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.

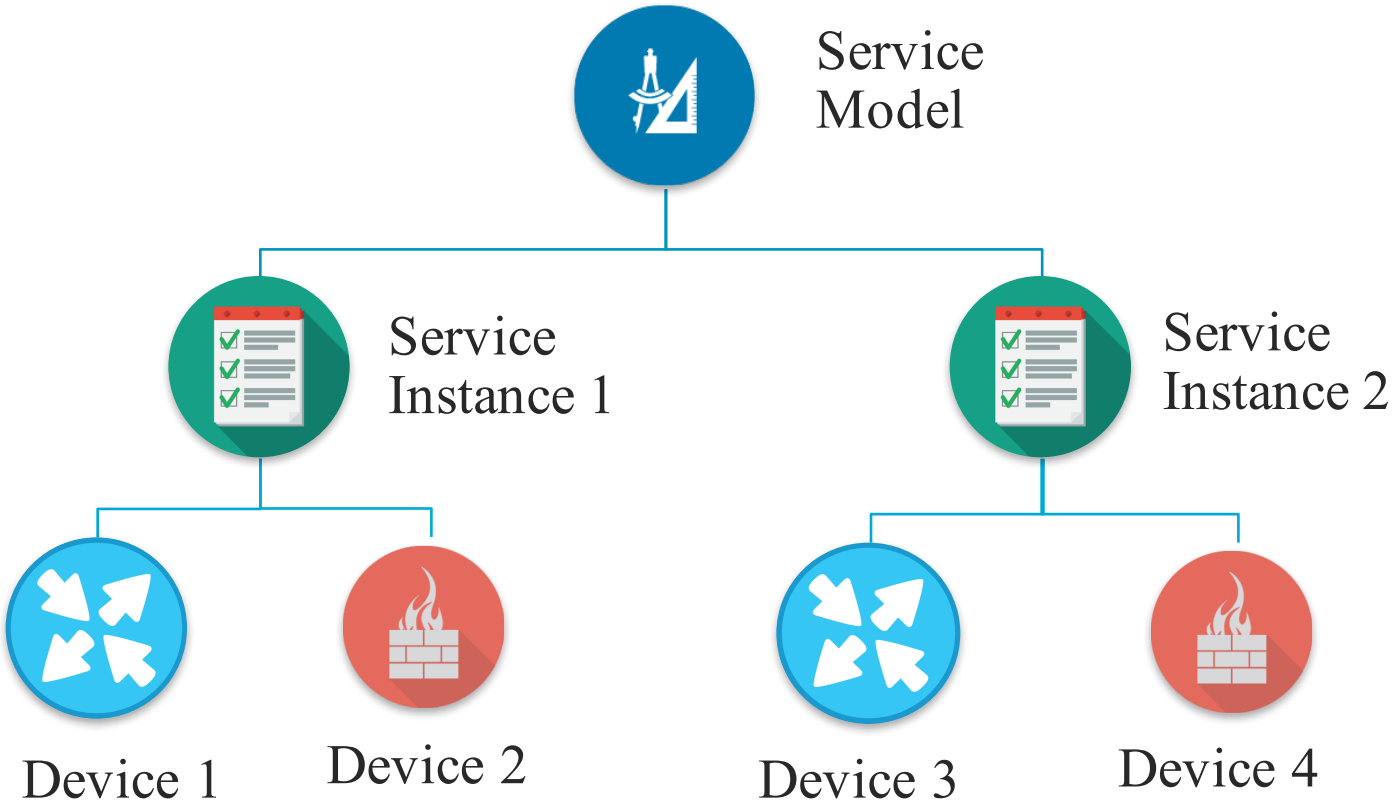
The “Service” in Network Service Orchestrator

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

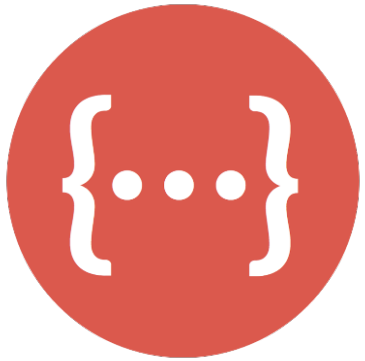
Service Features & Capabilities

- 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

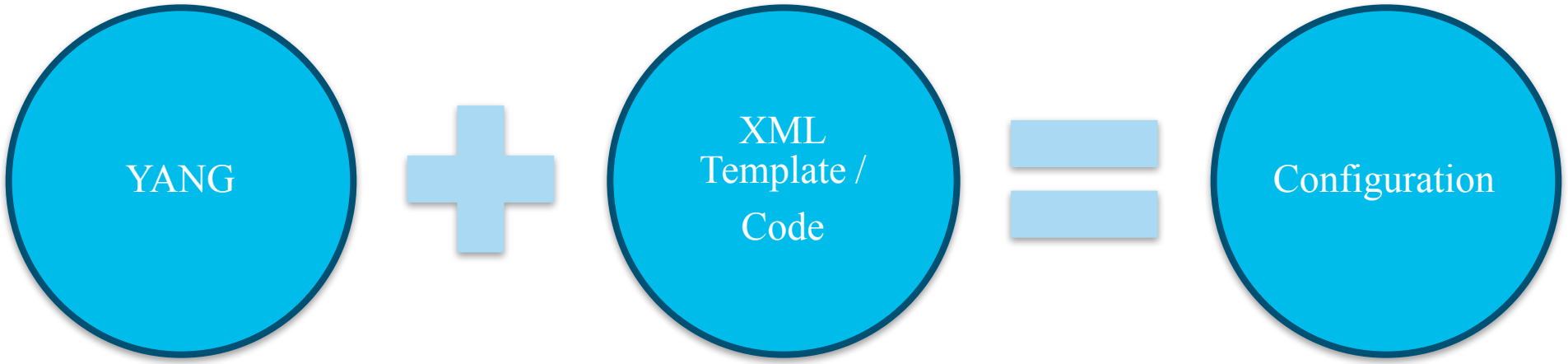
XML

Service Template

Python and Java

Service Logic

NSO Service Packages



Input Parameters

Configuration
Templates / Logic

Converted to CLI and
pushed to the device

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

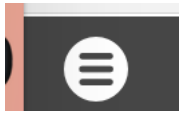
Demo building a service package

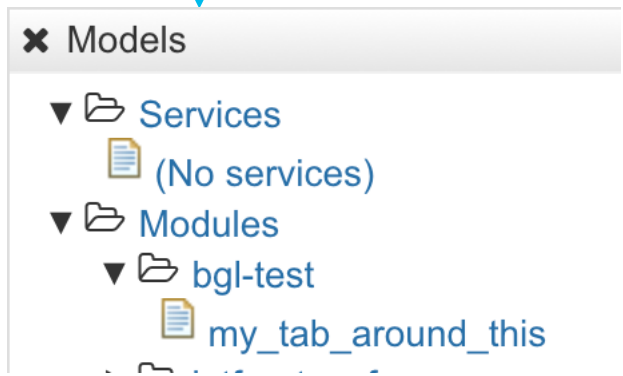
- ncs-make-package --service-skeleton template bgl-test
- Took out some dummy values, change into directory, remove xml template file for now, run make in src folder, and then packages reload

```
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$ kickoffnso
```

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



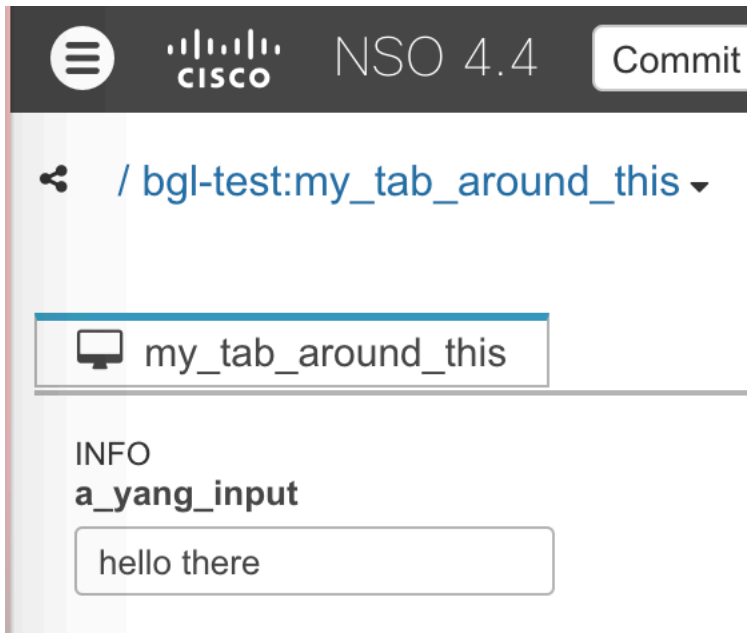


GUI
or
CLI

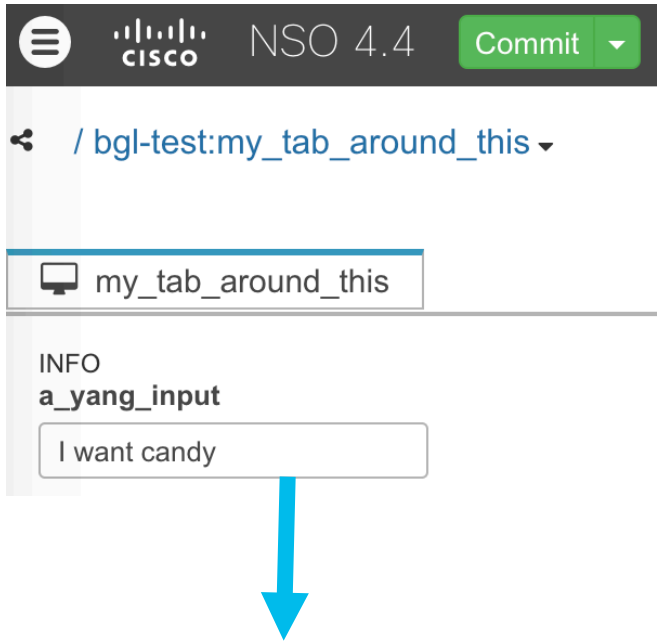
```
admin@ncs(config)# my_tab_around_this ?
Possible completions:
  a_yang_input
admin@ncs(config)# my_tab_around_this a_yang_input ?
Possible completions:
  <string>
admin@ncs(config)# my_tab_around_this a_yang_input
```

CLI to GUI

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



Or GUI to CLI




```
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"
```

Lab – bgl test service

Developing Services

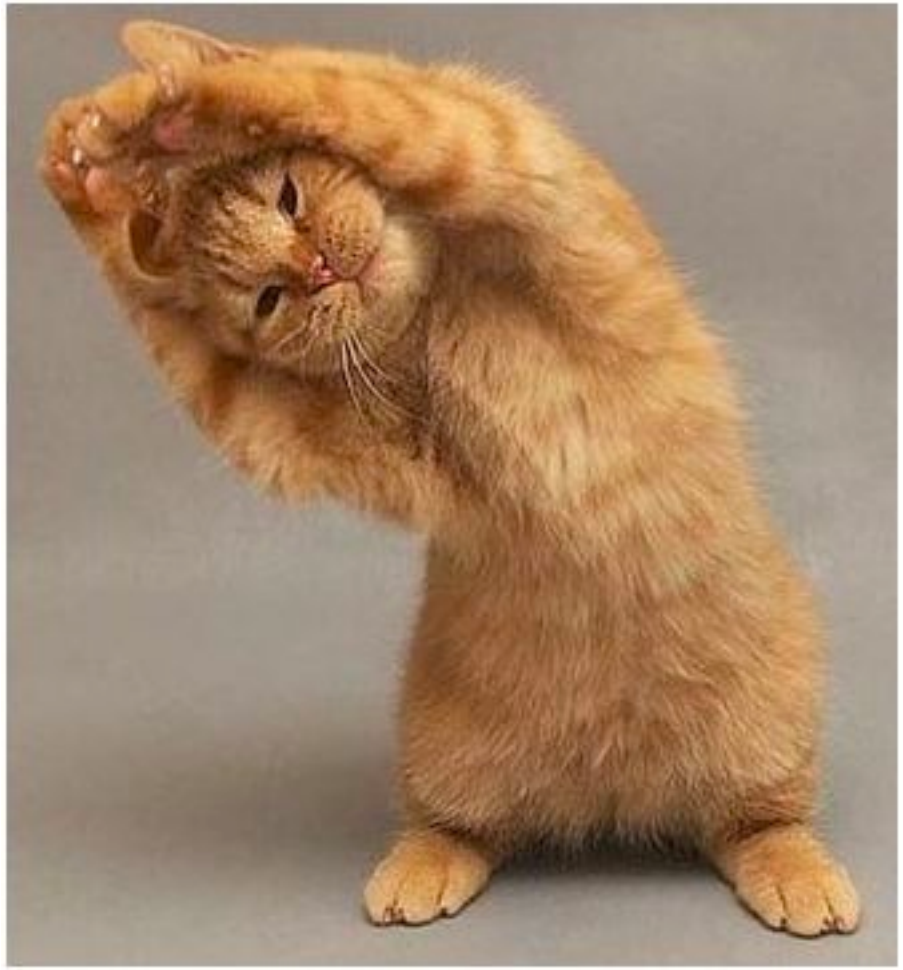
High-level development flow

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

Demo

Lab – Simple Service

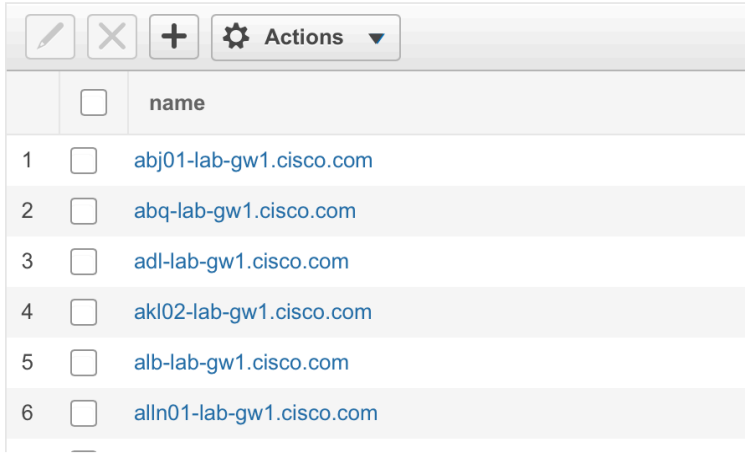
Stretch Break!

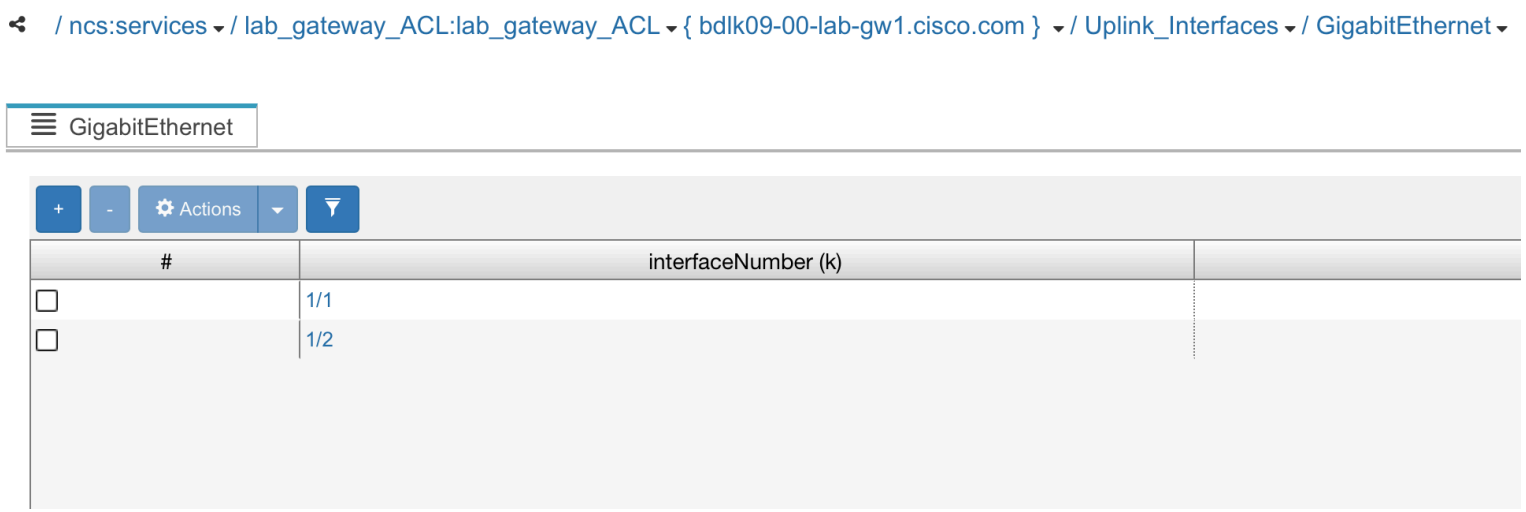


Service Database

- 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





“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

NSO Service Template (Correct Config)

NSO CDB Config

Device Config (Current Actual)

2. Check Compliance

1. Sync From

3. Redeploy correct Config to Database and Device

Device Config is now in Sync with CDB & Service

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
6        <name>{/device}</name>
7        <config>
8          <radius xmlns="urn:ios" when="{/Region='Amer'}">
9            <server>
10             <id>one</id>
11             <address>
12               <ipv4>
13                 <host>10.0.0.1</host>
14                 <auth-port>1812</auth-port>
15                 <acct-port>1813</acct-port>
16               </ipv4>
17             </address>
18
19             <backoff>
20               <exponential></exponential>
21             </backoff>
22           </server>
23         </radius>
24         <radius xmlns="urn:ios" when="{/Region='APJC'}">
25           <server>
26             <id>one</id>
27             <address>
28               <ipv4>
29                 <host>10.0.0.2</host>
30                 <auth-port>1812</auth-port>
31                 <acct-port>1813</acct-port>
32               </ipv4>
```

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

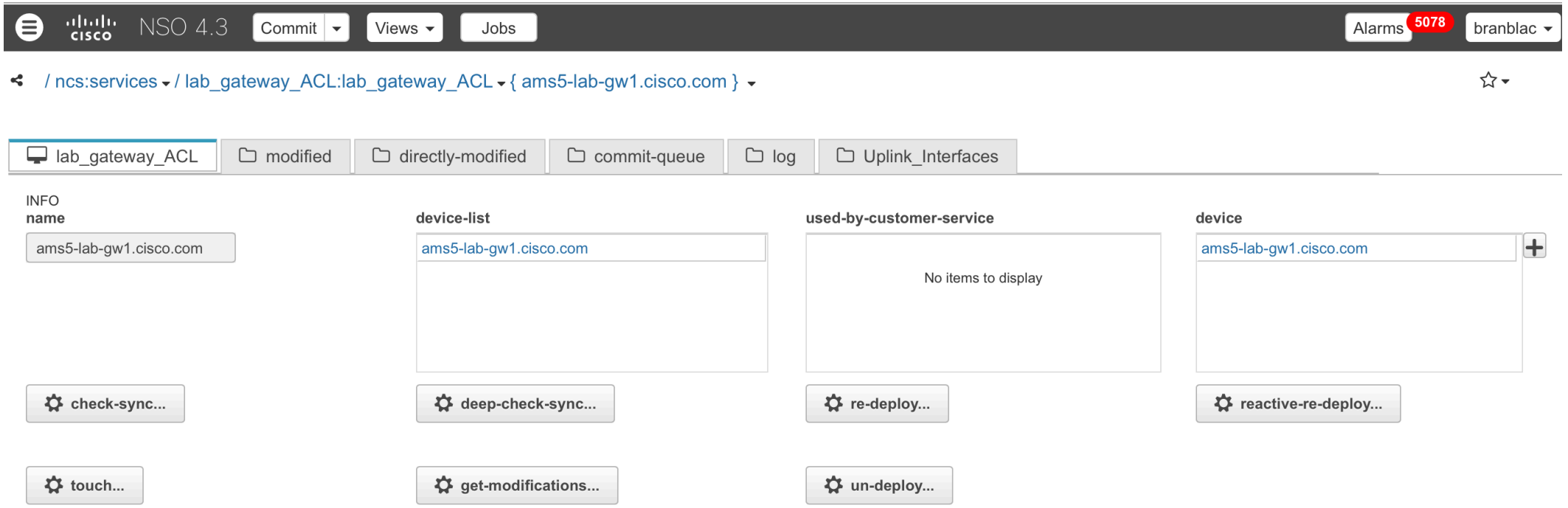
Compliance Remediation

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

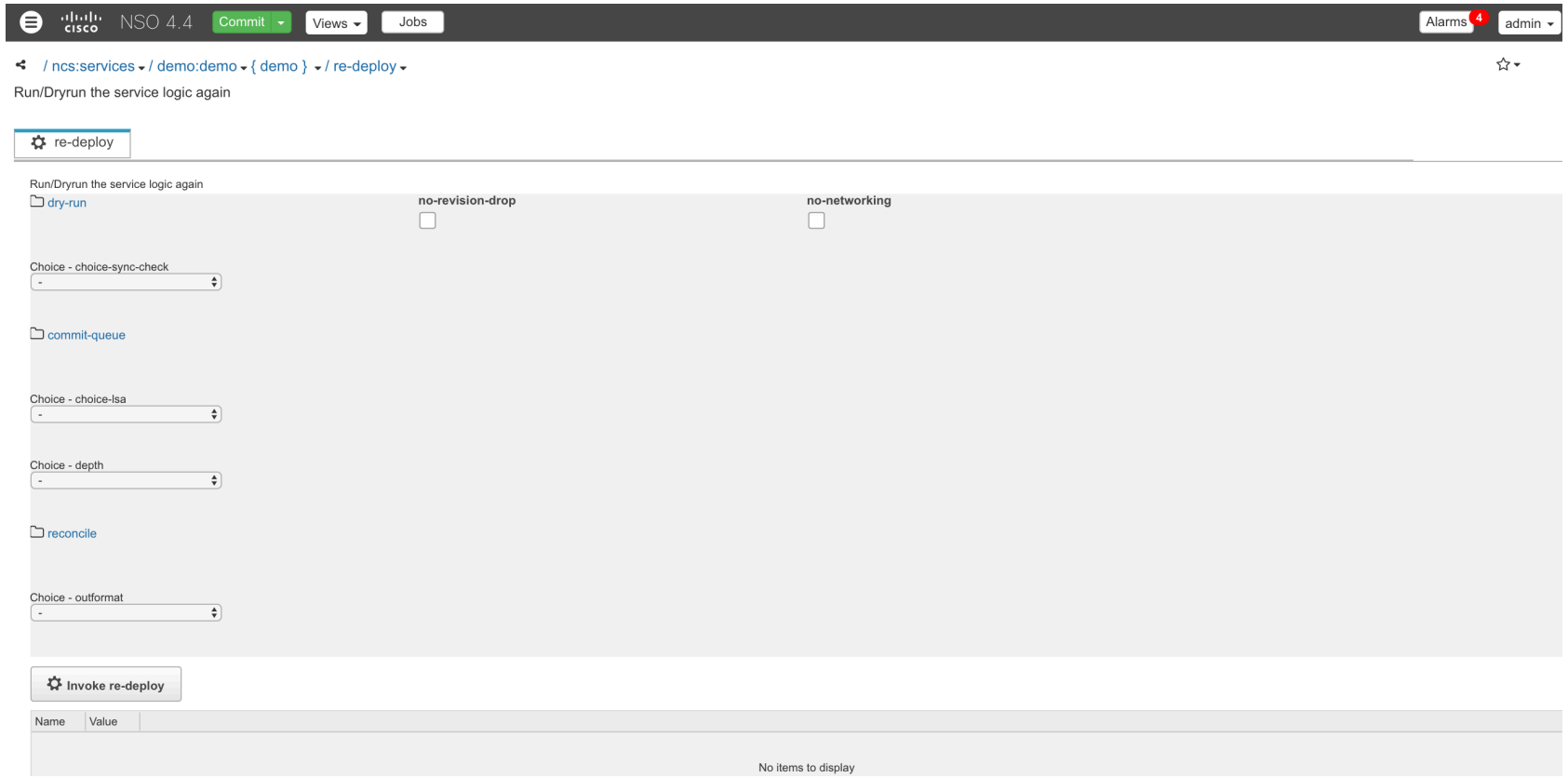
Navigate to a service instance and enter the “re-deploy…” action



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



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 their 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

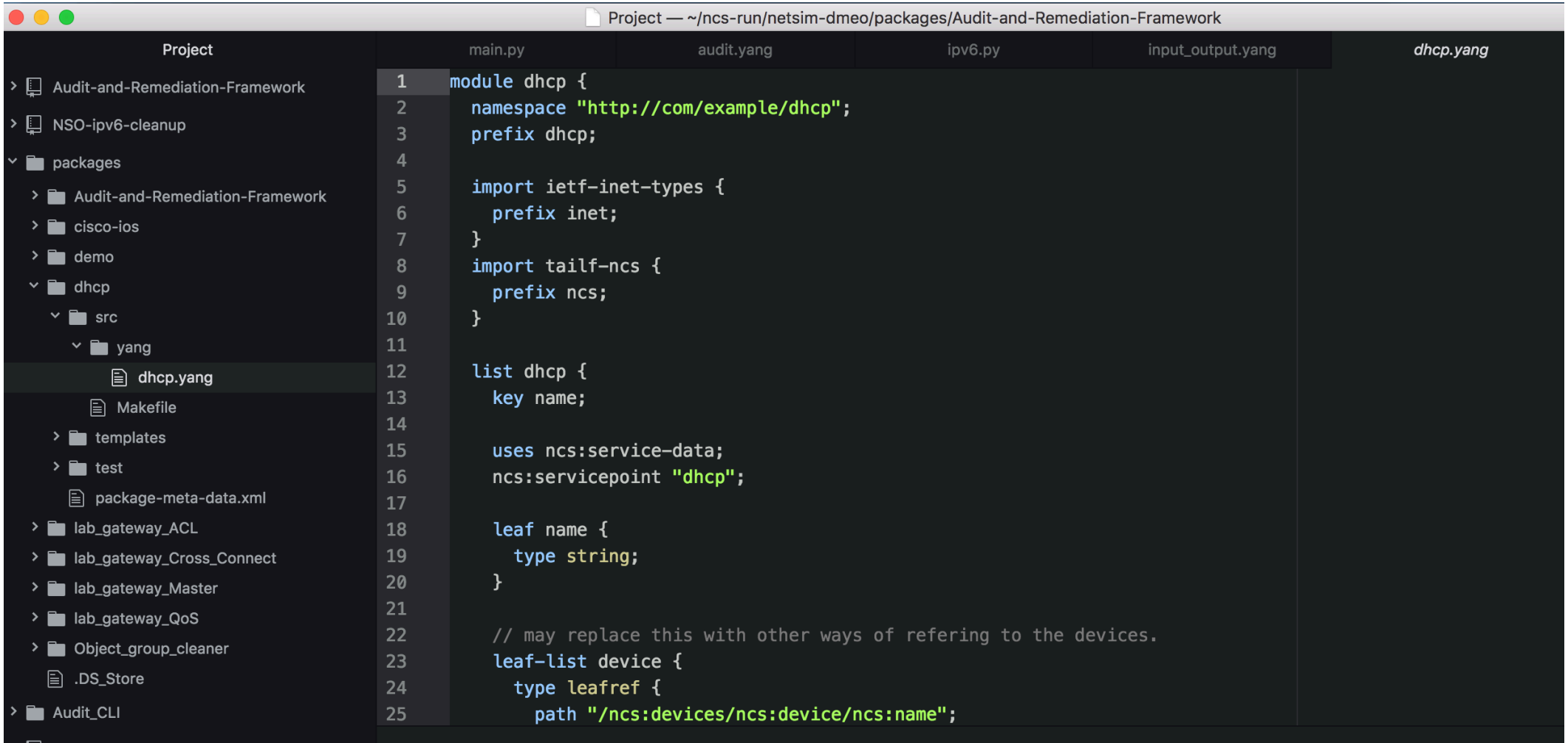
- Then we want to apply this to each inputted device

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 debuging and validation

Make YANG Model -- Example

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



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 refering 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):

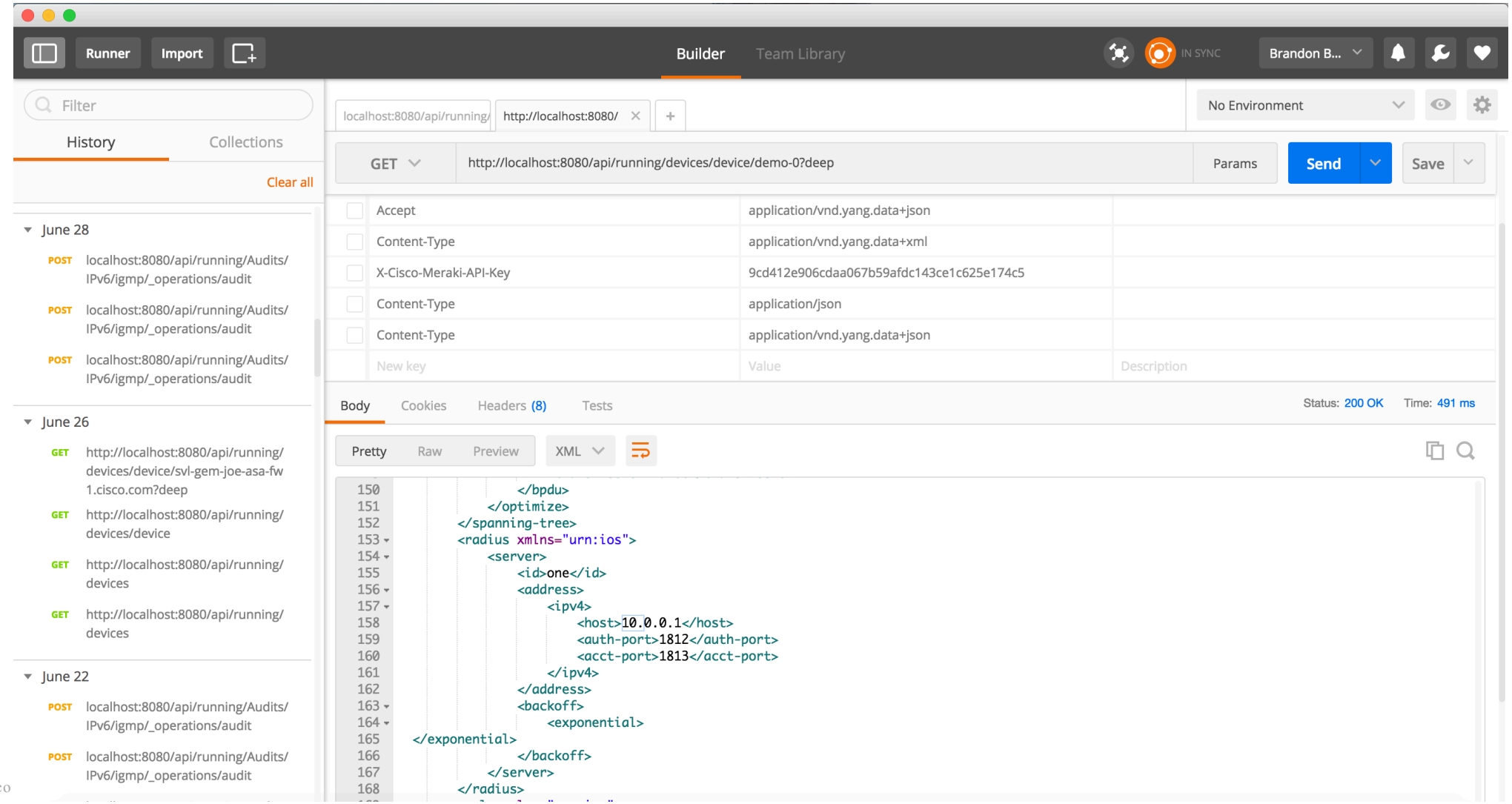
GET:

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

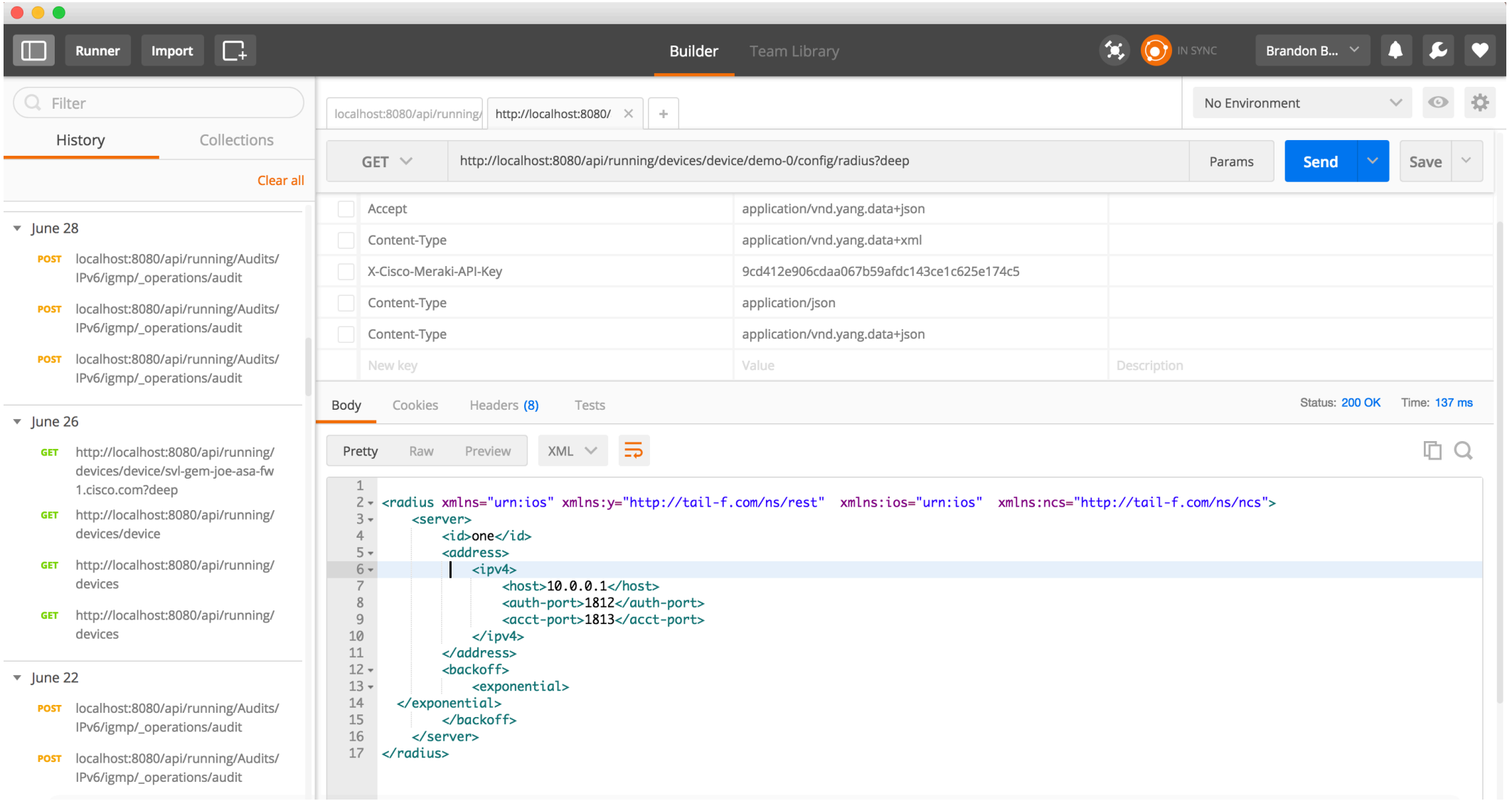
Will return the full config (may be big!) so we can filter it down via rest paths

Pick and choose the config you want!

Postman for getting XML config



Postman for getting filtered XML config



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 out 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

```
    <!--
        Select the devices from some data structure in the service
        model. In this skeleton the devices are specified in a leaf-list.
        Select all devices in that leaf-list:
    -->
    <name>{/device}</name>
    <config>
      <!--
          Add device-specific parameters here.
          In this skeleton the service has a leaf "dummy"; use that
          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="urn:ios"  xmlns:ncs="http://tail-f.com/ns/ncs">
          <server>
              <id>one</id>
              <address>
                  <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>
    </config>
  </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!

```
<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!

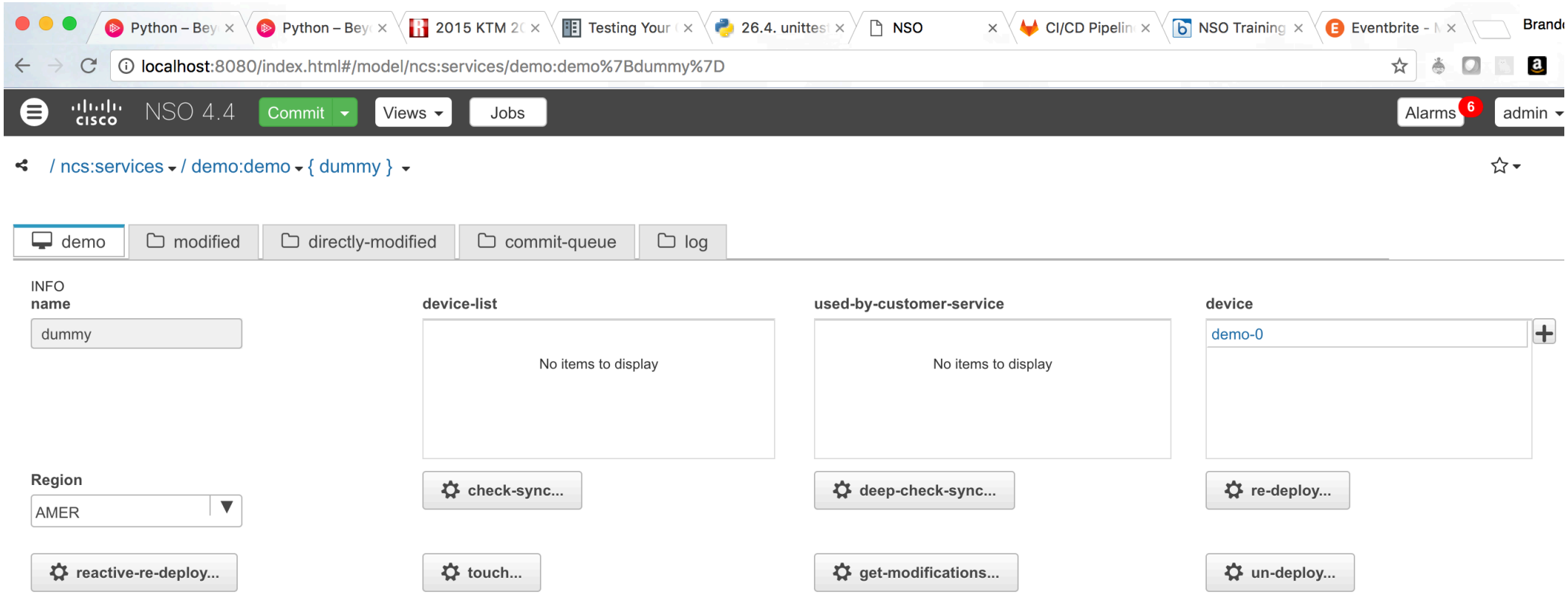
- We are now ready to compile and load our package!
- We can then test to ensure the service is deploying properly!
- To compile:
  - On the CLI, cd to the package directory's src folder and type: make
- To reload
  - Enter the nso cli and type "packages reload"
- Were now ready to test it out! (provided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

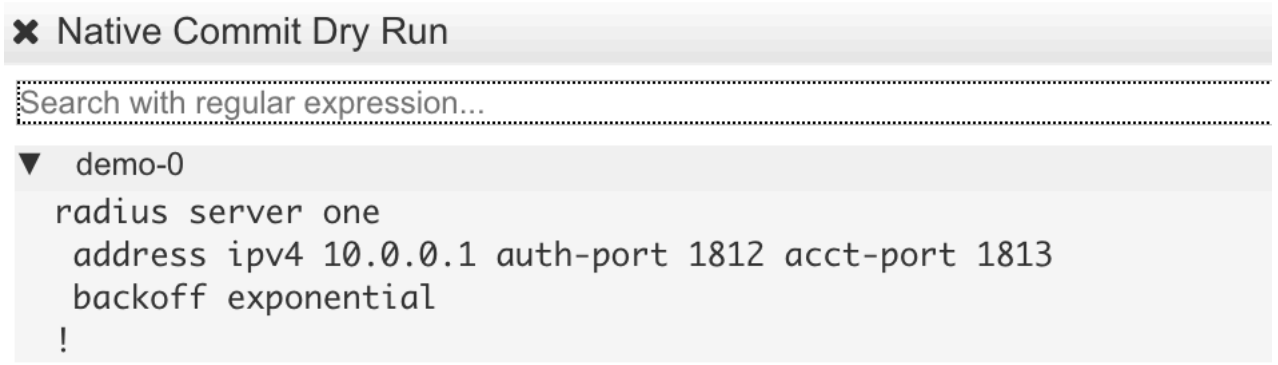
Test! --example

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



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 ☺



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