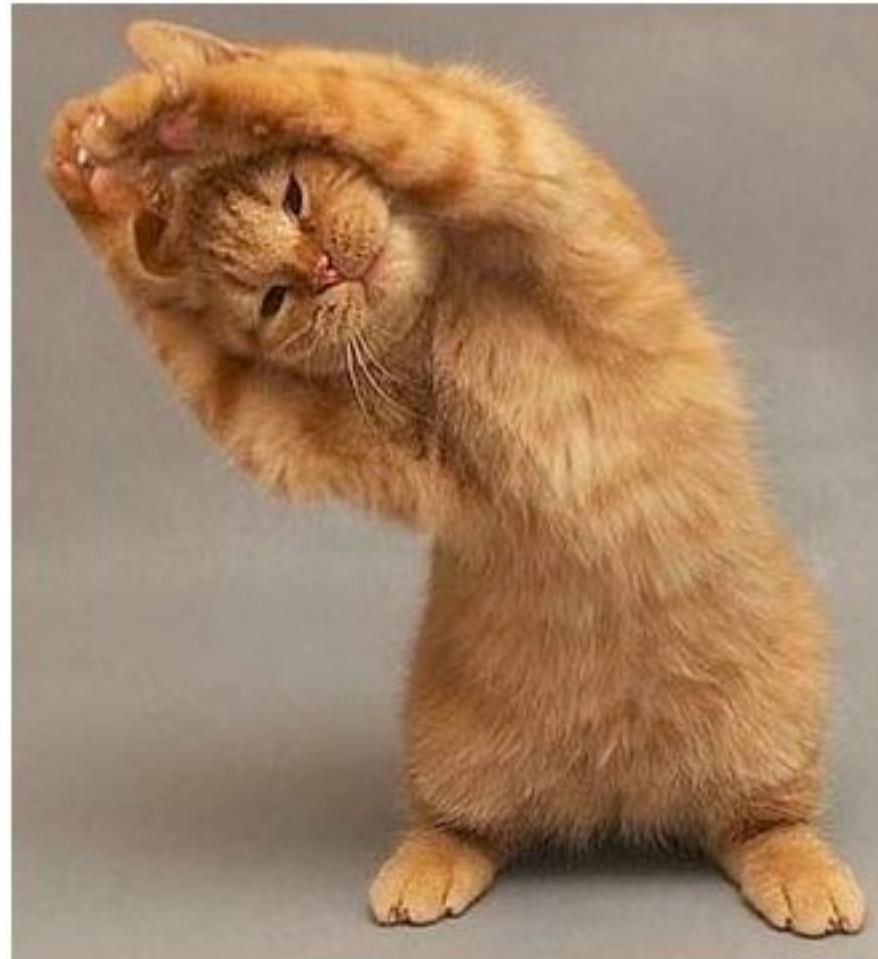
High-level development flow

- 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

A screenshot of a web-based service database interface. The URL in the address bar is /ncs:services/lab_gateway_ACL:lab_gateway_ACL. The page displays a table with a single column labeled "name". The table has 6 rows, indexed from 1 to 6. The data in the table is:

#	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 ▾

A screenshot of a web-based service database interface. The URL in the address bar is /ncs:services/lab_gateway_ACL:lab_gateway_ACL/{ bdlk09-00-lab-gw1.cisco.com }/Uplink_Interfaces/GigabitEthernet. The page displays a table with a single column labeled "interfaceNumber (k)". The table has 2 rows, indexed from 1 to 2. The data in the table is:

#	interfaceNumber (k)
1	1/1
2	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!

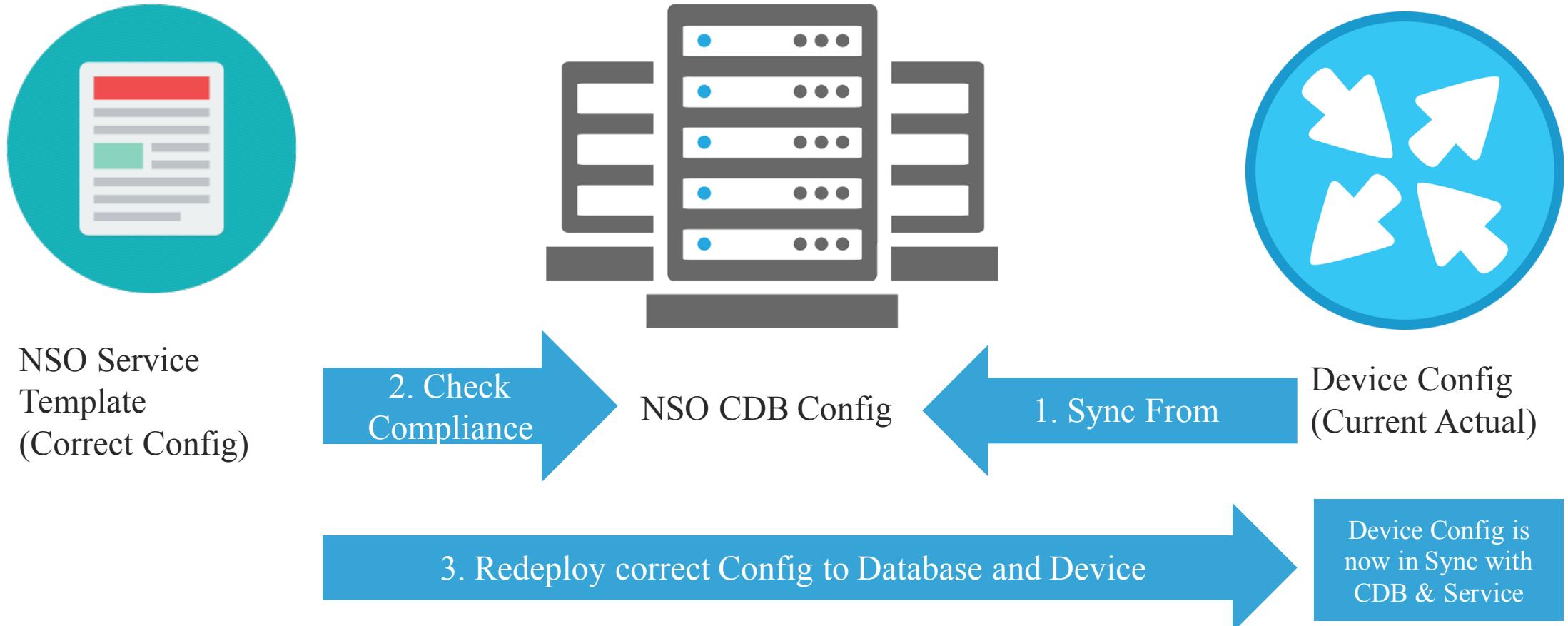
Compliance Reporting

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 deep-check-sync will check if all the devices for a specific service instance are “in-sync” or compliant with the service model. A service check-sync will only check the data in the CDB is correct, a deep-check-sync is required to check configuration on the devices is correct.
- 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 (+)
- Action Buttons:**
 - check-sync..., deep-check-sync..., re-deploy..., reactive-re-deploy...
 - touch..., get-modifications..., un-deploy...

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:** Cisco 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

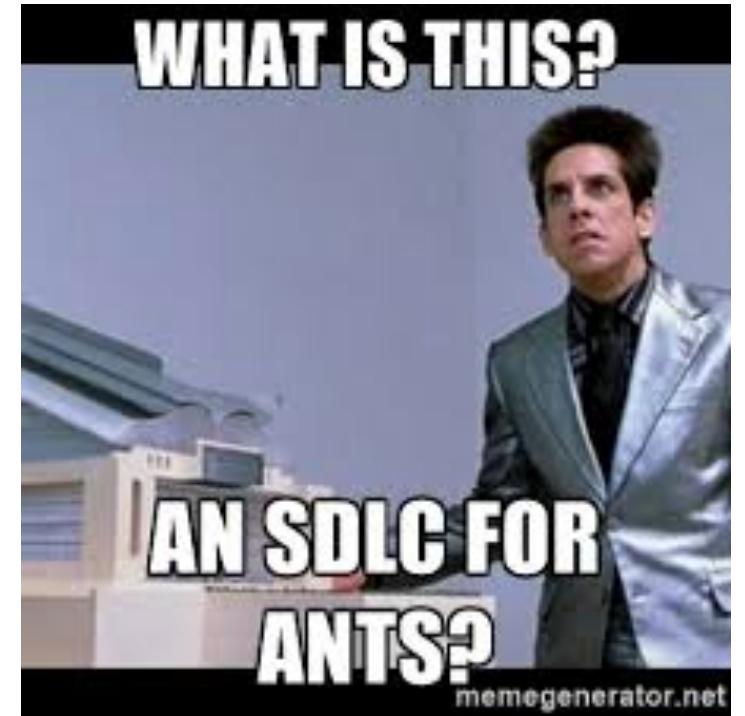
Service Development

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
```

!

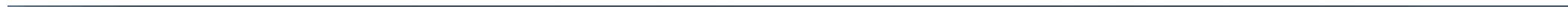
- If Region = AMER then IP_ADDRESS = 10.0.1.1
 - If Region = APJC then IP_ADDRESS = 10.0.2.1
 - If Region = EMEAR then IP_ADDRESS = 10.0.3.1
-
- Then we want to apply this to each inputted device

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 extra step due to the design of ncs-make-package...

Please include

“augment /ncs:services { list your_service_name {} }

In the yang file

This can also be done in the ncs-make-package through using the --augment argument

Services can reside outside of the /service yang model.

```
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/devices/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' section with requests categorized 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. The XML content is as follows:

```
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>
```

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... IN SYNC'). The left sidebar displays a timeline of API requests made on 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 lists several headers, and the 'Body' tab displays the XML response. The XML response is filtered to show only specific sections related to a radius server configuration.

Request URL: `http://localhost:8080/api/running/devices/device/demo-0/config/radius?deep`

Headers:

Header	Value
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

Body (Pretty):

```
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:
 - {xpath_to_service_model_node}
- With simple yang models this is often:
 - {/input_variable_name}
 - Using a “/” at the beginning makes it an absolute path rather than relative to it's template context
- 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      <!--
7          Select the devices from some data structure in the service
8              model. In this skeleton the devices are specified in a leaf-list.
9              Select all devices in that leaf-list:
-->
10         <name>/device</name>
11         <config>
12             <!--
13                 Add device-specific parameters here.
14                 In this skeleton the service has a leaf "dummy"; use that
15                 to set something on the device e.g.:
16                     <ip-address-on-device>/dummy</ip-address-on-device>
-->
17         <radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest" xmlns:ios="urn:ios" xmlns:ncs="http://tail-f.com/ns/ncs">
18             <server>
19                 <id>one</id>
20                 <address>
21                     <ipv4>
22                         <host>10.0.0.1</host>
23                         <auth-port>1812</auth-port>
24                         <acct-port>1813</acct-port>
25                     </ipv4>
26                 </address>
27                 <backoff>
28                     <exponential>
29                         </exponential>
30                     </backoff>
31                 </server>
32             </radius>
33         </config>
34     </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 main navigation bar shows the path: "/ ncs:services / demo:demo { dummy }". Below this, there are tabs for "demo", "modified", "directly-modified", "commit-queue", and "log".

The configuration pane displays the following details:

- INFO**:
 - name**: dummy
- device-list**: No items to display
- used-by-customer-service**: No items to display
- device**:
 - demo-0A "+" button is available to add more devices.

Below the configuration pane, there are several buttons:

- Region**: AMER dropdown
- check-sync...**
- deep-check-sync...**
- re-deploy...**
- reactive-re-deploy...**
- touch...**
- get-modifications...**
- 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