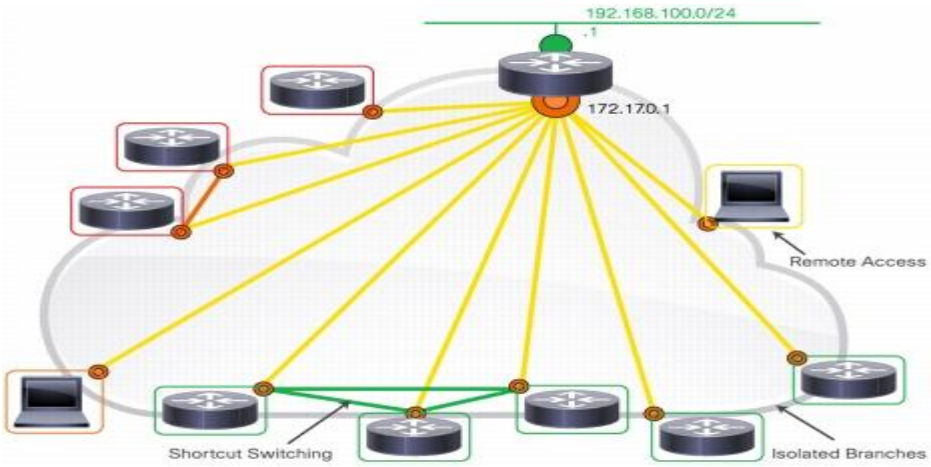


NX-OS Programming LAB

Abhinav Modi – Technical Marketing Engineer - Nexus
abmodi@cisco.com / Twitter: @abhinav_m

02/2016, Version 1.0

Introduction

This NX-OS Programming Lab is a self-paced lab launched in a virtualized environment.

The content of the lab at this time is focused on **NX-API** and Configuration Management with **Ansible**, with more to come!

Prerequisites

- Some familiarity with Python would be beneficial.
- Ansible familiarity will enable the candidate to dig deeper into Nexus configuration management aspects. However, all solutions have been provided to ensure that beginners get a feel of various aspects of configuration management.

Disclaimer

This training document is to familiarize with NX-API functionality and configuration. Although the lab design and configuration examples could be used as a reference, it's not a real design, thus not all recommended features are used, or enabled optimally. For the design related questions please contact your representative at Cisco, or a Cisco partner.

Related Sessions at Ciscolive

Other interesting sessions during Cisco Live on Programmability :

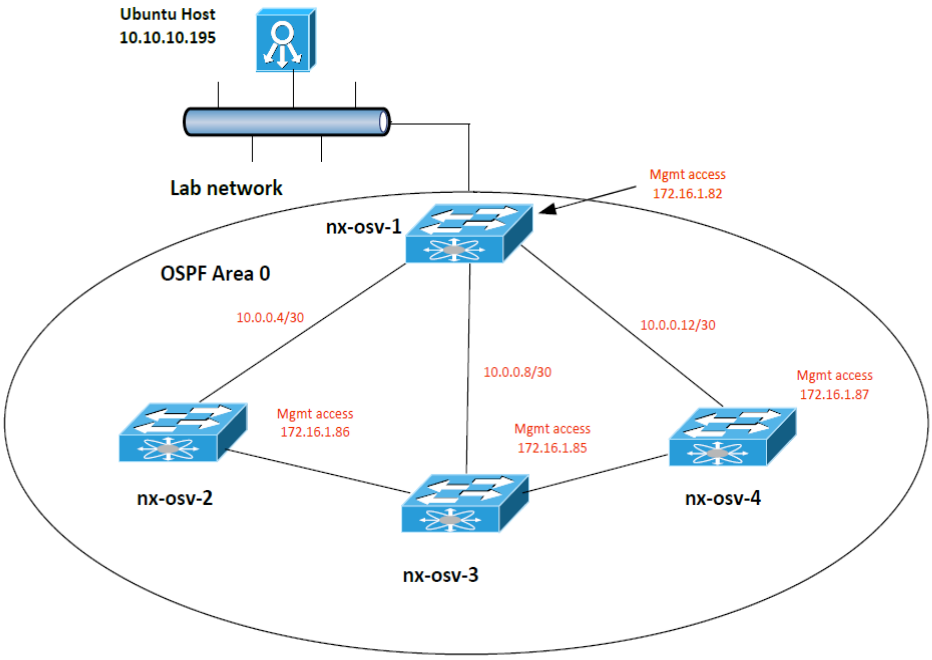
BRKDCT-2459
DEVNET-1075
DEVNET-1077

Table of Contents

Lab Topology and Addressing

Refer to access details and login credentials which were provided at the reception of the Walk-in Labs zone.

The following picture depicts the lab topology..



Note : This Lab guide is for REFERENCE ONLY. Hence, the connectivity details are redacted from this document.

The lab can be accessed via Putty once the Openconnect connection has been performed.

Putty connections can be done via :

1. Linux Host :

SSH to **10.10.10.195**
Username : <redacted>
Password : <redacted>

2. **NX-OSv Hosts : SSH with the credentials :**

   Username : cisco
   Password : cisco

   IP Addresses :

   Nx-osv-1 : 172.16.1.82
   Nx-osv-2 : 172.16.1.86
   Nx-osv-3 : 172.16.1.85
   Nx-osv-4 : 172.16.1.87

Lab 1 – Enabling NX-API and Exploring the Sandbox

NX-API is a base feature on NX-OS that is disabled by default.

```
nx-osv-1# show feature | i nxapi
nxapi                 1        disabled
nx-osv-1#
```

NX-API can be enabled as any other feature on the Nexus:

```
nx-osv-1# configure  terminal
Enter configuration commands, one per line.  End with CNTL/Z.
nx-osv-1(config)# feature nxapi
nx-osv-1(config)# end
```

With this, NX-API is now enabled and listening by default on HTTP Port 80.
Type in the following commands to verify:

```
nx-osv-1# show feature | i nxapi
```

```
nxapi                1        enabled
```

```
nx-osv-1# show nxapi

NX-API:      Enabled       Sandbox:     Disabled
HTTP Port:   80            HTTPS Port:  Disabled
```

As can be seen above, the Sandbox is disabled by default on NX-OSv and some of the Nexus platforms (Nexus 5xxx, Nexus 6000, and Nexus 7xxx series switches). To enable the sandbox, enter the following command:

```
nx-osv-1# configure  terminal
Enter configuration commands, one per line.  End with CNTL/Z.
nx-osv-1(config)# nxapi sandbox
nx-osv-1(config)#
```

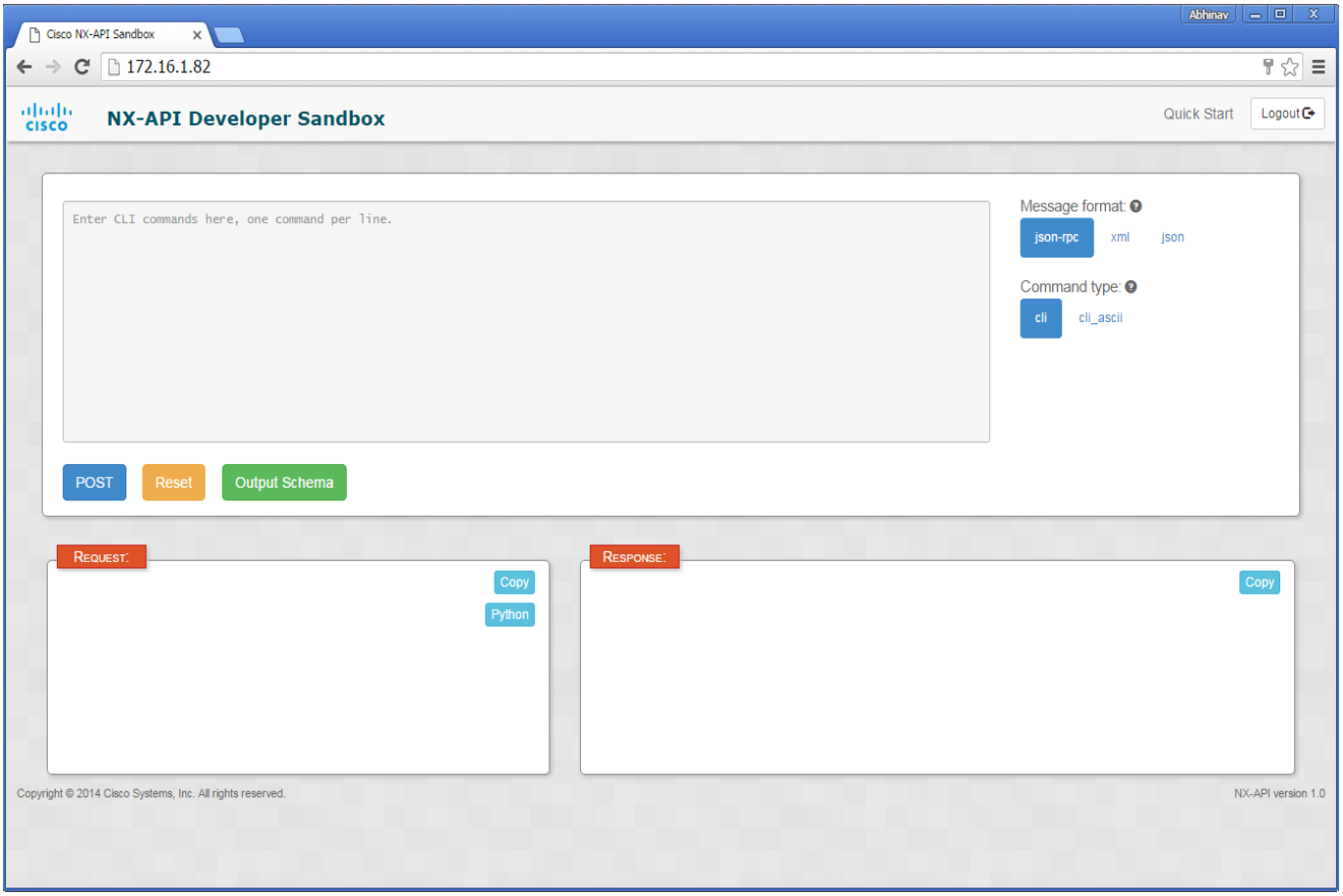
And verify:

```
nx-osv-1# show nxapi
NX-API:      Enabled       Sandbox:     Enabled
HTTP Port:   80            HTTPS Port:  Disabled
nx-osv-1#
```

We now have NX-API as well as the sandbox enabled.

Exploring the Sandbox

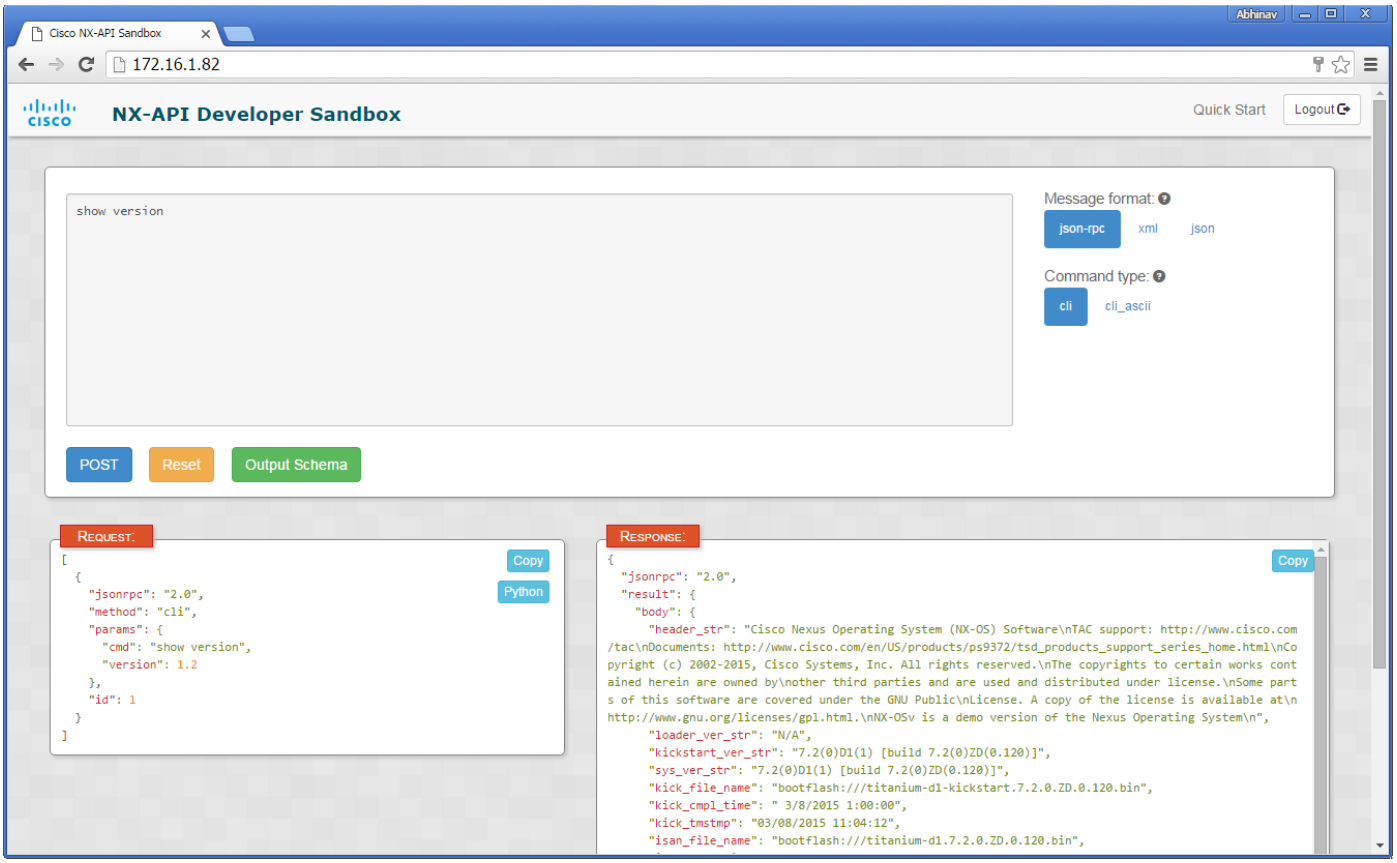
Open a browser and type in the Management IP. An interface similar to the one below will be available after passing in the credentials:



The Sandbox provides the following areas:

1. A text box to type in commands
2. Message Format Buttons to select how the input is formatted (Json/Json-rpc/xml)
3. Command type (Show command, configuration command, Ascii output)
4. The Post Button to send the request to the switch
5. The Request Area to see the request being sent
6. Finally, the Response Area to see the response from the switch

You can try a few commands to get familiar with the sandbox. A sample screenshot with “show version” being sent is provided below:



Lab 2 – Scripting with NX-API

The value of a programmatic interface such as NX-API is evident when we have scripts performing various actions from a management station. A typical use case would have the following steps:

- Log in to the switch(es) and collect operational data (show commands)
- Take specific actions based on the data, for example:
- Change Configuration
- Report status
- Generate logs, emails for alerting the Network Operations team
- Log the data for further analysis, time based statistics

The Anatomy of a Python NX-API script

A basic script will have the following:

The initial couple of lines are to import helper modules. In this case, we are using two modules that are essential as NX-API uses HTTP for communication (requests module) and JSON for the data interchange format (json module).

The “Requests” module is a library with functions that allow creating an HTTP request with appropriate Header and Body, and sending it across.

```
import requests
import json
```

The next few lines are to provide the IP Address/URL for the switch, followed by credentials to log onto the switch:

```
url='http://YOURIP/ins'
switchuser=’cisco’
switchpassword='cisco'
```

This is followed by the code to create the request to be sent.
The “myheaders” variable below contains the HTML headers that are required to be sent.

```
myheaders={'content-type':'application/json-rpc'}
```

The payload is the list of commands to be executed, along with other fields.

```
payload=[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show version",
      "version": 1.2
    },
    "id": 1
```

```
  }
]
```

The request is then sent via an HTTP POST call, and the return data available in the "response" variable

```
response=requests.post(url,data=json.dumps(payload),headers=myheaders,auth=(switchuser,switchpassword)).json()
```

This data can now be analyzed and appropriate action performed. In the below example, specific data from the show command output is extracted and printed on the screen:

As can be seen below, all the output in the show command is available as key-value pairs. In the below case, sys_ver_str is the attribute (key) that holds the NX-OS version string as value.

```
sys_ver = response['result']['body']['sys_ver_str']
print ("The Software Version is: " + sys_ver)
```

A script is available on the linux host (~/hello.py) for execution.

Lab Tasks:

1. Try running the script (**python ~/hello.py**) and getting the output for the image string.
2. Modify the script to get the output of two other strings of your choice
3. Modify the script to print out all the available data (keys), one key/value pair per line.

Lab 3 – Overview of Configuration Management and Ansible

Configuration Management Tools are a class of software that enables administrators to deploy and maintain a large number of devices. This originally spawned in the server space, where sysadmins needed to deploy, manage, upgrade hundreds of servers. To help with these tasks, home grown scripts gave place to configuration management software such as Puppet, Chef, and Ansible among others.

At a very high level, each of these open source tools provides :

- A "Master" program which runs on a server and manages all devices
- Infrastructure to manage information about various devices that need to be managed

- Libraries and Modules, often open source, that perform specific tasks (eg : upgrade an image, configure vlan)
- Optionally, an Agent software running on each managed device, which talks to the Master.

In case of Ansible, there is no requirement of Agent running on the managed device, but there is a requirement that the managed device have Python and SSH connectivity to the Master.

Ansible and NX-OS : Managing Nexus Switches

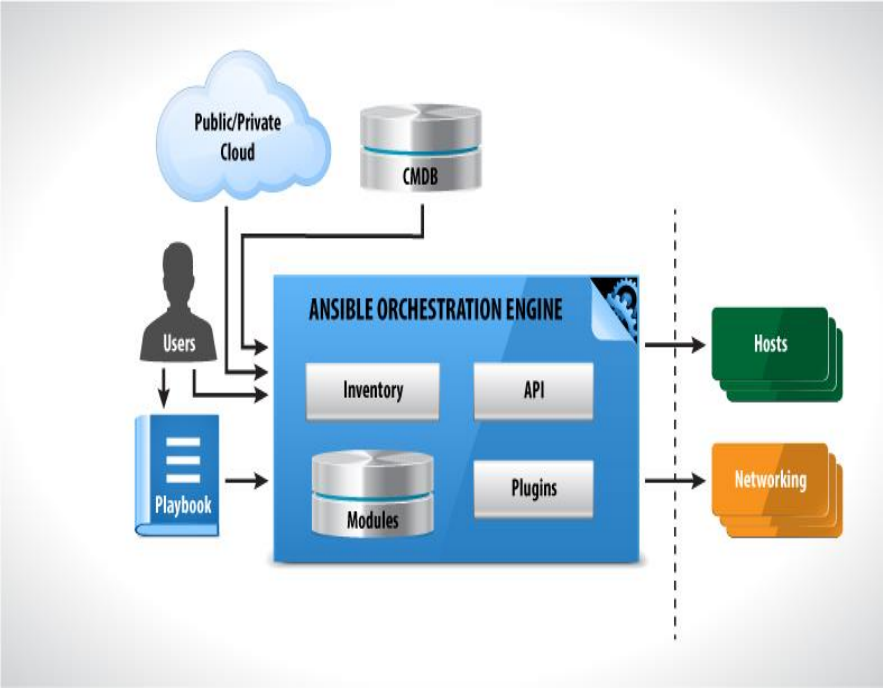
As mentioned above, Ansible typically requires Python and SSH to be enabled on the managed device.

However, in case of many networking devices, the communication is done via APIs provided by the NOS.

In case of NX-OS, we will use Ansible with NX-API, which was just explained in the previous lab.

Ansible in 10 minutes!!

Here is a crash course on Ansible



In the diagram above, the following are the key components

- The Ansible Orchestration Engine (A Server node that runs Ansible and manages other devices)
- The Managed nodes (Hosts). The Server talks to these nodes via SSH, or in case of NX-OS, NX-API

The following scripts / files are used to drive the process:

- The playbook, which contains the tasks that the user wants to do.
- The modules, installed on the Server, which provide library functions for a particular task
- The Inventory file, which provides the list of hosts (or managed devices, in our case switches) that the playbook could be potentially run against.

How to Instruct Ansible : Details

The format for playbooks : YAML

YAML is a data format which is very human readable. It is used for presenting data in a manner that both machines and humans can process.

The following in an example screenshot of a YAML file :

ping_playbook.yml

```
1  ---
2  - hosts: webservers
3    remote_user: ansible
4    tasks:
5      - name: test connection
6        ping:
```

The YAML file starts with three hyphens (---). Indentation is important and can be used to create nested structures.

The complexity of the actions is hidden behind libraries, known in Ansible parlance as modules. These modules, often open source, can be used in playbooks to perform various tasks such as intstall a package on a linux machine, upgrade a package, or in case of switches, configure a feature such as interface, vlan etc.

There are a set of **core modules** which ship with the Ansible distribution and are installed automatically. In addition, there are hundreds of modules available thanks to the developer community, which can be additionally installed and utilized.

Each **playbook (as above)** has a set of **tasks** which are executed against all mentioned **hosts (devices).**

A master list of hosts if kept in an **inventory file** and referenced in a call to Ansible.

To summarize, the following are the key items that are needed:

1. Inventory file containing list of devices, often grouped, which Ansible can reference.
2. A playbook which contains a list of tasks that need to be performed to various hosts

That's it. The playbook is run using the CLI, (ansible-playbook), with the above two parameters.

Introducing nxos-ansible

nxos-ansible is an open source module, developed by Jason Edelman, to interact with Nexus and provision it.

Nxos-ansible is available on Github at https://github.com/jedelman8/nxos-ansible

Nxos-ansible provides modules which can be used to configure and get data from Cisco Nexus Switches.

Following is a snippet:

Please Change the directory as below :

cd home/abhinav/gitprojs/ansiblej/nxos-ansible

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible$ ansible-playbook -v -i hosts nexus_ping.yml


PLAY [Ping Check] ***************************************************************


TASK: [Check whether Ping to internet works from each switch] *****************
ok: [nx-osv-1] => {"action": "PING 8.8.8.8 (8.8.8.8): 56 data bytes", "changed": false,
"command": "ping 8.8.8.8 count 4 vrf management", "count": 4, "dest": "8.8.8.8", "rtt": {"avg":
"37.173", "max": "40.848", "min": "34.571"}, "state": "absent", "summary": {"packet_loss":
"0.00%", "packets_rx": "4", "packets_tx": "4"}}
ok: [nx-osv-4] => {"action": "PING 8.8.8.8 (8.8.8.8): 56 data bytes", "changed": false,
"command": "ping 8.8.8.8 count 4 vrf management", "count": 4, "dest": "8.8.8.8", "rtt": {"avg":
```

```
"37.119", "max": "40.891", "min": "34.523"}, "state": "absent", "summary": {"packet_loss":
"0.00%", "packets_rx": "4", "packets_tx": "4"}}
ok: [nx-osv-3] => {"action": "PING 8.8.8.8 (8.8.8.8): 56 data bytes", "changed": false,
"command": "ping 8.8.8.8 count 4 vrf management", "count": 4, "dest": "8.8.8.8", "rtt": {"avg":
"36.294", "max": "37.029", "min": "35.188"}, "state": "absent", "summary": {"packet_loss":
"0.00%", "packets_rx": "4", "packets_tx": "4"}}
ok: [nx-osv-2] => {"action": "PING 8.8.8.8 (8.8.8.8): 56 data bytes", "changed": false,
"command": "ping 8.8.8.8 count 4 vrf management", "count": 4, "dest": "8.8.8.8", "rtt": {"avg":
"38.379", "max": "42.332", "min": "35.552"}, "state": "absent", "summary": {"packet_loss":
"0.00%", "packets_rx": "4", "packets_tx": "4"}}


PLAY RECAP
********************************************************************
nx-osv-1                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-2                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-3                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-4                   : ok=1    changed=0    unreachable=0    failed=0


abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible$
```

The above is a sample playbook which runs on the lab topology, initiates a ping from each switch using **nxos_ping** module, and returns success or failure.

The list of all modules is available in the nxos_ansible documentation here.

In case a specific module for a particular component does not exist, **nxos_command** module can be used to push CLIs directly via NX-API. This provides and interim solution until the module is developed.

In addition, in the spirit of open source, users are welcome to develop modules and submit a pull request on github to add the functionality to the module.

Checking the Results of a Playbook Execution :

The “-v” option is for verbosity, and useful for debugging purposes. For even more verbose output, “-vv” or “-vvv” can be used.

For this lab, we will be using the debug flag and basic templates (another item to learn !) to verify that the activity returned proper output.

There are better mechanisms to store this output in variables, write them out to files, etc, that we will not cover in detail for this introductory lab.

Lab Task 1: Gather Operational Data (CDP and Route Table)

Operational Data can be obtained using "show" commands on the Nexus. This lab will get the output for two commands :

1. CDP Neighborship (show cdp neighbors)
2. Route Table for Management vrf (show ip route vrf management)

Nxos-ansible library has a module for getting CDP data (**nxos_get_neighbors).** This will be used to execute the CLI and get the response.

The route table will be obtained using the **nxos_command** module.

Once this data is obtained, each task will also store the data in a variable (known as **register**) in Ansible parlance.

Finally, the registers will printed out into a file (one file per host) using **Jinja2 templates.**

Let's go through these in a step-wise fashion

1. The Playbook :

The playbook gather_data.yml consists of three tasks:

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible$ more gather_data.yml
---


- name: gather data
  hosts: all
  connection: local
  gather_facts: no



  tasks:
```

```
- name: get neighbors
  nxos_get_neighbors: type=cdp host={{ inventory_hostname }}
  register: my_neighbors
```

The above task uses the module **nxos_get_neighbors** to obtain CDP data from the host. Note that "**type=cdp"** is passed to the module.

In addition, **"register"** or variable *my_neighbors* is used to store the results of the task.

```
- name: get routing table for mgmt VRF
  nxos_command:
    type: show
    host: "{{ inventory_hostname }}"
    command: "show ip route vrf management"
  register: my_routes
```

Similarly, the above task uses **nxos_command** to execute the show command and get the data, which is stored in the register *my_routes.*

```
- name: Store to File
  template: src=templates/data.j2 dest=data/{{ inventory_hostname }}_gather_data.json
```

Finally, the above task uses the core Ansible module, template, to write data to the filesystem.

The template module takes in a src file, which is a Jinja2 template that specifies what to output. This can be a mix of raw text and register variables.

The module also requires a dest keyword, to specify the destination file. In the above case, the destination is a file called <hostname>_gather_data.json in the *data* directory. This ensures that each host has its own file (nx-osv-1_gather_data.json , nx-osv-2_gather_data.json, etc).

The template file data.j2 consists of:

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ more templates/data.j2
CDP Info:
{{ my_neighbors | to_nice_json }}
```

```
Route Info:
{{ my_routes | to_nice_json }}
```

As can be seen above, the file has some static text (**CDP Info)** followed by the line

```
{{ my_neighbors | to_nice_json }}
```

This signifies that the register my_neighbors is to be output, after converting to a pretty-printed JSON format.

Ansible takes care of doing this for each host for which the playbook is run.

The output of the playbook will be similar to below :

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ ansible-playbook -i hosts gather_data.yml

PLAY [get neighbor data] *******************************************************

TASK: [get neighbors] **********************************************************
ok: [nx-osv-4]
ok: [nx-osv-1]
ok: [nx-osv-3]
ok: [nx-osv-2]

TASK: [get routing table for mgmt VRF] *****************************************
ok: [nx-osv-1]
ok: [nx-osv-4]
ok: [nx-osv-3]
ok: [nx-osv-2]

TASK: [Store to File] **********************************************************
changed: [nx-osv-4]
changed: [nx-osv-2]
changed: [nx-osv-1]
```

```
changed: [nx-osv-3]


PLAY RECAP
********************************************************************
nx-osv-1                   : ok=3    changed=1    unreachable=0    failed=0
nx-osv-2                   : ok=3    changed=1    unreachable=0    failed=0
nx-osv-3                   : ok=3    changed=1    unreachable=0    failed=0
nx-osv-4                   : ok=3    changed=1    unreachable=0    failed=0
```

The data directory now contains :

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ ls -1 data/
nx-osv-1_gather_data.json
nx-osv-2_gather_data.json
nx-osv-3_gather_data.json
nx-osv-4_gather_data.json
```

Open each file to note that the data is consistent with that displayed using the CLI on the respective node.

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ more data/nx-osv-1_gather_data.json
CDP Info:
{
    "changed": false,
    "invocation": {
        "module_args": "type=cdp host=nx-osv-1",
        "module_name": "nxos_get_neighbors"
    },
    "resource": [
        {
            "local_interface": "mgmt0",
            "neighbor": "nx-osv-3",
```

```
        "neighbor_interface": "mgmt0",
        "platform": "N7K-C7018"
    },
    {
        "local_interface": "mgmt0",
        "neighbor": "nx-osv-4",
        "neighbor_interface": "mgmt0",
        "platform": "N7K-C7018"
    },
    {
        "local_interface": "mgmt0",
        "neighbor": "nx-osv-2",
        "neighbor_interface": "mgmt0",
        "platform": "N7K-C7018"
    },
    {
        "local_interface": "Ethernet2/2",
        "neighbor": "nx-osv-3",
        "neighbor_interface": "Ethernet2/1",
        "platform": "N7K-C7018"
    },
    {
        "local_interface": "Ethernet2/3",
        "neighbor": "nx-osv-1",
        "neighbor_interface": "Ethernet2/3",
        "platform": "N7K-C7018"
    },
    {
        "local_interface": "Ethernet2/3",
        "neighbor": "nx-osv-4",
        "neighbor_interface": "Ethernet2/1",
        "platform": "N7K-C7018"
```

```
        }
    ]
}

Route Info:
{
    "changed": false,
    "commands": "show ip route vrf management",
    "invocation": {
        "module_args": "",
        "module_name": "nxos_command"
    },
    "proposed": {
        "cmd_type": "show",
        "command": "show ip route vrf management"
    },
    "results": {
        "TABLE_vrf": {
            "ROW_vrf": {
                "TABLE_addrf": {
                    "ROW_addrf": {
                        "TABLE_prefix": {
                            "ROW_prefix": [
                                {
                                    "TABLE_path": {
                                        "ROW_path": {
                                            "clientname": "static",
                                            "hidden": "false",
                                            "ipnexthop": "172.16.1.254",
                                            "metric": "0",
                                            "pref": "1",
                                            "stale": "false",
```

```
                    "stale-label": "false",
                    "ubest": "true",
                    "unres": "false",
                    "uptime": "P1M13DT2H45M55S"
                }
            },
            "attached": "false",
            "ipprefix": "0.0.0.0/0",
            "mcast-nhops": "0",
            "ucast-nhops": "1"
        },
… (output truncated intentionally for lab guide)
```

Lab 4 : Configuring Interfaces using Ansible

This final lab will focus on two key items:

1. Configuration tasks on Nexus using Ansible modules
2. The concept of **idempotence** which is key to most configuration management tools

What is Idempotence ?

Simply put, Idempotence in the context of configuration management/Ansible indicates that a **change of state / change of configuration** will **only be performed if required.**

In other words, if the entity is already in the ***desired*** *state***,** then the configuration will not be applied.

Ansible modules are hence written in a way to indicate desired state. For example:

```
  - nxos_interface: interface=Ethernet2/4 description='Configured by Ansible' mode=layer3 host={{ inventory_hostname }}
```

In the above, the task is to ***ensure*** that Ethernet 2/4 has the description mentioned and is in L3 mode.

Hence, the module **nxos_interface**, being idempotent, will **first check the description and mode using show commands** (show interface ..).

If they match, no action is required and hence the task will return “ok” without making any change (the summary will have “changed=0”)

On the other hand, if the description / mode are different, only then the module will go ahead and make the change.

This ensures that only **required** changes are performed, and unnecessary commands that result in no-operations are omitted.

A simple way to see this in action is to run the same playbook **twice.** If it is idempotent, the first run will change the feature to the desired state, and the next run will result in a no-operation.

Let’s see this using a single task mentioned above :

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ more idempotence.yml
---


- name: Playbook to demonstrate idempotence
  hosts: all
  connection: local
  gather_facts: no


  tasks:


    - nxos_interface: interface=Ethernet2/4 description='ABCDEFGH' mode=layer3 host={{
inventory_hostname }}
```

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ ansible-playbook -i hosts -v idempotence.yml


PLAY [Playbook to demonstrate idempotence] *************************************
```

```
TASK: [nxos_interface interface=Ethernet2/4 description='ABCDEFGH' mode=layer3
host={{inventory_hostname}}] ***

changed: [nx-osv-3] => {"changed": true, "commands": "interface ethernet2/4 ; description
ABCDEFGH ;", "existing": {"admin_state": "up", "description": "Not Configured by Ansible",
"duplex": "auto", "mode": "layer3", "speed": "auto"}, "final": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "interface": "Ethernet2/4", "mac_address":
"0000.0000.002f", "mode": "layer3", "speed": "auto", "state": "down", "type": "ethernet"},
"proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode": "layer3"}, "state":
"present"}

changed: [nx-osv-1] => {"changed": true, "commands": "interface ethernet2/4 ; description
ABCDEFGH ;", "existing": {"admin_state": "up", "description": "Not Configured by Ansible",
"duplex": "auto", "mode": "layer3", "speed": "auto"}, "final": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "interface": "Ethernet2/4", "mac_address":
"0000.0000.002f", "mode": "layer3", "speed": "auto", "state": "down", "type": "ethernet"},
"proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode": "layer3"}, "state":
"present"}

changed: [nx-osv-4] => {"changed": true, "commands": "interface ethernet2/4 ; description
ABCDEFGH ;", "existing": {"admin_state": "up", "description": "Not Configured by Ansible",
"duplex": "auto", "mode": "layer3", "speed": "auto"}, "final": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "interface": "Ethernet2/4", "mac_address":
"0000.0000.002f", "mode": "layer3", "speed": "auto", "state": "down", "type": "ethernet"},
"proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode": "layer3"}, "state":
"present"}

changed: [nx-osv-2] => {"changed": true, "commands": "interface ethernet2/4 ; description
ABCDEFGH ;", "existing": {"admin_state": "up", "description": "Not Configured by Ansible",
"duplex": "auto", "mode": "layer3", "speed": "auto"}, "final": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "interface": "Ethernet2/4", "mac_address":
"0000.0000.002f", "mode": "layer3", "speed": "auto", "state": "down", "type": "ethernet"},
"proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode": "layer3"}, "state":
"present"}


PLAY RECAP
********************************************************************

nx-osv-1                   : ok=1    changed=1    unreachable=0    failed=0

nx-osv-2                   : ok=1    changed=1    unreachable=0    failed=0

nx-osv-3                   : ok=1    changed=1    unreachable=0    failed=0

nx-osv-4                   : ok=1    changed=1    unreachable=0    failed=0


abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫
```

Now running the same playbook **again** results in the "**changes**" being 0 and the playbook exiting successfully

```
abhinav@ubuntu:~/gitprojs/ansiblej/nxos-ansible⟫ ansible-playbook -i hosts -v idempotence.yml


PLAY [Playbook to demonstrate idempotence] ************************************


TASK: [nxos_interface interface=Ethernet2/4 description='ABCDEFGH' mode=layer3
host={{inventory_hostname}}] ***
ok: [nx-osv-4] => {"changed": false, "commands": "", "existing": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "mode": "layer3", "speed": "auto"}, "final":
{"admin_state": "up", "description": "ABCDEFGH", "duplex": "auto", "mode": "layer3",
"speed": "auto"}, "proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode":
"layer3"}, "state": "present"}
ok: [nx-osv-1] => {"changed": false, "commands": "", "existing": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "mode": "layer3", "speed": "auto"}, "final":
{"admin_state": "up", "description": "ABCDEFGH", "duplex": "auto", "mode": "layer3",
"speed": "auto"}, "proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode":
"layer3"}, "state": "present"}
ok: [nx-osv-3] => {"changed": false, "commands": "", "existing": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "mode": "layer3", "speed": "auto"}, "final":
{"admin_state": "up", "description": "ABCDEFGH", "duplex": "auto", "mode": "layer3",
"speed": "auto"}, "proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode":
"layer3"}, "state": "present"}
ok: [nx-osv-2] => {"changed": false, "commands": "", "existing": {"admin_state": "up",
"description": "ABCDEFGH", "duplex": "auto", "mode": "layer3", "speed": "auto"}, "final":
{"admin_state": "up", "description": "ABCDEFGH", "duplex": "auto", "mode": "layer3",
"speed": "auto"}, "proposed": {"admin_state": "up", "description": "ABCDEFGH", "mode":
"layer3"}, "state": "present"}


PLAY RECAP
********************************************************************

nx-osv-1                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-2                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-3                   : ok=1    changed=0    unreachable=0    failed=0
nx-osv-4                   : ok=1    changed=0    unreachable=0    failed=0
```

This **concludes the** whirl-wind tour of NX-API and Ansible. For extra credit, please try **other configuration tasks**. Some tasks / modules to get started :

1. Interface configurations
2. VLAN configurations
3. VRF, Layer 3 configurations

In addition, you could also try any configuration you want using the nxos_command module !

What Next

To know more about Nexus programmability, please visit us at our WoS booth. Some breakout sessions have already been listed.

If there is a use case you would like to discuss with the lab author, please do get in touch at abmodi@cisco.com / @abhinav_m (Email works better!)

Credits: The examples for the Ansible session use nxos-ansible library authored by Jason Edelman. The examples also derive from the samples packaged as part of the library.

Appendix

A) References

1. NX-API Guide on Cisco.com : http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus7000/sw/programmability/guide/b_Cisco_Nexus_7000_Series_NX-OS_Programmability_Guide/b_Cisco_Nexus_7000_Series_NX-OS_Programmability_Guide_chapter_0101.html

2. NX-API Devnet Site : https://developer.cisco.com/site/nx-api/

3. Nxos-ansible : https://github.com/jedelman8/nxos-ansible

B) Solutions to Lab Tasks

All scripts are available in the following directory on the Lab POD :

Lab 1 : Not Applicable. Please type in the commands mentioned. NX-API may already be enabled when you access the pod.

Lab 2 : /home/abhinav/hello.py

Lab 3:

a) **Ping Check** : /home/abhinav/gitprojs/ansiblej/nxos-ansible /nexus_ping.yml

b) **Lab Task 1: Gather Operational Data (CDP and Route Table) :** : /home/abhinav/gitprojs/ansiblej/nxos-ansible/gather_data.yml

Lab 4: Idempotence : /home/abhinav/gitprojs/ansiblej/nxos-ansible/idempotence.yml

Note : A copy of all these files are available at
https://github.com/abhinavmodi/scriptrepo/tree/master/2016cleur-labnms1023