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

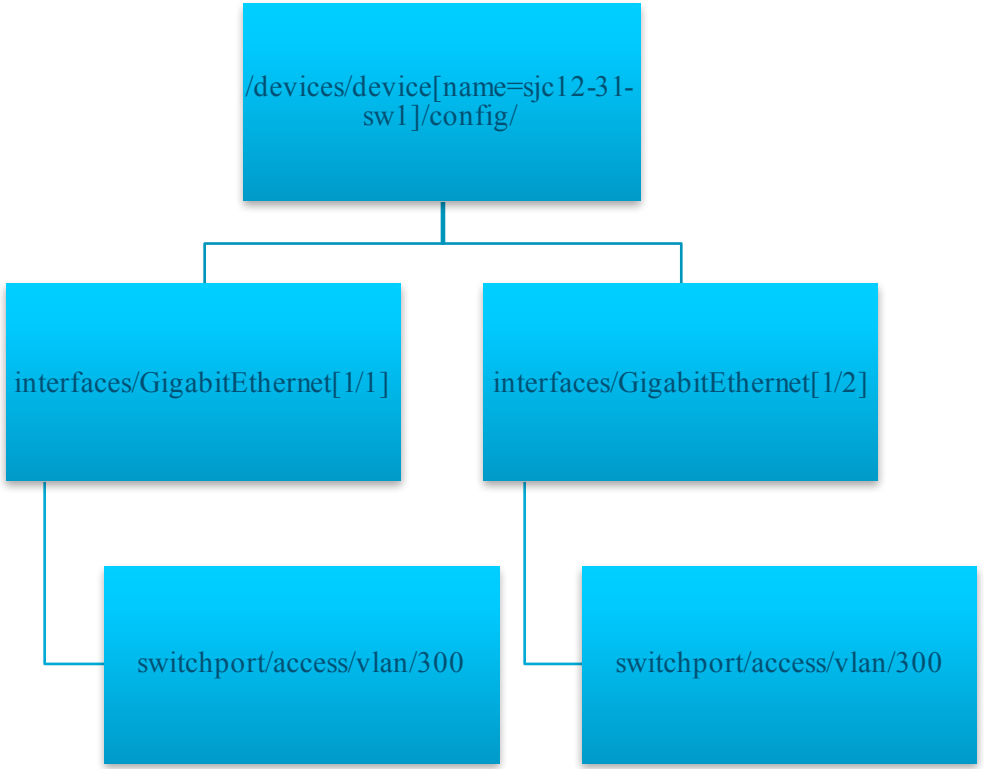
Current network automation within Cisco Network IT involves manipulating unstructured data through the Cisco CLI

```
import netmiko
ssh_con = netmiko.ConnectHandler(device_type='cisco_ios',  ip=" 10.1.1.1", username='userna,e',
password='webpassword'
ssh_con.find_prompt()
output = ssh_con.send_command("show  running-config")
```

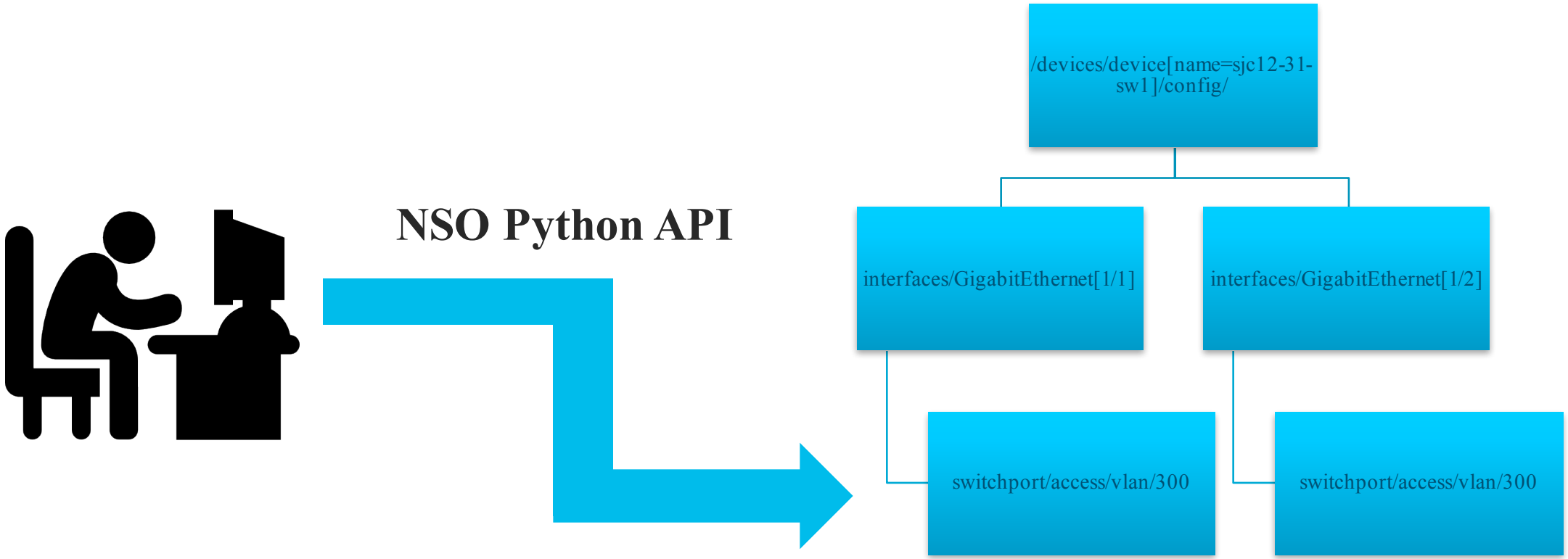
Use ciscoconfparse, TextFSM, or regular expressions to parse running config

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

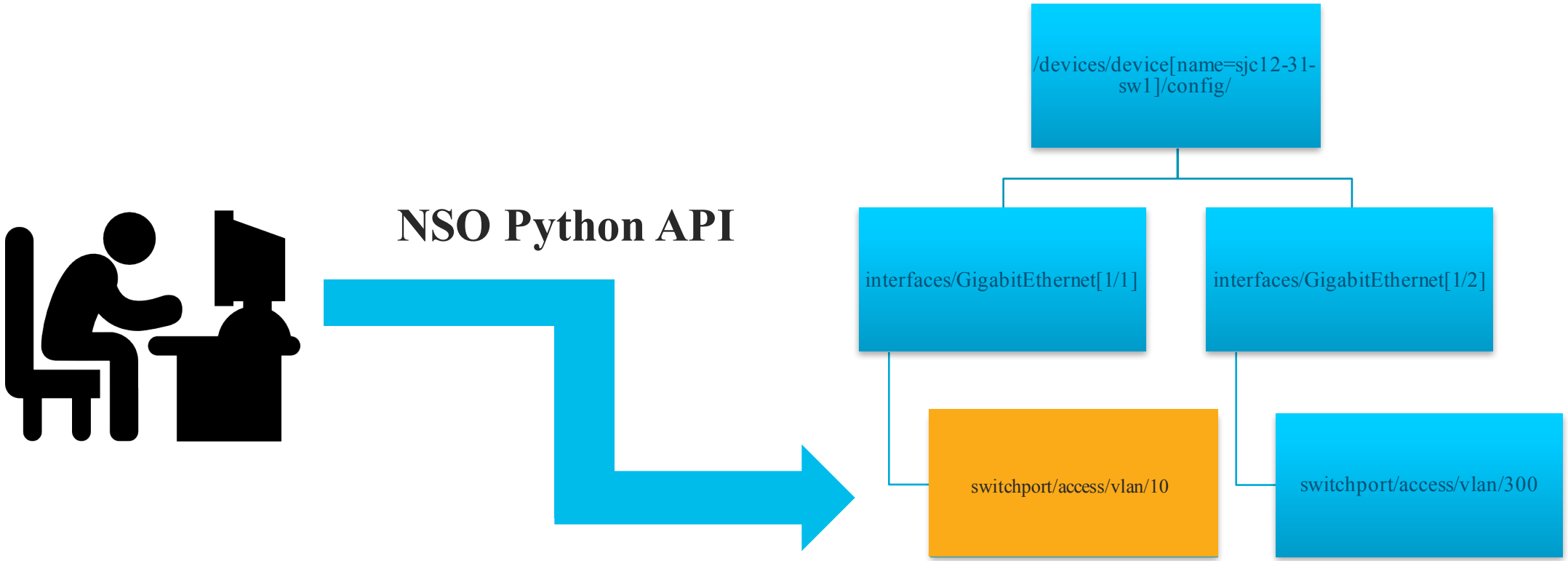


Python, YANG, and XML



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

Python, YANG, and XML

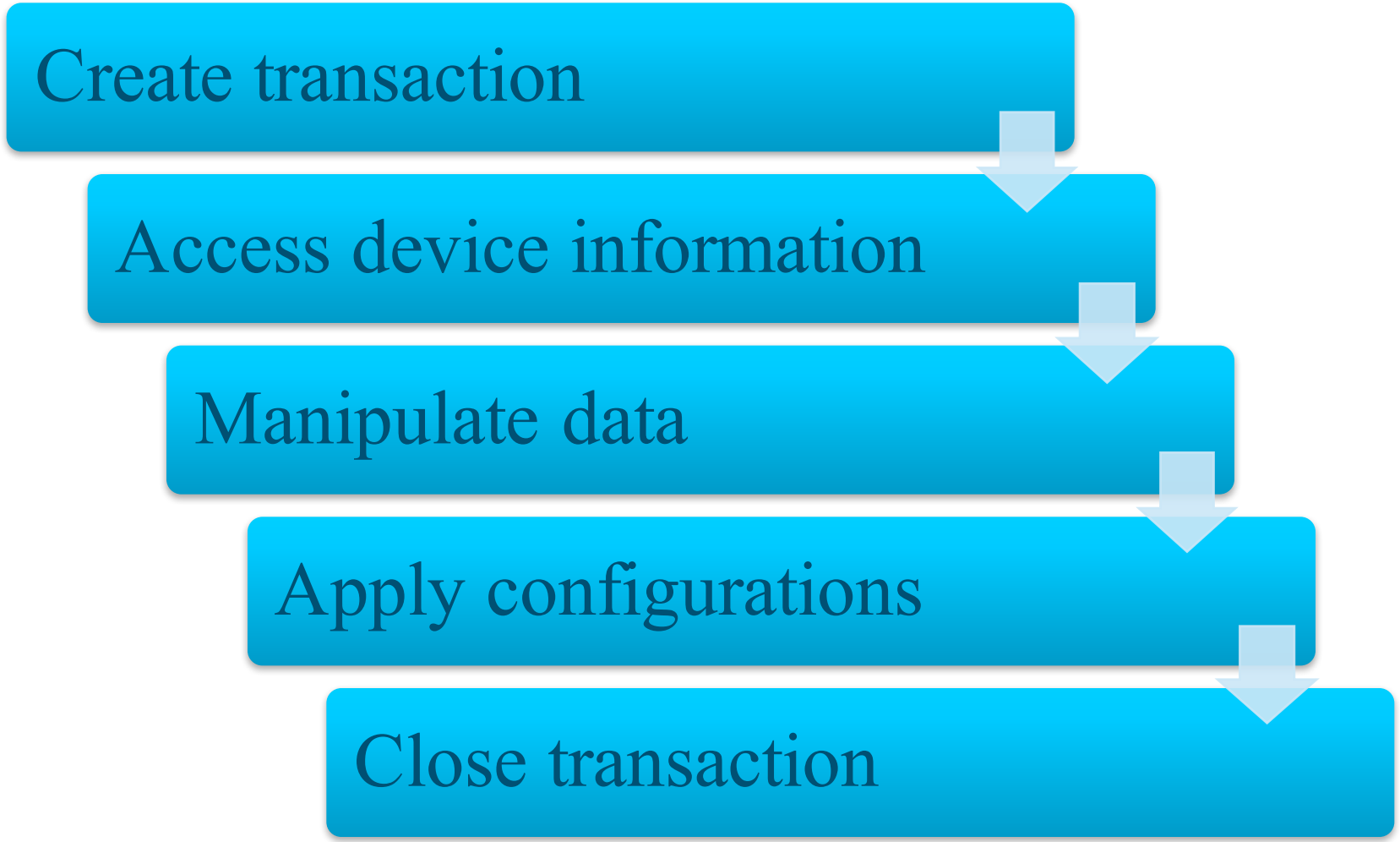


NSO Python API traverses the YANG tree and manipulates the data inside the XML

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()
```

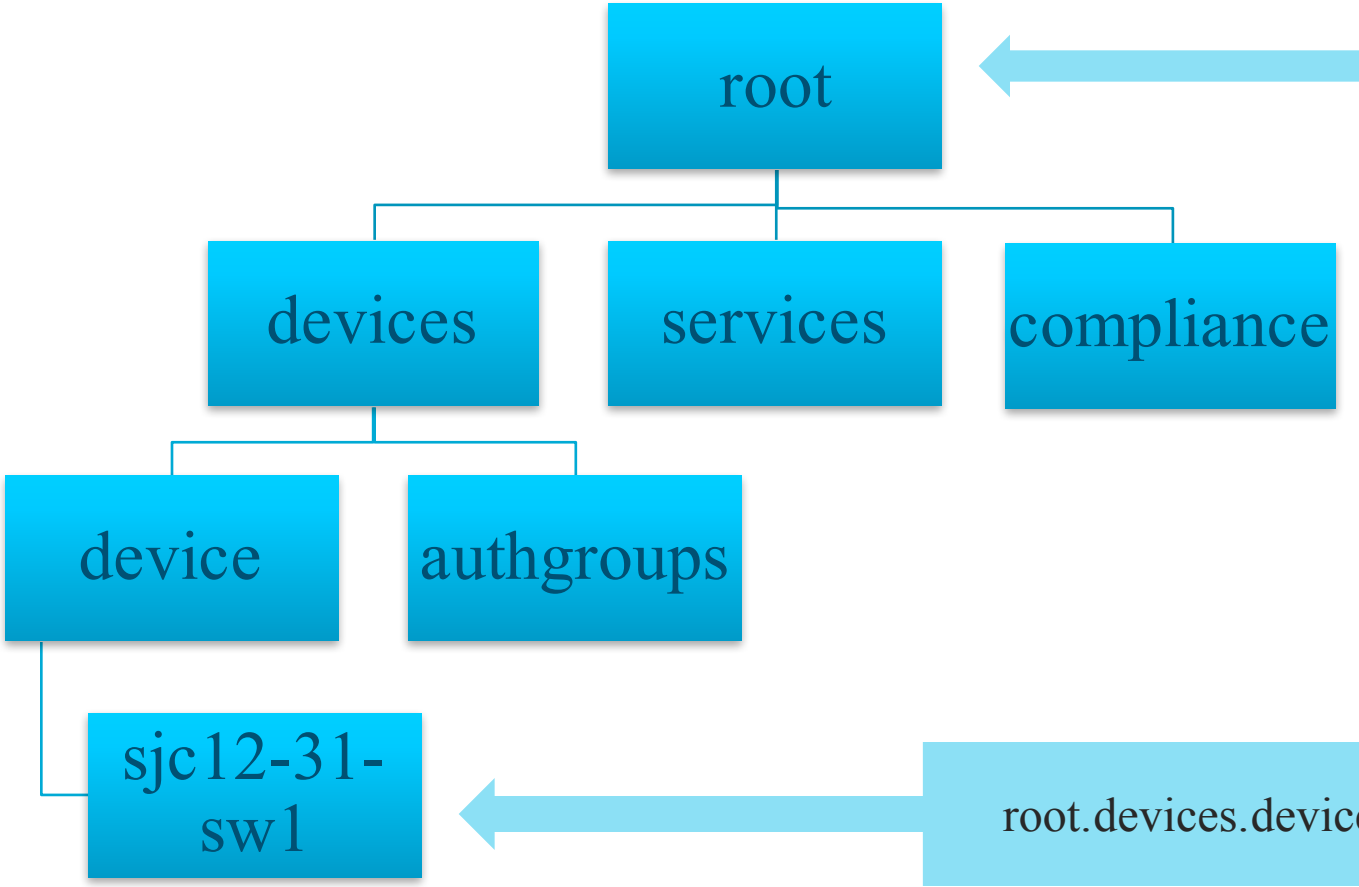
NSO Python Example

```
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



ncs.maagic.get_root(trans)

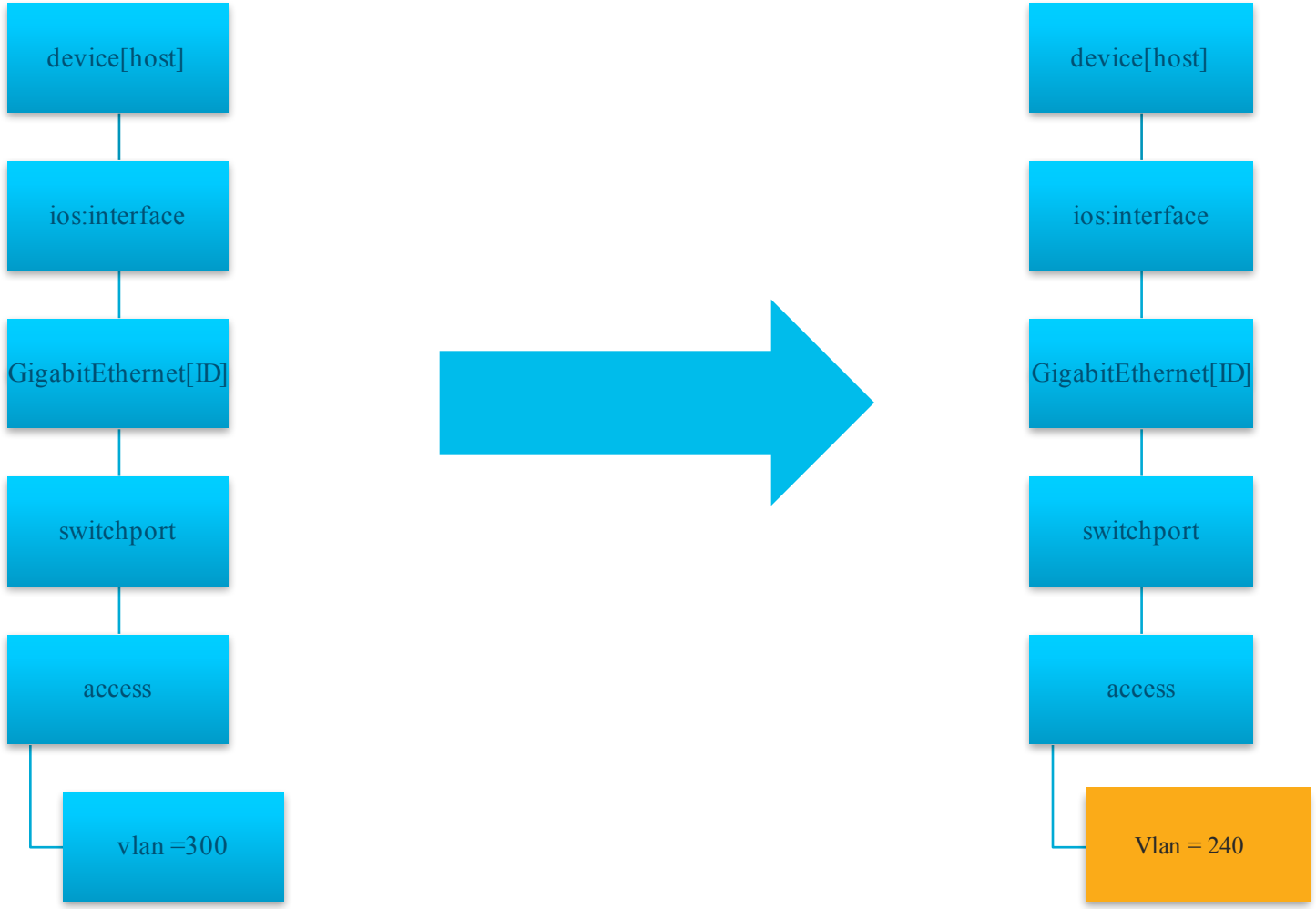
- You can traverse the YANG hierarchy by first going to the root of the NSO tree
- By using dot notation and indices, you can retrieve specific parameters

root.devices.device[“sjc12-31-sw1”]

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 Maapi session allows 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

- In order to read data from NSO, you need to open a transaction, either read or write.
- 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…

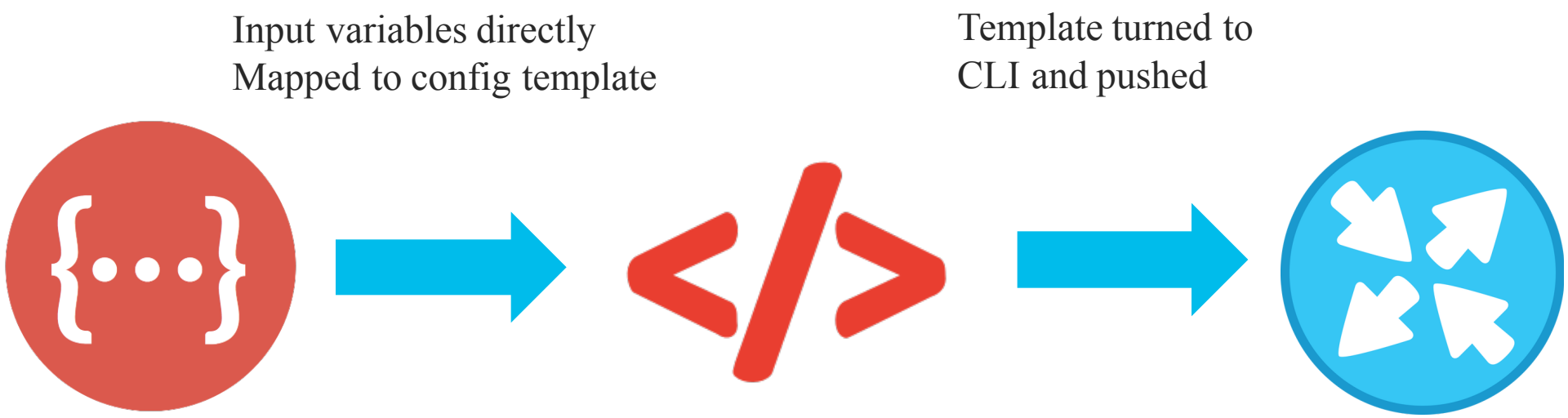
```
with ncs.maapi.Maapi() as m:
        with ncs.maapi.Session(m, 'admin', 'python'):
    root = ncs.maagic.get_root(m)
    device = root.devices.device[device_name]
    input1 = device.live_status.ios_stats__exec.show.get_input()
    input1.args = [command]
    output = device.live_status.ios_stats__exec.show(input1).result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

Lab 5

Services with Code

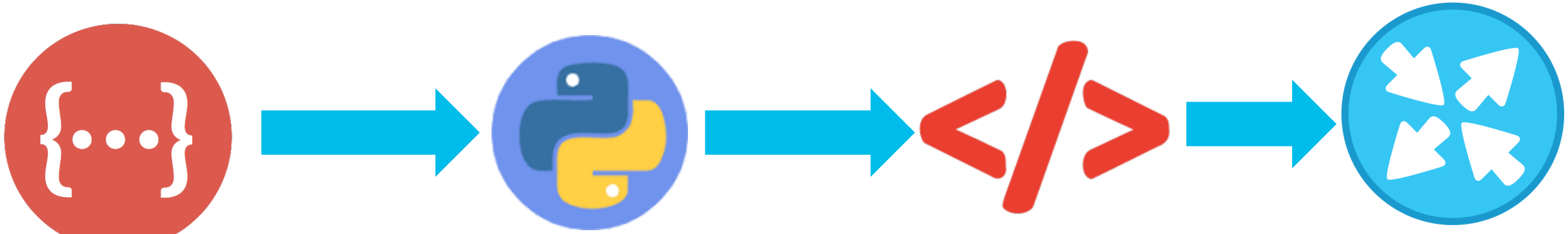
Developing Services with Code

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!

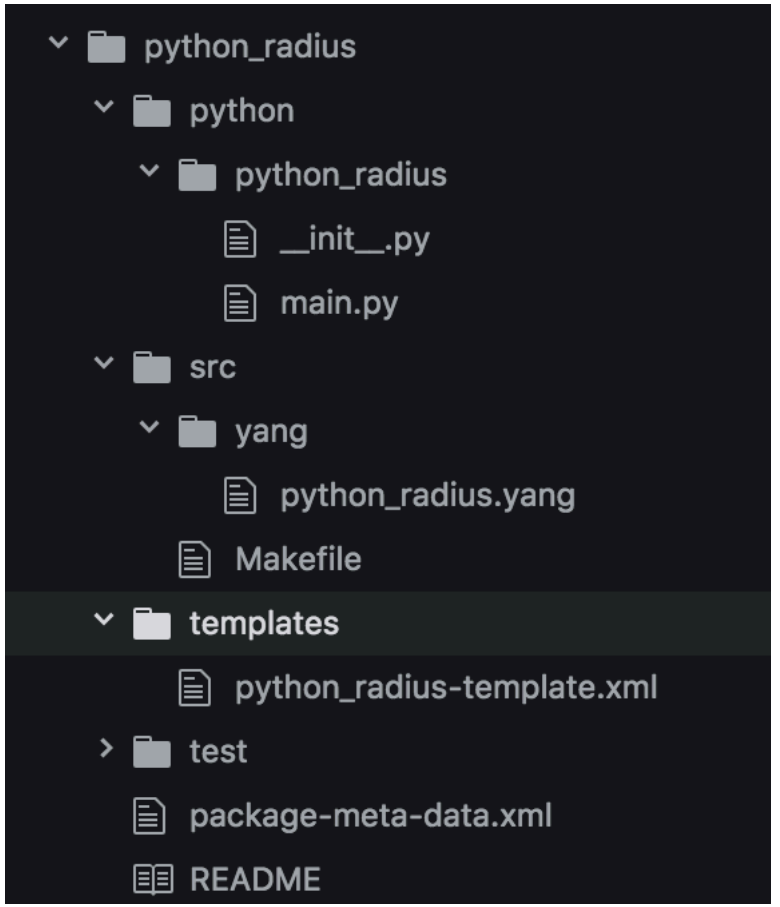
Were gonna jump in right here!

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 specificy 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 {/varaible_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!

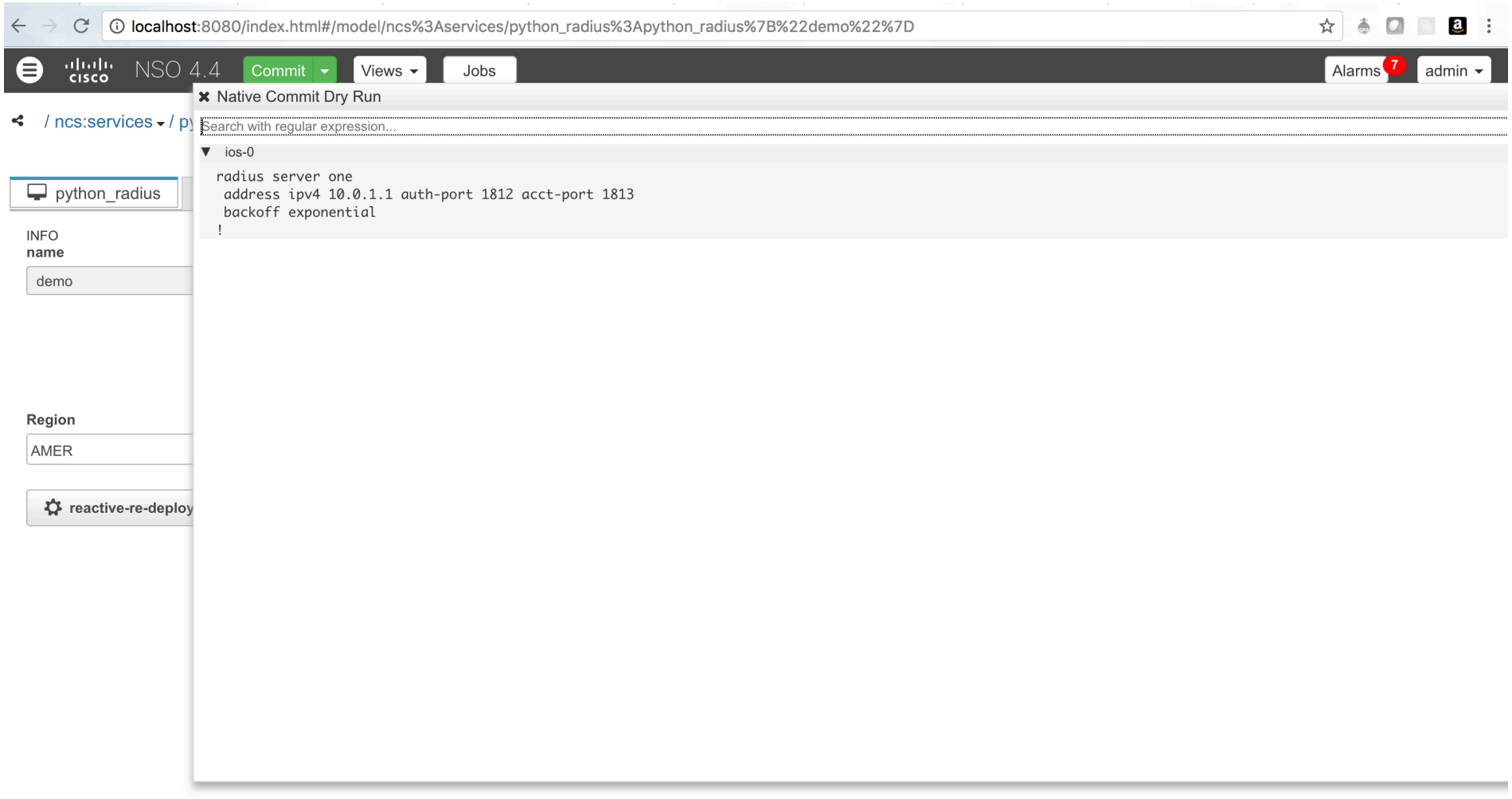
```
<name>{/device}</name>
<config>
  <!--
      Add device-specific parameters here.
      In this skeleton the, java code sets a variable DUMMY, use it
      to set something on the device e.g.:
      <ip-address-on-device>{$DUMMY}</ip-address-on-device>
  -->
  <radius xmlns="urn:ios" xmlns:y="http://tail-f.com/ns/rest"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



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)
```

Services with Code - Lab

Check out NSO Developer Hub: https://github.com/NSO-developer