Ansible for Cisco Nexus Switches v1

Last Updated: 26-MAY-2016

About This Solution

Cisco Nexus switches support the latest in automation and DevOps tools. Ansible by Red Hat is a configuration management tool commonly used for server and application automation. Ansible is one of several tools used to automate the provisioning and operations of Cisco Nexus switches. Ansible uses an agentless, push-based model, which means that no software must be installed on the managed device. Instead, the centralized server pushes configuration down to the device.

Use this demonstration to configure Cisco Nexus switches by using Ansible (using the Pycsco module) and learn how to use Ansible for configuration templating.

About This Demonstration

This demonstration includes the following scripted scenarios:

- Scenario 1: Start Simulation
- Scenario 2: Set Up Switches for Management with Ansible
- Scenario 3: Gather Data from Switches Using Ansible
- Scenario 4: Review More Complex Playbook and Variables
- Scenario 5: Configure Layer 2 and Layer 3 with Ansible NX-OS
- Scenario 6: Demonstrate Idempotency
- Scenario 7: Explore Ansible Configuration Templating

For more information:

- View the content profile: https://dcloud-cms.cisco.com/?p=23585
- Visit the Cisco dCloud Help pages: https://dcloud-cms.cisco.com/help
- View all available Cisco dCloud content: https://dcloud.cisco.com

Requirements

Table 1. Requirements

Required	Optional
• Laptop	• Cisco AnyConnect

Topology

This content includes preconfigured users and components to illustrate the scripted scenarios and features of the solution. Most components are fully configurable with predefined administrative user accounts. You can see the IP address and user account credentials to use to access a component by clicking the component icon in the **Topology** menu of your active session and in the scenario steps that require their use.

Please note that in this topology, we have labeled the top switches as “spine” and the bottom switches as “leaf.” While this is not a classic spine-and-leaf topology, we have kept this terminology for the demonstration to more closely mirror a modern data center deployment.

Figure 1. dCloud Topology

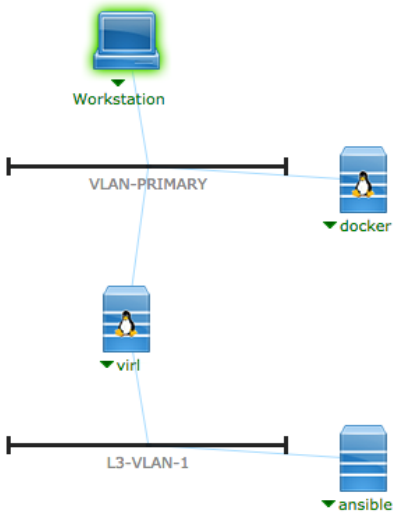
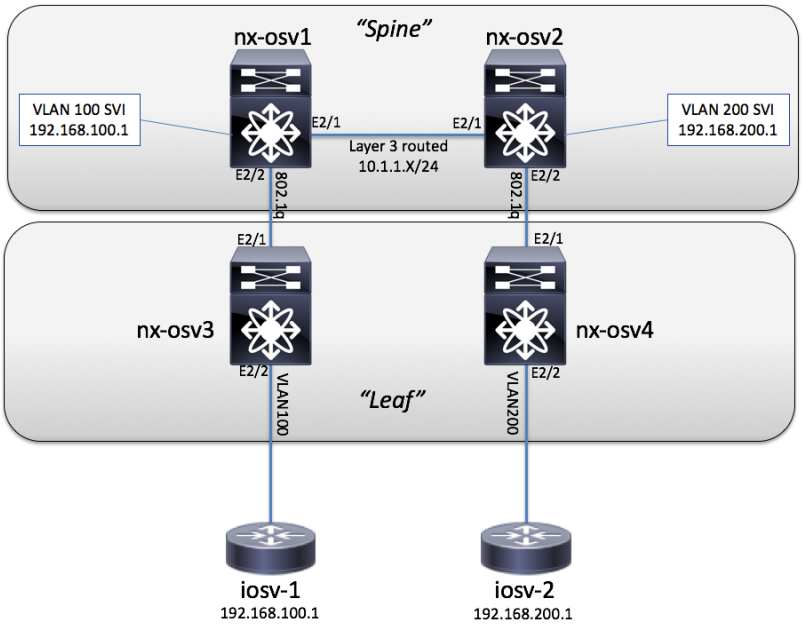


Figure 2. Physical Topology



Get Started

BEFORE PRESENTING

We strongly recommend that you go through this document and work with an active session before presenting in front of a live audience. This will allow you to become familiar with the structure of the document and content.

PREPARATION IS KEY TO A SUCCESSFUL PRESENTATION.

Follow the steps to schedule a session of the content and configure your presentation environment.

1. Browse to dcloud.cisco.com, select the location closest to you, and log in with your Cisco.com credentials.
2. If this is the first time you will use the router with dCloud, register and configure your router. [Show Me How]
3. Schedule a session. [Show Me How]
4. Test your connection. [Show Me How]
5. Verify that the status of your session is **Active** in **My Dashboard > My Sessions**.

NOTE: It may take up to 10 minutes for your session to become active.

6. Click **View** to open the active session.
7. For best performance, connect to the workstation with **Cisco AnyConnect VPN** [Show Me How] and the **local RDP client on your laptop** [Show Me How]
    - Workstation 1: **198.18.133.252**, Username: **administrator**, Password: **C1sco12345**

NOTE: You can also connect to the workstation using the Cisco dCloud Remote Desktop client [Show Me How]. The dCloud Remote Desktop client works best for accessing an active session with minimal interaction. However, many users experience connection and performance issues with this method.

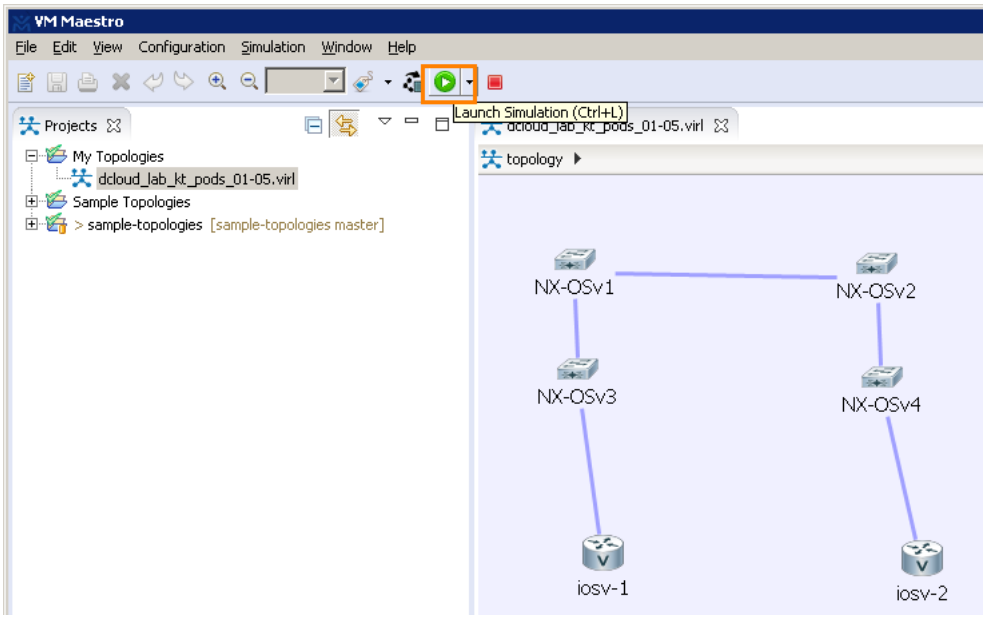
Scenario 1. Start Simulation

This scenario demonstrates how to start the simulation.

Steps

1. From the workstation desktop, double-click the **VM Maestro** icon.
2. From the **dcloud_lab_kt_pods_01-05.virl** tab, notice the demonstration topology.
3. Click **Launch Simulation** and click **OK** in the **Simulation Launched** dialog box.

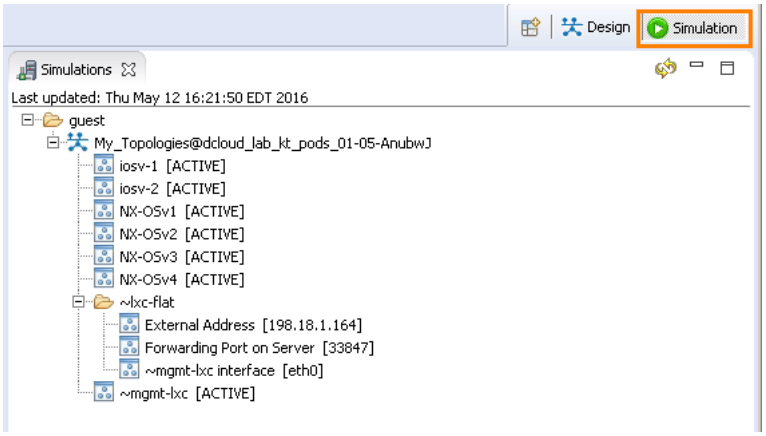
Figure 3. Launch Simulation



4. Click **Simulation** and notice the switches activating in the **Simulations** pane.

NOTE: It may take a few minutes for the switches to boot.

Figure 4. Simulation



5. From the workstation desktop, double-click **Ansible** to open a PuTTY session to the Ansible server.

Figure 5. Ansible

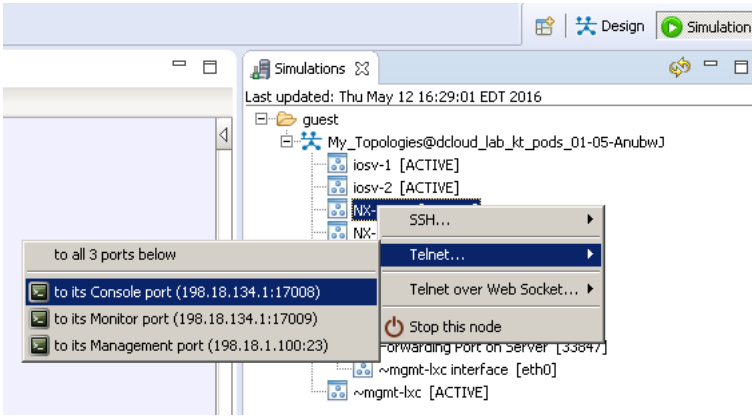
Scenario 2. Set Up Switches for Management with Ansible

This scenario demonstrates how to set up switches for management with Ansible. This demonstration uses the NX-OS Ansible module to manage the switches. Ansible modules are libraries of code that extend the features of Ansible. The NX-OS Ansible module communicates with the switches using NX-API. This is different from most Ansible modules, which copy a Python script to the /tmp directory of the device they manage. NX-API provides a safer and more secure way to communicate with Cisco Nexus devices.

Steps

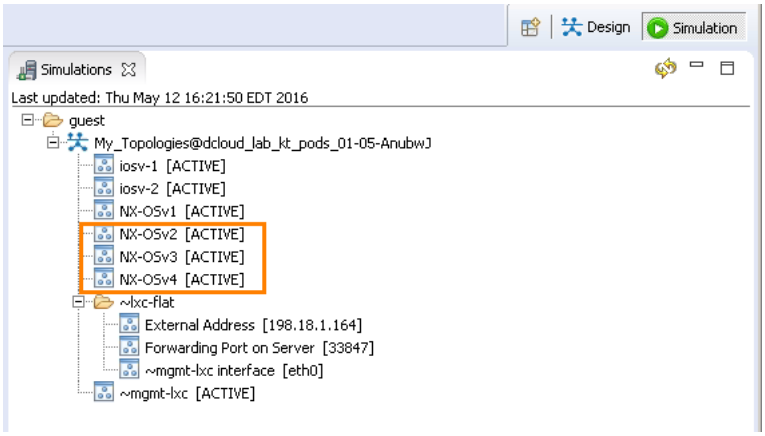
1. From the **Simulations** pane, right-click **NX-OSv1** and choose **Telnet > to its Console port** to open a console connection to the switch.

Figure 6. Log In to Console Port



2. Log in to the switch with username: **guest**, password: **cisco123**.
3. Repeat steps 1 and 2 to log in to the remaining three NX-OSv switches.

Figure 7. Log In to All NX-OSv Switches



NOTE: After starting the simulation, it may take a few minutes for the switches to boot fully. You may see boot-up messages on the console during this time.

4. Enter the following commands to enable NXAPI on **NX-OSv1**, **NX-OSv-2**, **NX-OSv3**, and **NX-OSv4**:

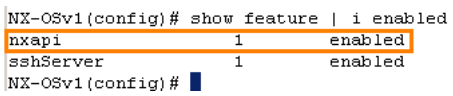
NOTE: This step is a requirement for using the Ansible NXOS module. Execute the following CLI commands (only nx-osv1 is shown):

```
NX-OSv1# configure
NX-OSv1(config)# feature nxapi
```

5. Enter the following command on each of the four switches to verify that you enabled the feature:

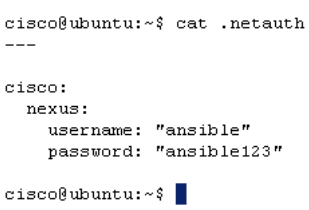
```
NX-OSv1(config)# show feature | i enabled
```

Figure 8. Feature Enabled



6. Return to the **Ansible** host to examine the initial setup on the Ansible machine.
7. If you are not there already, enter **cd ~** to go to the home directory.
8. Enter **cat .netauth** to examine the contents of the authorization file.

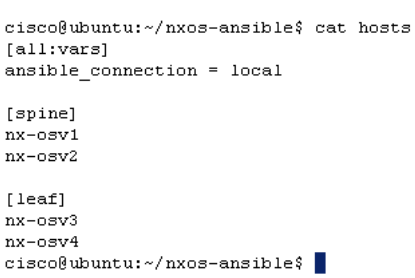
Figure 9. Examine Authorization File



NOTE: Ansible uses a special account to connect to the switches. This account uses RBAC to limit what Ansible can configure.

9. Enter **cd nxos-ansible** to change to the /nxos-ansible directory.
10. Enter **cat hosts** to examine the contents of the Ansible hosts file. Notice that the switches exist in two categories: spine and leaf.

Figure 10. Spine and Leaf



NOTE: You can ping these devices by name to get a response.

NOTE: The Ansible hosts file is used by Ansible only, and it is separate from the Linux hosts file, which is stored in the /etc directory.

11. Enter **ansible -version** to verify that Ansible is installed and working.

Figure 11. Verify Ansible

```
cisco@ubuntu:~/nxos-ansible$ ansible --version
ansible 2.0.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = /usr/share/ansible
cisco@ubuntu:~/nxos-ansible$
```

Scenario 3. Gather Data from Switches Using Ansible

This scenario demonstrates how to use Ansible to collect the CDP neighbor tables and management VRF routing tables and store them on the server in a JSON file. JSON formatting represents structured data. The NX-OS Ansible modules support for a number of commands, such as CDP, but some commands must be sent directly through CLI. This scenario examines both use cases, as well as the file format used by Ansible, called YAML, which stands for YAML Ain't a Markup Language.

Steps

1. From the Ansible server, enter **cd ~/nxos-ansible**, if you are not already in this directory.
2. Enter **ls data** to verify that no files exist in the data directory.
3. Ansible configuration files exist in YAML (.yml) format and are called playbooks. Enter **cat gather_data.yml** to examine the playbook for the current scenario.

Figure 12. Examine Playbook

```
cisco@ubuntu:~/nxos-ansible$ cat 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

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

  - name: store to file
    template: src=templates/data.j2 dest=data/{{ inventory_hostname }}_gather_data.json
cisco@ubuntu:~/nxos-ansible$
```

NOTE: The file begins with three hyphens (---). All YAML files must begin this way. Notice the overall structure of the file: It contains lists of tasks with each task preceded by a hyphen. The first task provides overall information for the playbook. The author of the playbook assigns the playbook name, which is assigned when the playbook runs. The playbook runs on all hosts.

4. Examine the playbook tasks:
   - **get neighbors** uses the NXOS Ansible module **nxos_get_neighbors** to query the CDP table. Notice that the command uses two parameters (nxos_get_neighbors and register). When the playbook runs, {{ inventory_hostname }}, a special reserved variable, is replaced with the host name of the switch.
   - **get routing table** uses an NXOS Ansible module to send a command.
   - **store to file** takes the output of the previous commands and puts them into a file based on a template that has already been defined. The destination is a file with the switch name followed by **_gather_data.json** in the data directory.

5. Enter **cat templates/data.j2** to examine the contents of the template. This file is written using the Jinja2 templating language, which is integrated with Ansible.

Figure 13. Examine Template

```
cisco@ubuntu:~/nxos-ansible$ cat templates/data.j2
CDP Info:

{{my_neighbors | to_nice_json}}

Route Info:

{{my_routes | to_nice_json}}
cisco@ubuntu:~/nxos-ansible$
```

NOTE: This template takes the my_routes and my_neighbors output from the playbook and pipes them to a JSON formatting utility.

6. Enter **ansible-playbook -i hosts gather_data.yml** to run the playbook.

7. While the playbook runs, notice the output from Ansible:

   - Each task in the playbook runs and Ansible reports the results for each device.
   - At the end of the run, a recap shows what tasks ran.

Figure 14. Ansible Output

```
cisco@ubuntu:~/nxos-ansible$ ansible-playbook -i hosts gather_data.yml

PLAY [Gather Data] *************************************************************

TASK [get neighbors] ***********************************************************
ok: [nx-osv4]
ok: [nx-osv2]
ok: [nx-osv3]
ok: [nx-osv1]

TASK [get routing table for mgmt VRF] ******************************************
ok: [nx-osv2]
ok: [nx-osv3]
ok: [nx-osv4]
ok: [nx-osv1]

TASK [store to file] ***********************************************************
changed: [nx-osv3]
changed: [nx-osv4]
changed: [nx-osv2]
changed: [nx-osv1]

PLAY RECAP *********************************************************************
nx-osv1                    : ok=3    changed=1    unreachable=0    failed=0
nx-osv2                    : ok=3    changed=1    unreachable=0    failed=0
nx-osv3                    : ok=3    changed=1    unreachable=0    failed=0
nx-osv4                    : ok=3    changed=1    unreachable=0    failed=0

cisco@ubuntu:~/nxos-ansible$
```

8. Enter **ls data** to examine the data directory. There should now be a file for each device.

Figure 15. Files for Each Device

```
cisco@ubuntu:~/nxos-ansible$ ls data
nx-osv1_gather_data.json  nx-osv2_gather_data.json  nx-osv3_gather_data.json  nx-osv4_gather_data.json
cisco@ubuntu:~/nxos-ansible$
```

9. Enter **less data/nx-osvX_gather_data.json** to examine one of the files. In the command, **X** is a switch number for which you must enter **1**, **2**, **3**, or **4**.

10. Notice the switch CDP and IP routing information in JSON format.

Figure 16. CDP and IP Routing Information

```
CDP Info:

{
    "ansible_facts": {
        "neighbors": [
            {
                "local_interface": "Ethernet2/1",
                "neighbor": "NX-OSv2(TB3EA68A27B)",
                "neighbor_interface": "Ethernet2/1"
            },
            {
                "local_interface": "Ethernet2/2",
                "neighbor": "NX-OSv3(TB3EC7C9E4B)",
                "neighbor_interface": "Ethernet2/1"
            }
        ]
    },
    "changed": false
}

Route Info:

{
    "changed": false,
    "commands": "show ip route vrf management",
    "proposed": {
        "cmd_type": "show",
        "commands": "show ip route vrf management",
        "text": null
    },
    "response": [
        {
            "body": {
                "TABLE_vrf": {
                    "ROW_vrf": {
                        "TABLE_addrf": {
                            "ROW_addrf": {
                                "TABLE_prefix": {
                                    "ROW_prefix": [
                                        {
                                            "TABLE_path": {
                                                "ROW_path": {
                                                    "clientname": "direct",
                                                    "hidden": "false",
                                                    "ifname": "mgmt0",
                                                    "ipnexthop": "198.18.1.100",
                                                    "metric": "0",
                                                    "pref": "0",
                                                    "stale": "false",
                                                    "stale-label": "false",
                                                    "ubest": "true",
                                                    "unres": "false",
data/nx-osv1_gather_data.json
```

NOTE: Do not be concerned if you do not fully understand the JSON format. The primary concept to understand is that Ansible collected this data and saved it to a file. Other playbooks or even Python scripts could also use this data.

11. Press **Ctrl-C** to stop viewing the file.

Scenario 4. Review More Complex Playbook and Variables

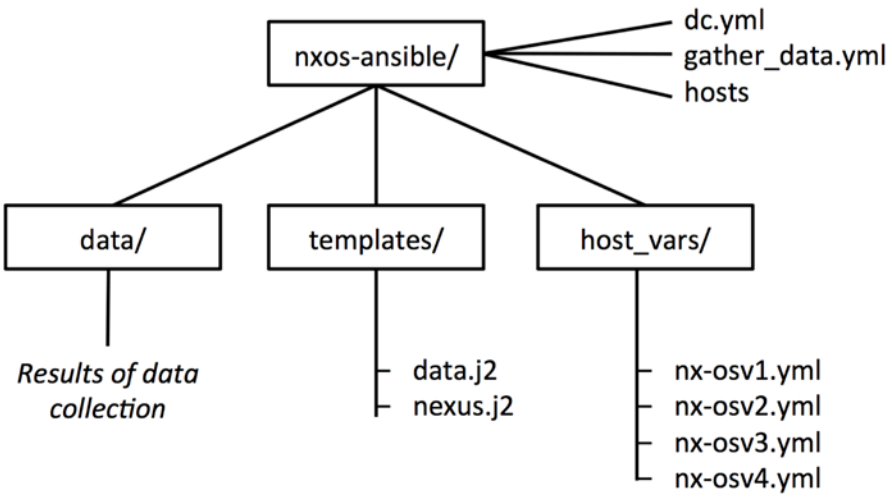
Scenario 1 examined the host file. The host file contains two roles (spine and leaf) with hosts assigned to one of these two roles. Roles are assigned by placing devices under a specific role in the host file.

The simple playbook from Scenario 2 ran on all hosts and only used one variable, the reserved {{ inventory_hostname }} variable. This scenario and Scenario 4 use a more complex playbook that classifies devices based on role. The playbook in this scenario uses host variables.

In an Ansible playbook, some configurations apply to all hosts assigned a given role. For example:

- All spine switches might be configured with VPC.
- Some configurations are hard coded into the playbook.
- Variables provide other configurations that may be specific to a host or a group.
- Perhaps you want some VLANs configured on all spine switches, but not on leaf switches; these VLANs would be defined as a group variable.
- Host-specific parameters, such as interface IP addresses, are defined as host variables.

Figure 17. Demonstration Directory Structure



Steps

1. From the Ansible server, enter **cd nxos-ansible** and then enter **cd host_vars**.
2. Enter **ls** to examine the contents of the directory. There is a YAML file for each switch.

Figure 18. YAML Files

```
cisco@ubuntu:~/nxos-ansible/host_vars$ ls
nx-osv1.yml  nx-osv2.yml  nx-osv3.yml  nx-osv4.yml
cisco@ubuntu:~/nxos-ansible/host_vars$
```

3. Enter **cat nx-osv1.yml** to examine the contents of the first file.

Figure 19. File Contents

```
---

vlan: 100
svi_ip: 192.168.100.1
l3_ip:  10.1.1.1
static_rt:  192.168.200.0/24
nh: 10.1.1.2
hostname: nx-osv1
mgmt_ip:  198.18.1.100
if_list: ['Eth1/1','Eth1/2','Eth1/3']

cisco@ubuntu:~/nxos-ansible/host_vars$
```

NOTE: The file begins with three hyphens (---) and contains several defined variables. The name of the variable appears, followed by a colon and the value of the variable.

4. Enter **cd ..** to return to the /nxos-ansible directory.
5. Enter **cat dc.yml** to examine the contents of the dc.yml playbook.
6. The playbook contains four distinct sections or plays:
    - The first play configures **feature interface-vlan** on the spine switches. This play pushes raw CLI to a device using the nxos-command module. Notice again that the reserved variable **{{ inventory_hostname }}** specifies hosts, but it only pushes to hosts in the spine group.

NOTE: Use the **nxos-command** module if a given configuration item is not available in the NXOS Ansible modules, or use it to set specific parameters not available through the modules.

- The next three plays configure the VLANs and interfaces. Notice that different modules, such as **nxos_interface** and **nxos_switchport**, perform these tasks. In some cases, a parameter is hard coded (**Ethernet 2/1**), but elsewhere a variable is used (**{{ vlan }}**). For each host, Ansible replaces that variable with the value in the appropriate **host_vars** file.

NOTE: Each Ansible playbook contains several plays like those shown in the example.

Figure 20. DC.YML Playbook

```
cisco@ubuntu:~/nxos-ansible$ cat dc.yml
---

- name:  Configuring features required
  hosts:  spine
  tasks:
    - name: Configuring feature interface-vlan
      nxos_command:
        host: "{{ inventory_hostname }}"
        type: config
        command: "feature interface-vlan"

- name:  Configure VLANs
  hosts: all
  tasks:
  - nxos_vlan: vlan_id=100 host={{ inventory_hostname }}
  - nxos_vlan: vlan_id=200 host={{ inventory_hostname }}

- name:  Configure interfaces on spine
  hosts: spine
  tasks:
  - nxos_interface: interface=Ethernet2/1 mode=layer3 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=Ethernet2/2 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=vlan{{ vlan }} mode=layer3 admin_state=up host={{ inventory_hostname }}
  - nxos_ipv4_interface: interface=Ethernet2/1 ip_addr={{ l3_ip }} mask=24 host={{ inventory_hostname }}
  - nxos_switchport: interface=eth2/2 mode=trunk host={{ inventory_hostname }}
  - nxos_ipv4_interface: interface=vlan{{ vlan }} ip_addr={{ svi_ip }} mask=24 host={{ inventory_hostname }}

- name:  Configure interfaces on leaves
  hosts:  leaf
  tasks:
  - nxos_interface: interface=Ethernet2/1 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=Ethernet2/2 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_switchport: interface=Ethernet2/1 mode=trunk host={{ inventory_hostname }}
  - nxos_switchport: interface=Ethernet2/2 mode=access access_vlan={{ access_vlan }} host={{ inventory_hostname }}
```

NOTE: Ansible allows a more hierarchical and structured approach to building playbooks and variables. For more information, refer to Appendix A.

Scenario 5. Configure Layer 2 and Layer 3 with NX-OS Ansible

This scenario configures leaf switches with basic layer 2 (L2) functionality and spine switches with layer 3 (L3) and L2 functionality. The spine and leaf functions are somewhat limited in this scenario because virtual NX-OS does not support full data-plane capabilities. However, the functionality is sufficient to demonstrate Ansible's power.

This scenario demonstrates how to:

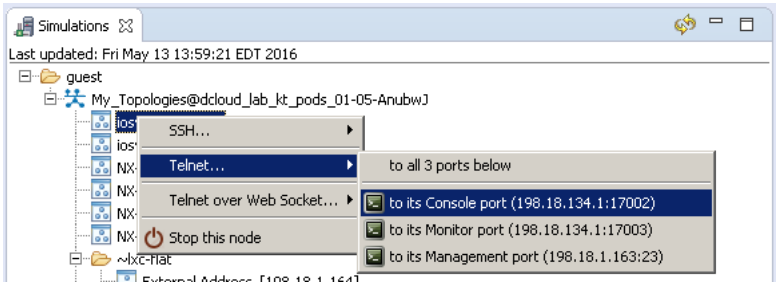
- Enable an access port on the leaf switches facing the IOS-V ping initiator/target.
- Enable 802.1q trunk ports between the spine and leaf switches.
- Enable an L3 port between the spine switches.
- Create SVI (VLAN interfaces) for the two VLANs in use (100 and 200).
- Add static routes to complete the picture.

After completing the configuration, pings between the IOS-V devices will work.

Steps

1. From the workstation desktop, double-click the **VM Maestro** icon to start VM Maestro.
2. From the **Simulations** pane, right-click **iosv-1** and choose **Telnet > to its Console port** to open a console connection to iosv-1.

Figure 21. Log In to Console Port



3. Run the following command to confirm that you cannot ping the other iosv devices:

```
iosv-1>ping 192.168.200.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.200.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

4. Run the following commands on NX-OSv1 and NX-OSv2 to verify that no additional VLANs or interfaces are configured:

```
show run
show vlan brief
show ip int brief
```

5. Using vi, open the dc.yml playbook.

```
vi dc.yml
```

NOTE: A preconfigured playbook (dc-complete.yml) already exists in the /nxos-ansible directory. As alternative to editing the dc.yml file, proceed to step 10 and enter **ansible-playbook -i hosts dc-complete.yml** instead.

6. Press **G** to skip to the end of the playbook.

NOTE: You must type an upper-case G.

7. Press lower-case **o** to begin a new line.

8. Enter the following commands to provision the static routes defined in the host variables:

NOTE: Be sure to type spaces for indentation. Do not press **Tab** and do not use copy and paste shortcuts. Enter the indentation and commands exactly as shown; YAML is sensitive.

```
- name:  Install static routes
  hosts:  spine
  tasks:
  -nxos_static_routes: prefix={{ static_rt }} next_hop={{ nh }} host={{ inventory_hostname }}
```

NOTE: These commands define a new play that runs on the spine switches. It installs a static route to a prefix in the host_vars file called **static_rt** with a next hop of **nh**.

9. Press **ESC** and enter **:wq**.

10. Enter **ansible-playbook –i hosts dc.yml** to run the playbook.

NOTE: If you did not complete steps 5 through 9, enter **ansible-playbook -i hosts dc-complete.yml** as the alternative command.

NOTE: As the playbook runs, Ansible reports on each step of the playbook.

11. Return to the iosv-1 console and run the following command again and confirm that the ping succeeds.

```
iosv-1>ping 192.168.200.1
```

12. Optionally, run the following commands on NX-OSv1 and NX-OSv2 again to see the changes:

```
show run
show vlan brief
show ip int brief
```

Scenario 6. Demonstrate Idempotency

Simply put, idempotency means that a change of state is performed only if required. In other words, if the entity is already in the desired state, then the configuration is not applied. Therefore, Ansible modules are written in a way to indicate the desired state.

For example:

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

In the example, the task ensures that Ethernet 2/4 has the description mentioned and is in L3 mode. Therefore, the module nxos_interface, being idempotent, first checks the description and mode using show commands, such as **show interface**.

If the description and mode match, no action is required and the task will return **ok** without making any changes (the summary will have **changed=0**). Alternatively, if the description and mode are different, the module makes the change. Idempotency ensures that only required changes are performed, and unnecessary commands that result in no operations are omitted.

The easiest way to see idempotency is to run a playbook twice.

Steps

1. Enter the following command to run the dc.yml playbook:

```
ansible-playbook -i hosts dc.yml
```

2. Notice that instead of seeing **changed**, most of the tasks now show **ok**, because Ansible determined that the configuration is complete as is and required no additional updates.

NOTE: The feature configuration still shows **changed** because the configuration is pushed as CLI using the nxos_command module. Therefore, it is not idempotent.

Scenario 7. Explore Ansible Configuration Templating

Scenario 1 demonstrated templating in action by using a template to save data to a file. This scenario demonstrates how to use Ansible to generate configuration files for devices based on a Jinja2 template. Previous configurations in this guide used either a raw CLI push using nxos-command or declarative NX-OS Ansible modules, such as nxos_vlan.

Ansible also generates configuration files based on a template, which can then be copied to the switch by using Ansible or another method. This scenario uses a template to generate configurations, but pushing the configurations is outside of the scope of the scenario.

Steps

1. From the Ansible host, enter **cd ~/nxos-ansible**, if you are not already in this directory.
2. Enter **ls configs** to verify that the /configs directory is empty:
3. Enter **cat templates/nexus.j2** to examine the template file

Figure 22. Template File

```
cisco@ubuntu:~/nxos-ansible$ cat dc.yml
---

- name:  Configuring features required
  hosts:  spine
  tasks:
    - name: Configuring feature interface-vlan
      nxos_command:
        host: "{{ inventory_hostname }}"
        type: config
        command: "feature interface-vlan"

- name:  Configure VLANs
  hosts: all
  tasks:
  - nxos_vlan: vlan_id=100 host={{ inventory_hostname }}
  - nxos_vlan: vlan_id=200 host={{ inventory_hostname }}

- name:  Configure interfaces on spine
  hosts: spine
  tasks:
  - nxos_interface: interface=Ethernet2/1 mode=layer3 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=Ethernet2/2 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=vlan{{ vlan }} mode=layer3 admin_state=up host={{ inventory_hostname }}
  - nxos_ipv4_interface: interface=Ethernet2/1 ip_addr={{ l3_ip }} mask=24 host={{ inventory_hostname }}
  - nxos_switchport: interface=eth2/2 mode=trunk host={{ inventory_hostname }}
  - nxos_ipv4_interface: interface=vlan{{ vlan }} ip_addr={{ svi_ip }} mask=24 host={{ inventory_hostname }}

- name:  Configure interfaces on leaves
  hosts:  leaf
  tasks:
  - nxos_interface: interface=Ethernet2/1 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_interface: interface=Ethernet2/2 mode=layer2 admin_state=up host={{ inventory_hostname }}
  - nxos_switchport: interface=Ethernet2/1 mode=trunk host={{ inventory_hostname }}
  - nxos_switchport: interface=Ethernet2/2 mode=access access_vlan={{ access_vlan }} host={{ inventory_hostname }}

cisco@ubuntu:~/nxos-ansible$ ls configs
cisco@ubuntu:~/nxos-ansible$ cat templates/nexus.j2
hostname {{ hostname }}

interface mgmt0
  ip address {{ mgmt_ip }}/24
  vrf context management

{% for intf in if_list %}
interface {{ intf }}
  no shutdown
  switchport
  switchport mode trunk

{% endfor %}
```

NOTE: The template is not declarative, but simply a snippet of actual NX-OS configuration. Various values in the configuration are replaced by variables.

4. Look for the section that is bracketed by the following statements:

```
{% for intf in if_list %}
{% endfor %}
```

NOTE: Jinja2 iterates through a list of values when building a configuration. The configuration block between the **for** and **endfor** statements is replicated as many times as there are values in the list. The template file runs a no shut command in a series of interfaces and puts them into trunk mode.

5. Enter **cat host_vars/nx-osv1.yml** to examine the host-specific variables files again.

Figure 23. Host-Specific Variables (Some Variables Removed)

```
vlan: 100
svi_ip: 192.168.100.1
l3_ip:  10.1.1.1
static_rt:  192.168.200.0/24
nh: 10.1.1.2
hostname: nx-osv1
mgmt_ip:  198.18.1.100
if_list: ['Eth1/1','Eth1/2','Eth1/3']
```

NOTE: Notice the variables that Ansible will insert into the template, especially the interface list, which names three interfaces.

6. Enter **ansible-playbook -i hosts template.yml** to run the playbook.

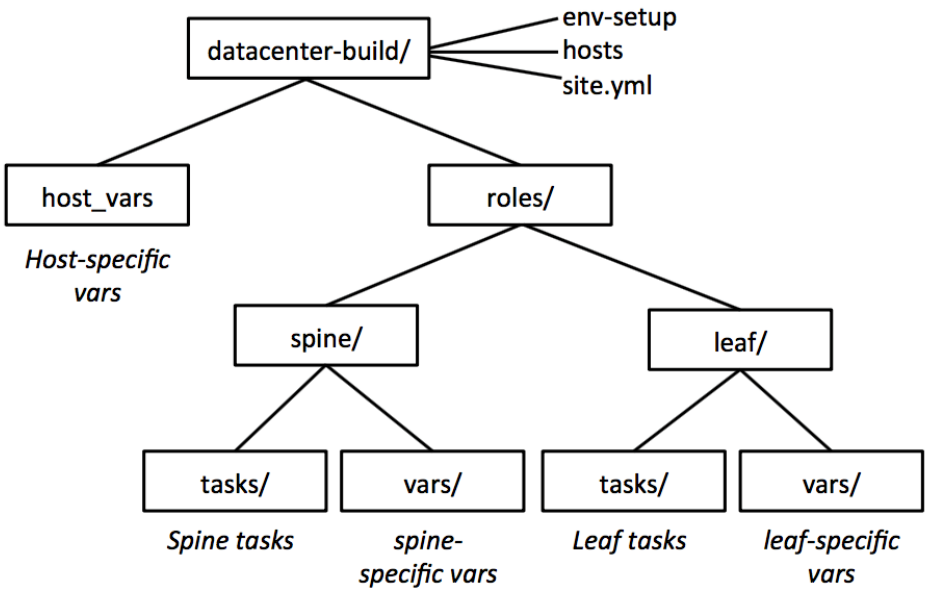
NOTE: When Ansible finishes running the playbook, there will be four .cfg files in the /configs directory.

7. Enter **cat configs/nx-osv1.cfg** to examine any of the files. Notice the host-specific variables included and all of the Ethernet interfaces in the list replicated.

Appendix A. Advanced Directory Structure

The simple Ansible playbook featured for demonstration purposes would not scale well in a large data center. However, Ansible provides a more structured approach for building playbooks. An example of this hierarchical approach is in the /datacenter-build directory. You can examine the contents of the playbooks, but they will not run in the Cisco dCloud environment.

Figure 24. Hierarchical Directory Structure



This directory structure is ideal for building a large data center. Instead of using a single file, the tasks and variables for each role (in this case, spine and leaf) are defined in separate directories. The spine role has two subdirectories, one with the tasks that will be executed, and one with the variables that are specific to that group. Host-specific variables are defined at the root of the directory, along with the hosts file and the site.yml file, which launches the data center build.

Appendix B. Resources

For more information about using Ansible with Nexus switches, refer to the following resources:

- https://github.com/jedelman8/nxos-ansible
  Ansible Modules to Automate Cisco NX-OS switches provides detailed instructions about how to use NX-OS Ansible modules. It also contains an excellent walkthrough.
- http://jedelman.com/home/automating-cisco-nexus-switches-with-ansible/
  Automating Cisco Nexus Switches with Ansible provides a detailed walkthrough of Ansible automation.
- http://packetpushers.net/python-jinja2-tutorial/
  Python and Jinja2 Tutorial includes good videos with examples for using Jinja2 templating.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**