

Infrastructure as Code Made Easy: NX-OS and Nexus Dashboard Automation with Ansible

cisco Live !

Alessandro De Prato
Technical Marketing Engineer

Cisco Webex App

Questions?

Use Cisco Webex App to chat with the speaker after the session

How

- 1 Find this session in the Cisco Live Mobile App
- 2 Click “Join the Discussion”
- 3 Install the Webex App or go directly to the Webex space
- 4 Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until 14 November 2025.



<https://cislive.ciscoevents.com/cislivebot/#BRKDCN-2946>

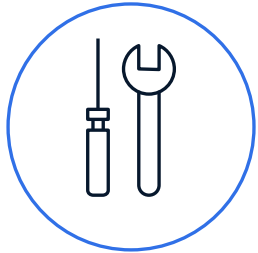
Agenda

- 01 What is Infrastructure as Code?**
- 02 IaC with NX-OS and Ansible**
- 03 IaC with Nexus Dashboard and Ansible**
- 04 Start your IaC Journey!**

What is Infrastructure as Code?

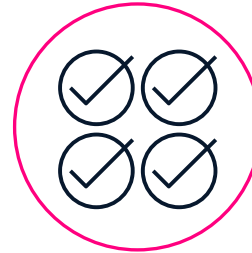
We Start From Automation

Lots of Benefits



Increase Efficiency

Save time and increase efficiency by automating repetitive tasks



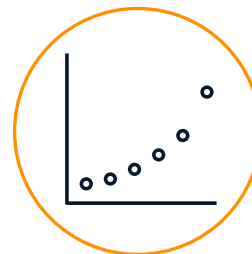
Reduce Risk

Minimize human error and mistakes, increase consistency and predictability



More Resources

Release resources in your organization for innovation and moving your business forward



Accelerate Business

Reduce time to deploy infrastructure, hence reducing time to market and accelerating business outcomes

Automation

Myth Busters



**I can't advantage from automation
because I am not a software
developer**

Cody Packet
CLI Enthusiast

Automation

Myth Busters



False! Whether you are a developer or not, you can still implement automated processes.



Alessandro De Prato
TME, CLI enthusiast & Automation Lover



ANSIBLE
BRKDCN-2946



Jenkins



HashiCorp
Terraform

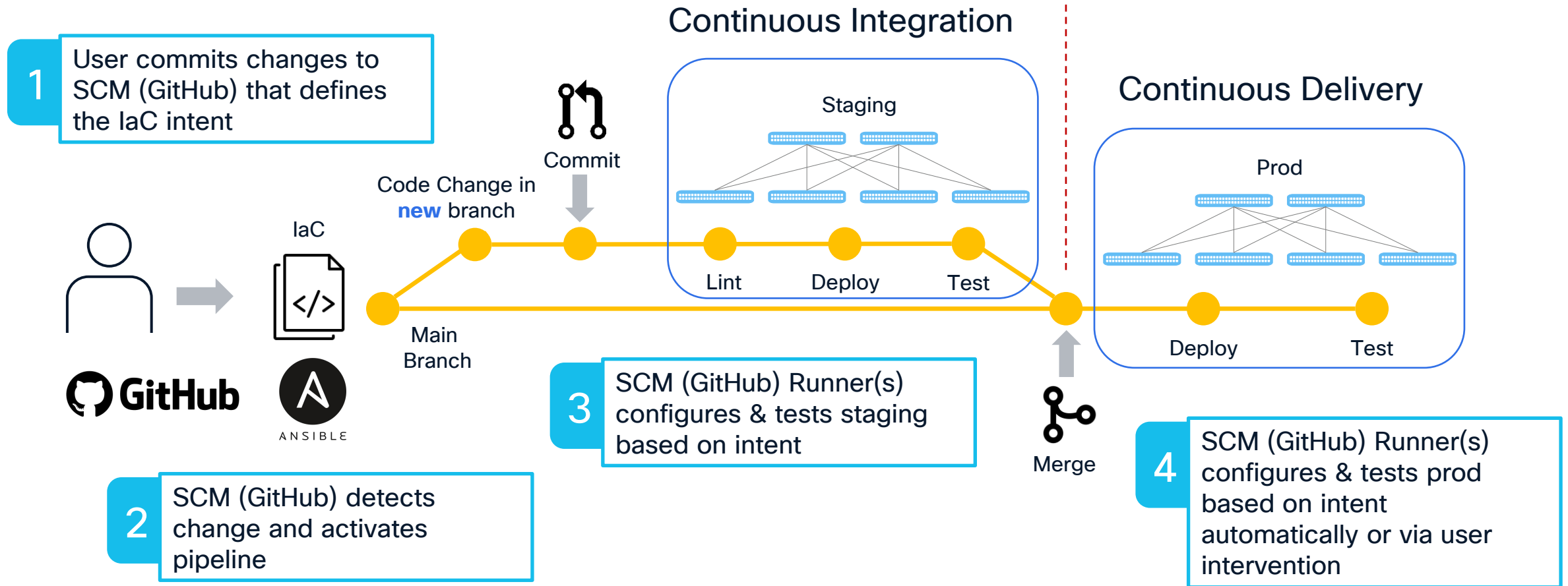
Infrastructure as Code for Network Ops

- Easy to implement, most of the times the biggest barrier is the initial adoption
- Using “code” to provision and/or manage infrastructure
- Infrastructure as Code is not specific to a particular automation engine or specific programming language
- The intended configuration state of network devices are sourced from source code management (git) instead of the devices themselves

Today's Focus

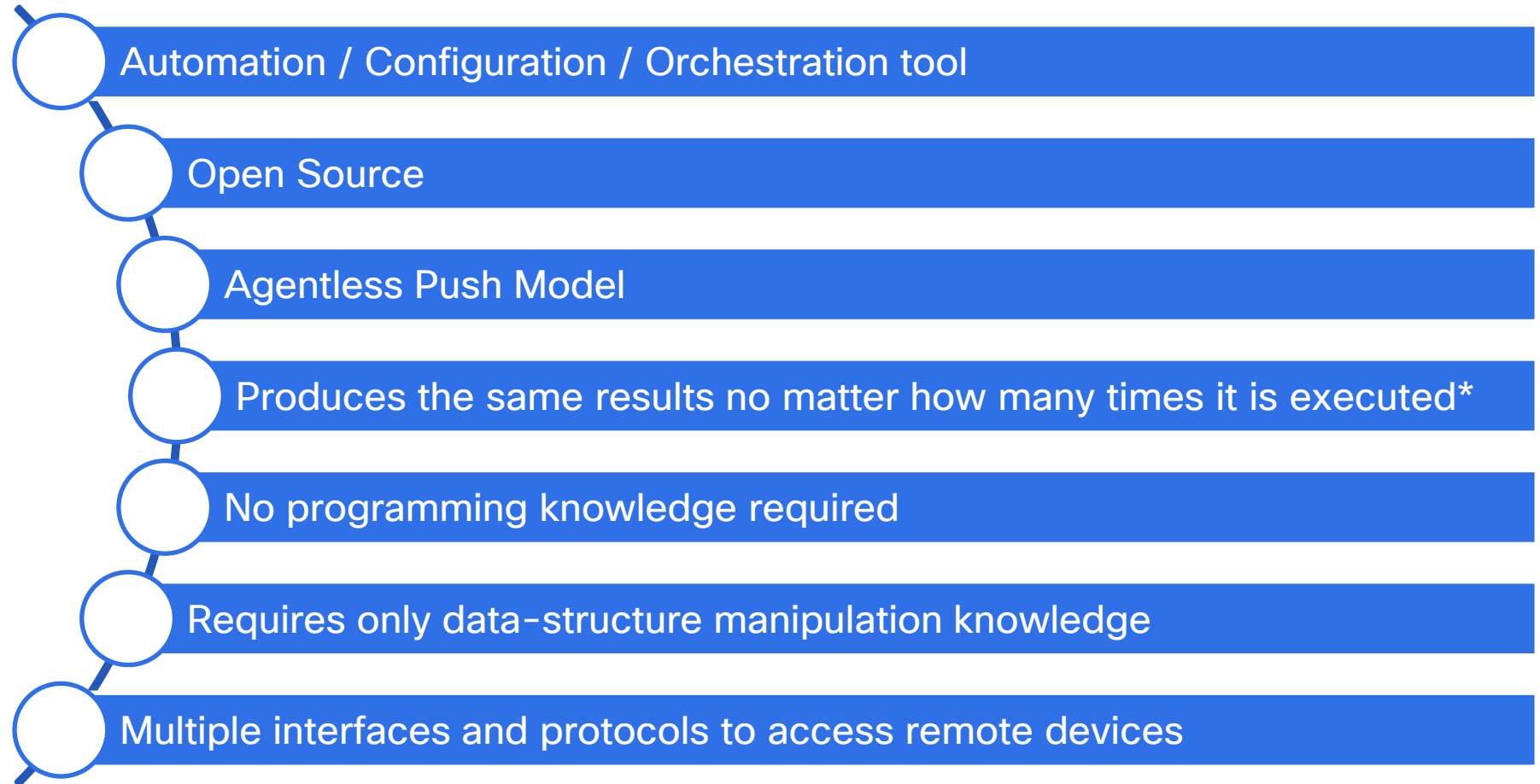


Infrastructure as Code for Network Ops



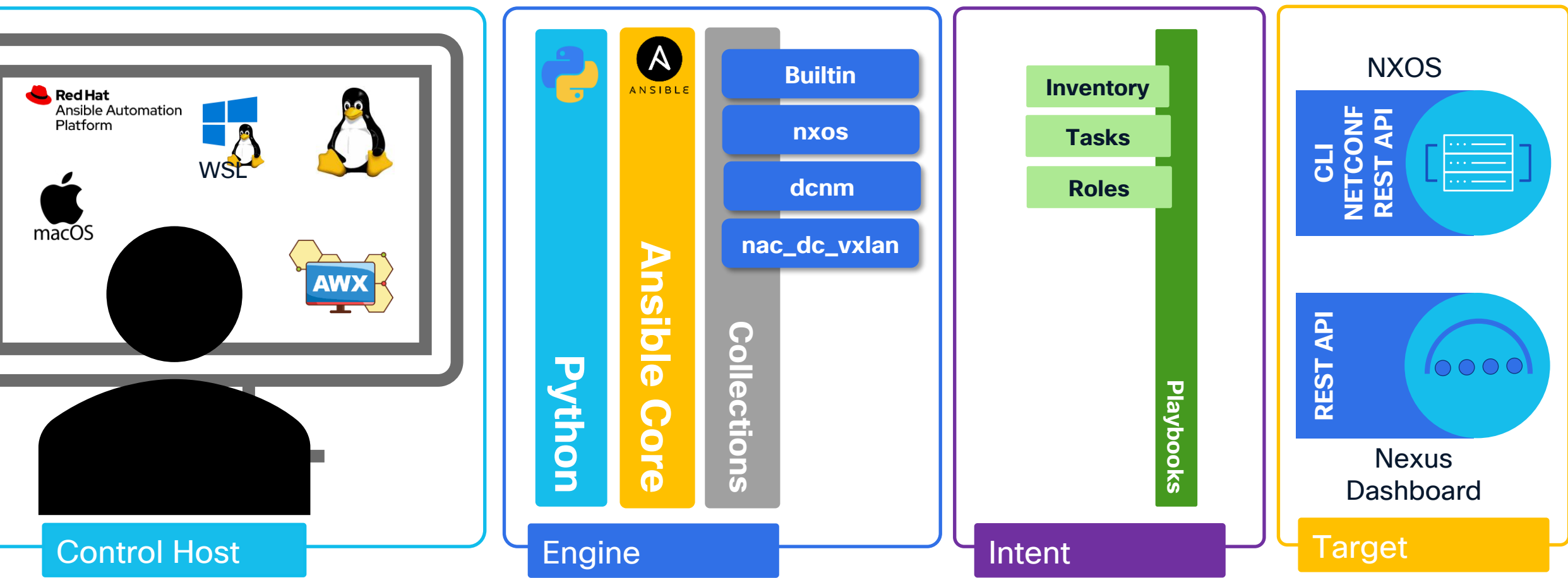
Infrastructure as Code with NX-OS and Ansible

What is Ansible?



*idempotent

What makes up Ansible?



Installing Ansible

```
% pip install ansible-core
```

- Installs only the core components
- Collections must be installed by you
- Smaller footprint and more control
- Assures install of latest collection version released!

```
% pip install ansible
```

- “batteries included”
- Installs community-curated selection of Ansible Collections
- Complete package but larger footprint on filesystem
- Might not install the latest version of a desired collection!

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Ansible Collections

Ansible Collections

- Introduced in Ansible 2.9
- External collections are developed and shipped at their own pace
 - Allows vendor flexibility in relation to product releases
- Uses Ansible Galaxy as the delivery vehicle
- Contains modules, plugins, filters

```
% ansible-galaxy collection install cisco.nxos
```

```
% ansible-galaxy collection install cisco.dcnm
```

```
% ansible-galaxy collection install cisco.nac_dc_vxlan
```

(NXOS | DCNM | NaC) - [https://galaxy.ansible.com/cisco/\(nxos | dcnm | nac_dc_vxlan \)](https://galaxy.ansible.com/cisco/(nxos|dcnm|nac_dc_vxlan))

Ansible Cisco.Nxos

Collection Modules

- Modules perform specific task like configure vlans, interfaces, OSPF, BGP, and more
- Documentation provides usage details, required variables, default variables, etc
- Actively maintained by RedHat with Cisco support



Ansible Community Documentation

ANSIBLE GALAXY

Galaxy User Guide

Galaxy Developer Guide

REFERENCE & APPENDICES

Collection Index

Collections in the Amazon Namespace

Collections in the Ansible Namespace

Collections in the Arista Namespace

Collections in the Awx Namespace

Collections in the Azure Namespace

Collections in the Check_point Namespace

Collections in the Chocolatey Namespace

Collections in the Cisco Namespace

Cisco.Aci

Cisco.Dnac

Cisco.Intersight

Cisco.Ios

Cisco.Iosxr

Cisco.Meraki

Cisco.Mso

Cisco.Nxos

Description

Plugin Index

Cisco.Ucs

Plugin Index

These are the plugins in the cisco.nxos collection:

Modules

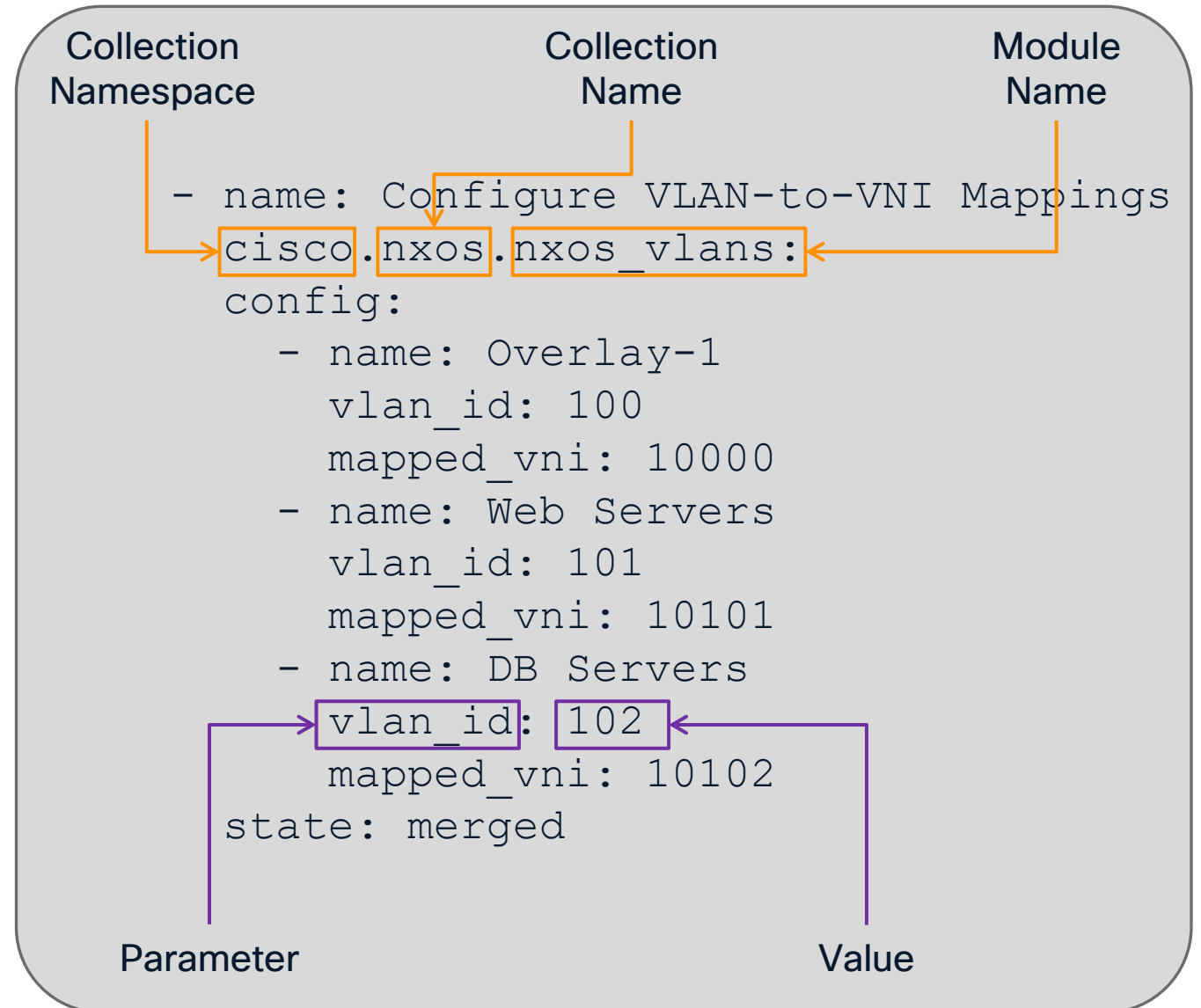
- [nxos_aaa_server module](#) – Manages AAA server global configuration.
- [nxos_aaa_server_host module](#) – Manages AAA server host-specific configuration.
- [nxos_acl_interfaces module](#) – ACL interfaces resource module
- [nxos_acls module](#) – ACLs resource module
- [nxos_banner module](#) – Manage multiline banners on Cisco NXOS devices
- [nxos_bfd_global module](#) – Bidirectional Forwarding Detection (BFD) global-level configuration
- [nxos_bfd_interfaces module](#) – BFD interfaces resource module
- [nxos_bgp_address_family module](#) – BGP Address Family resource module.
- [nxos_bgp_global module](#) – BGP Global resource module.
- [nxos_bgp_neighbor_address_family module](#) – BGP Neighbor Address Family resource module.
- [nxos_bgp_templates module](#) – BGP Templates resource module.
- [nxos_command module](#) – Run arbitrary command on Cisco NXOS devices
- [nxos_config module](#) – Manage Cisco NXOS configuration sections
- [nxos_devicealias module](#) – Configuration of device alias for Cisco NXOS MDS Switches.
- [nxos_evpn_global module](#) – Handles the EVPN control plane for VXLAN.
- [nxos_evpn_vni module](#) – Manages Cisco EVPN VXLAN Network Identifier (VNI).
- [nxos_facts module](#) – Gets facts about NX-OS switches
- [nxos_fc_interfaces module](#) – Fc Interfaces resource module
- [nxos_feature module](#) – Manage features in NX-OS switches.
- [nxos_file_copy module](#) – Copy a file to a remote NXOS device.
- [nxos_gir module](#) – Trigger a graceful removal or insertion (GIR) of the switch.
- [nxos_gir_profile_management module](#) – Create a maintenance-mode or normal-mode profile for GIR.
- [nxos_hostname module](#) – Hostname resource module.
- [nxos_hsrp module](#) – (deprecated, removed after 2028-06-01) Manages HSRP configuration on NX-OS switches.
- [nxos_hsrp_interfaces module](#) – HSRP interfaces resource module

<https://docs.ansible.com/ansible/latest/collections/cisco/nxos/index.html>

Ansible Modules

Calling a module

- Always use the fully qualified collection name (FQCN) for the module
- The modules require parameters with values assigned that define your configuration intent
 - Data is encoded in yaml with a wide variable type support
- Module documentation provides details on default values and required values
- Documentation contains easily consumable examples



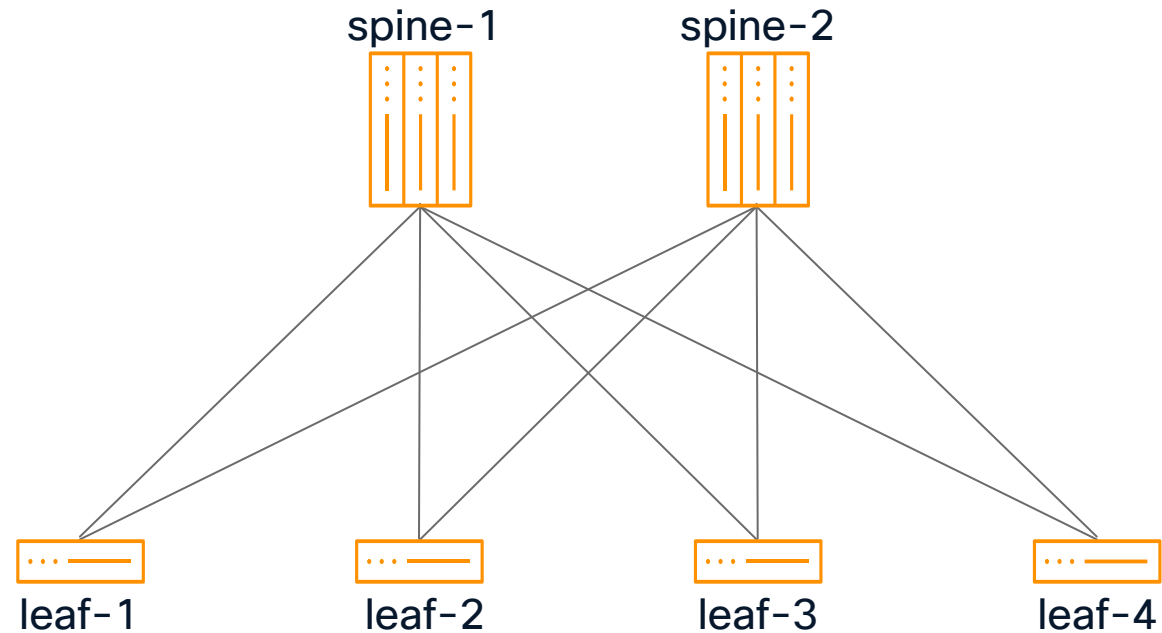
Ansible Concepts

Ansible Concepts

VXLAN EVPN Fabric Anatomy

Common configurations:

- Hostname
- NTP/DNS
- External Authentication
- Software Version

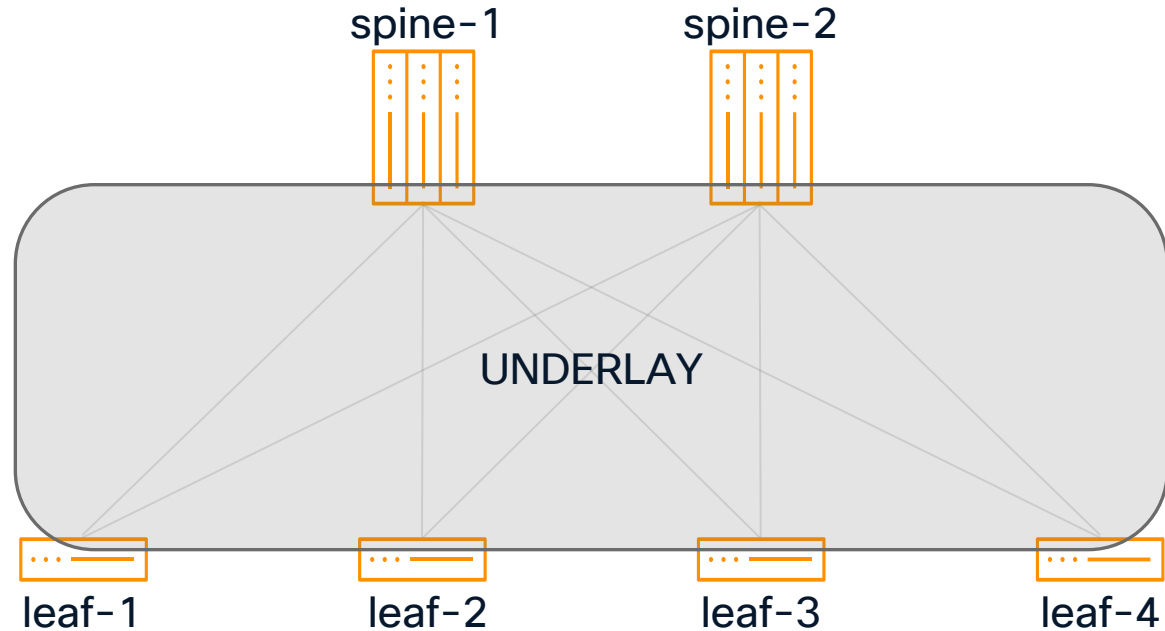


Ansible Concepts

VXLAN EVPN Fabric Anatomy

Underlay configurations:

- Point-to-point links
- Loopbacks
- Dynamic Routing Adjancencies
- Multicast (optional)

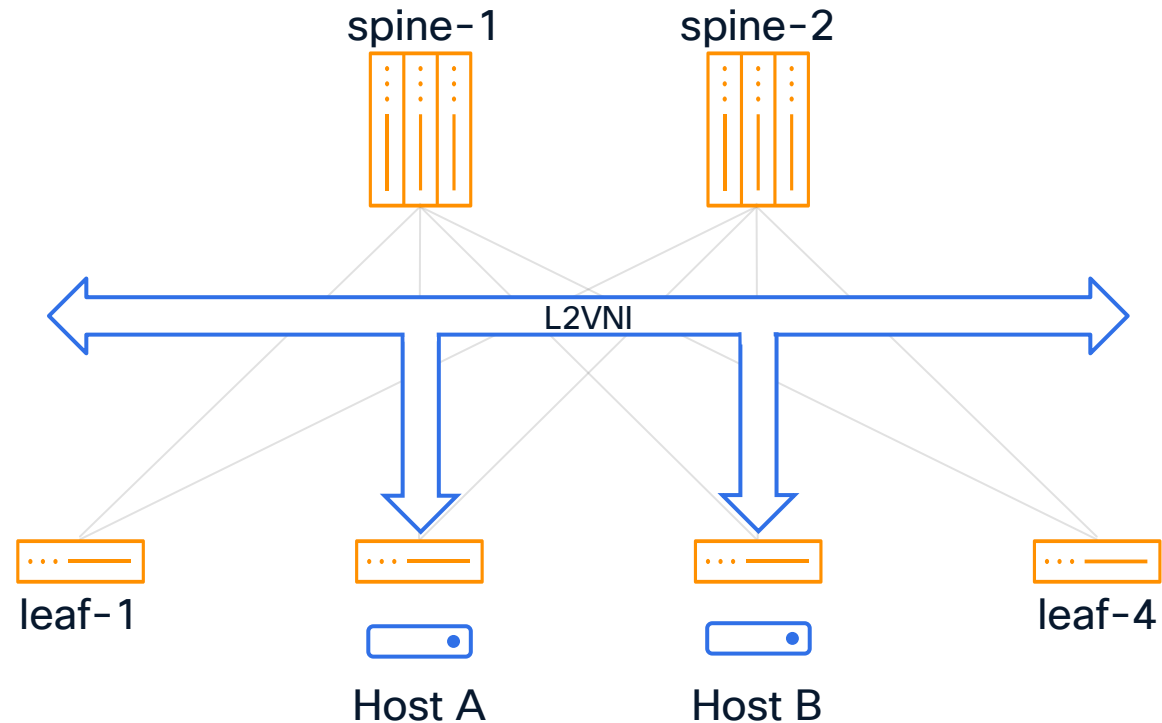


Ansible Concepts

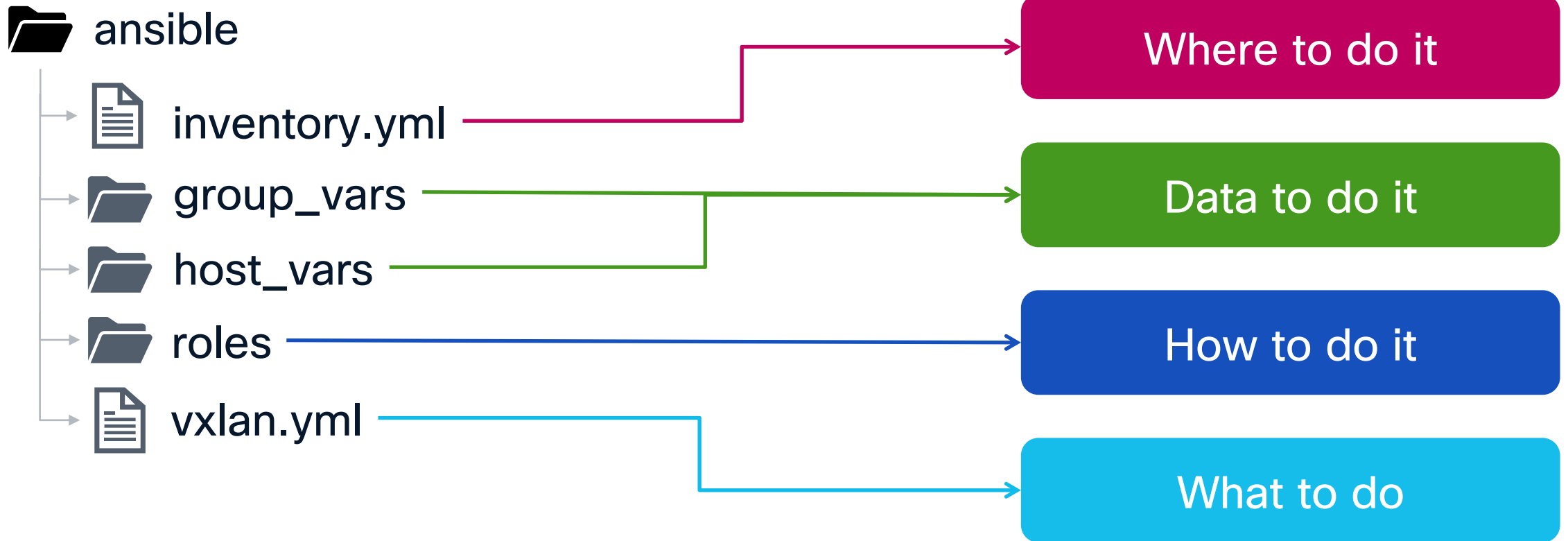
VXLAN EVPN Fabric Anatomy

Overlay configurations:

- VRFs
- Broadcast Domains
- SVIs (Distributed Anycast Gateway)
- External Connectivity
- Group Policy Option



Ansible Directory Structure



Ansible Inventory

Where to do it



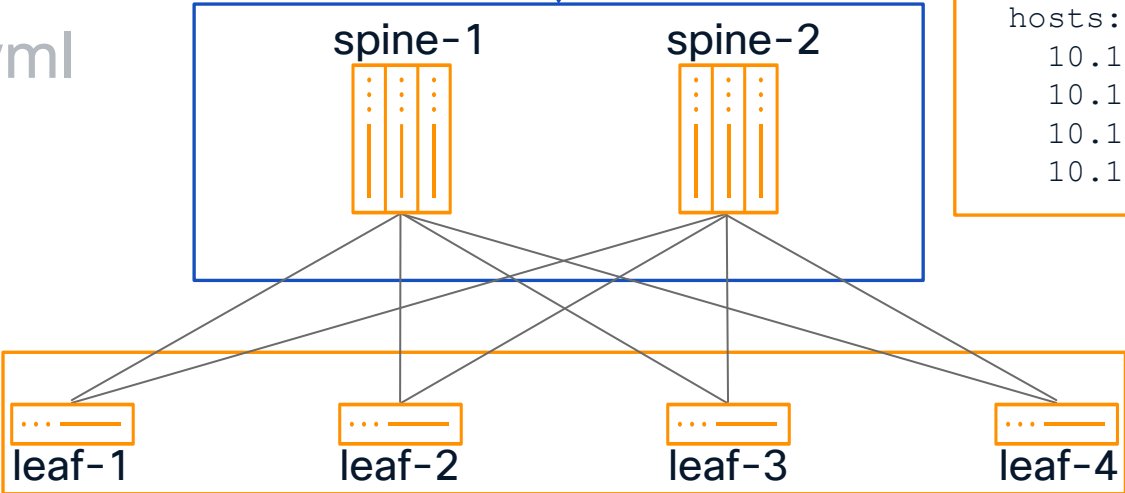
```
---
# main inventory file

all:
  vars:
    ansible_connection: ansible.netcommon.network_cli
    ansible_user: "nxos_username"
    ansible_password: "nxos_password"
    ansible_network_os: cisco.nxos.nxos
```

```
children:
  spines:
    hosts:
      10.15.1.11:
      10.15.1.12:
```

```
    leaves:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
        10.15.1.16:
```

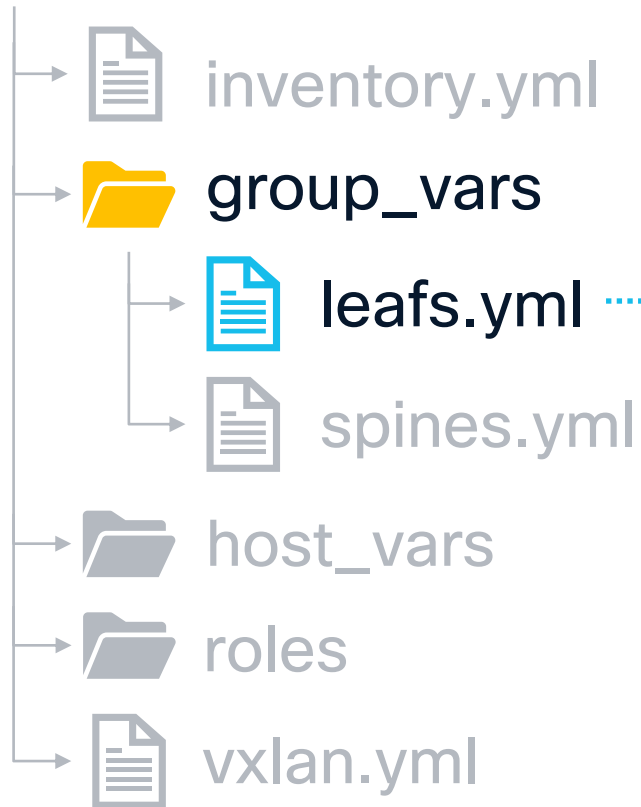
Connection information for switches



Ansible Group Vars

Data to do it

ansible



```
---
# var file for leafs group

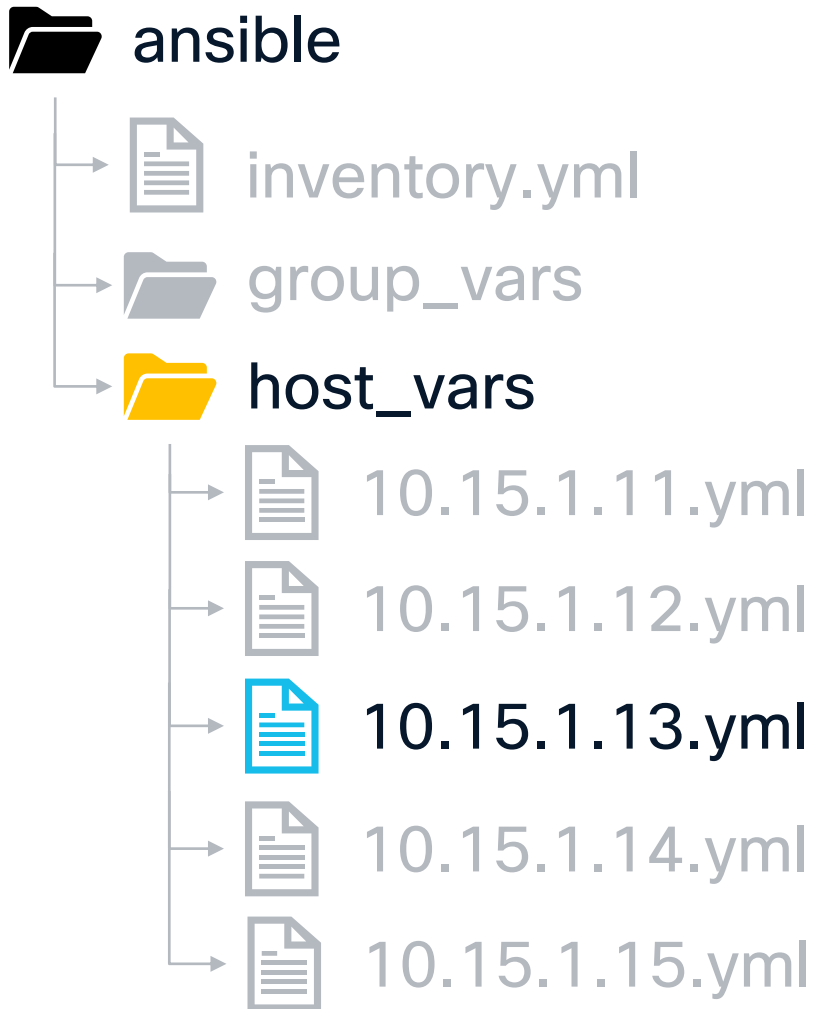
features:
  - ospf
  - pim
  - bgp
  - nv overlay
  - vn-segment-vlan-based
  - interface-vlan

networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

- contains data common to specified group
- match inventory group
- group_vars files are named and referenced after inventory groups

Ansible Host Vars

Data to do it



```
---
# vars file for L1

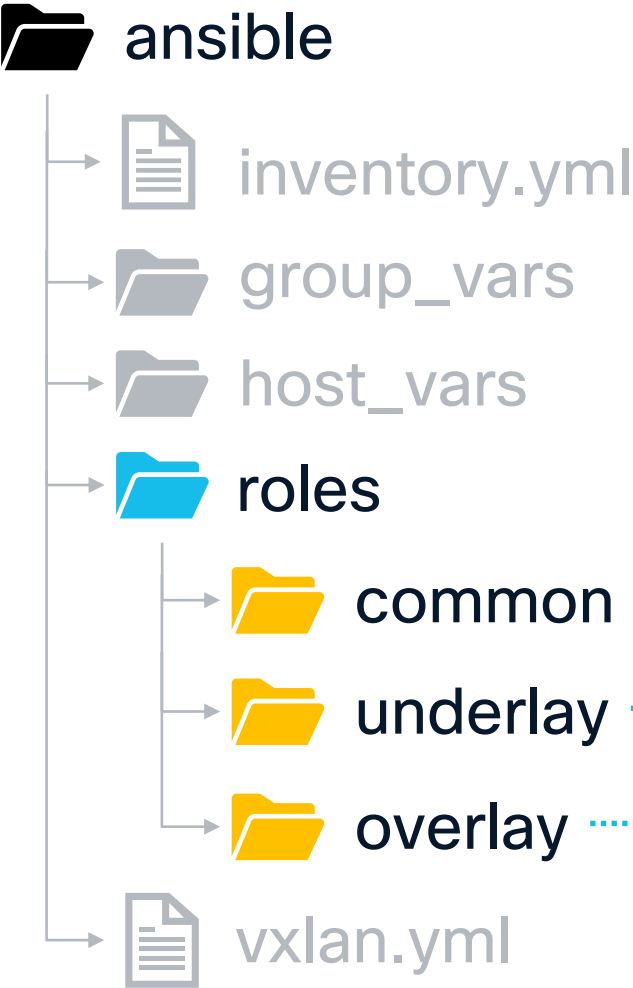
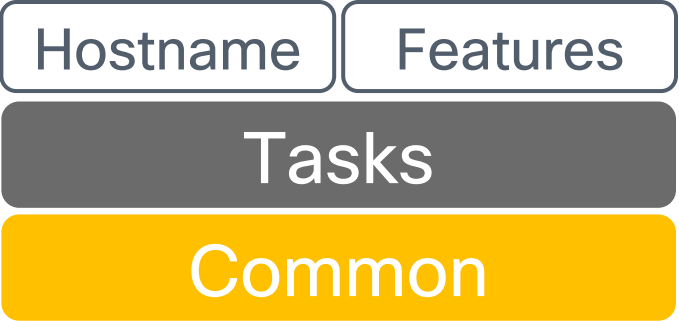
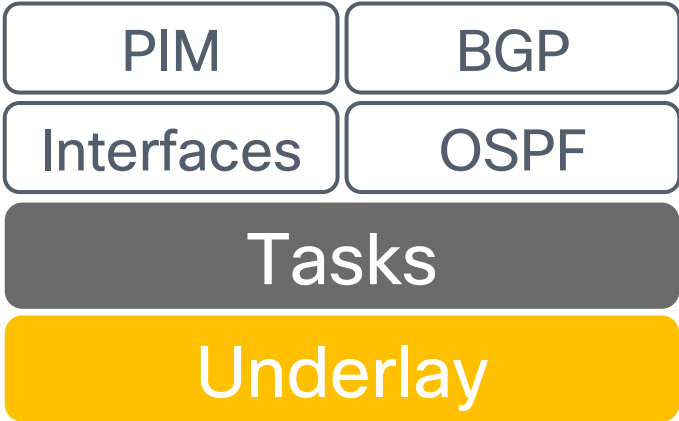
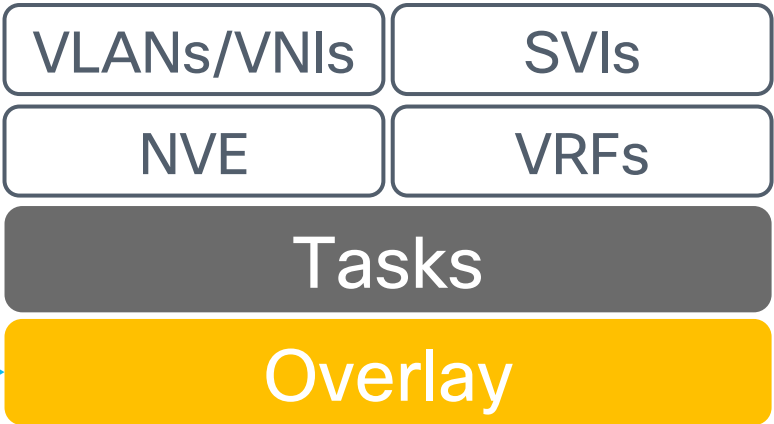
hostname: L1

layer3_physical_interfaces:
  - interface: ethernet1/11
    description: To S1 Eth1/1
    mode: layer3
    ip_address: 10.1.1.1
    mask: 31
    mtu: 9216
  - interface: ethernet1/12
    description: To S2 Eth1/1
    mode: layer3
    ip_address: 10.2.2.1
    mask: 31
    mtu: 9216
```

- contains data specific to that device
- match inventory name
- host_vars files are named and referenced after device IP address or FQDN

Ansible Roles – Tasks

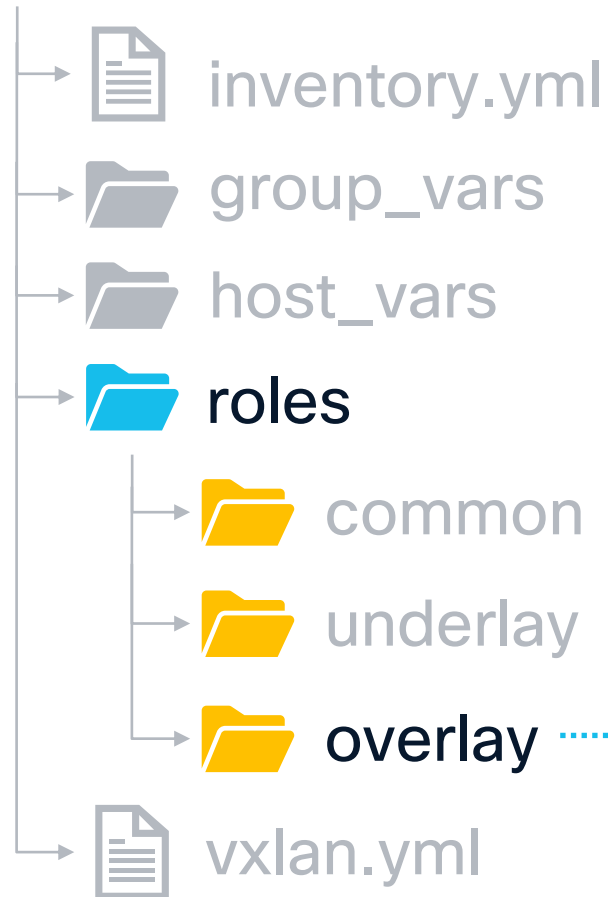
How to do it



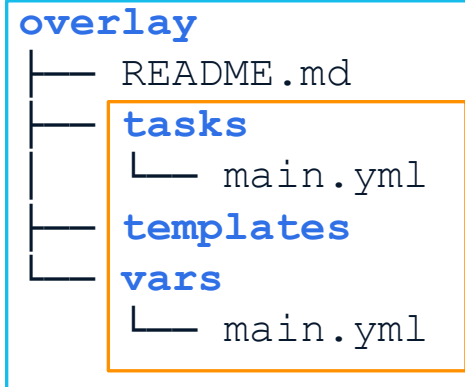
Ansible Roles

How to do it

ansible



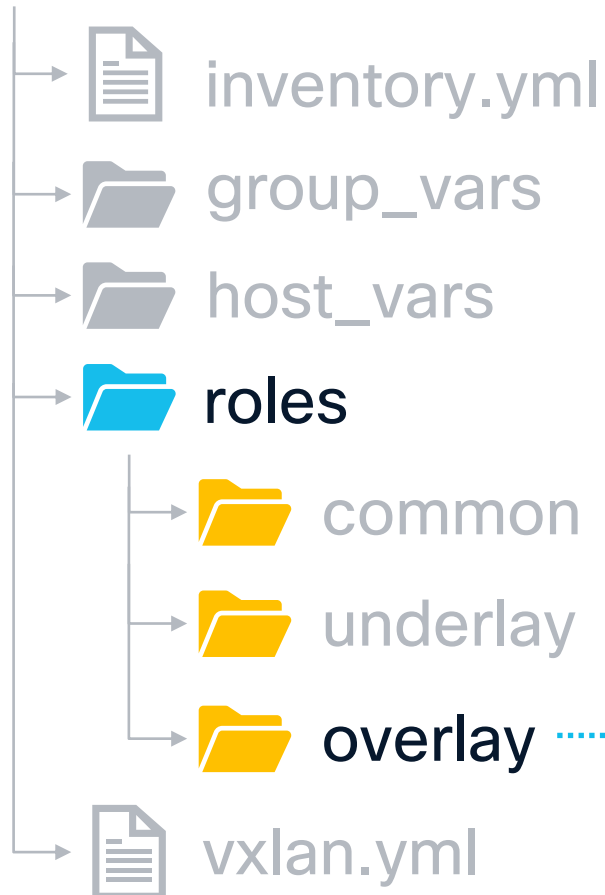
```
> ansible-galaxy init overlay
- Role overlay was created successfully
> tree overlay
```



Ansible Roles – Tasks

How to do it

ansible



roles/overlay/tasks/main.yml

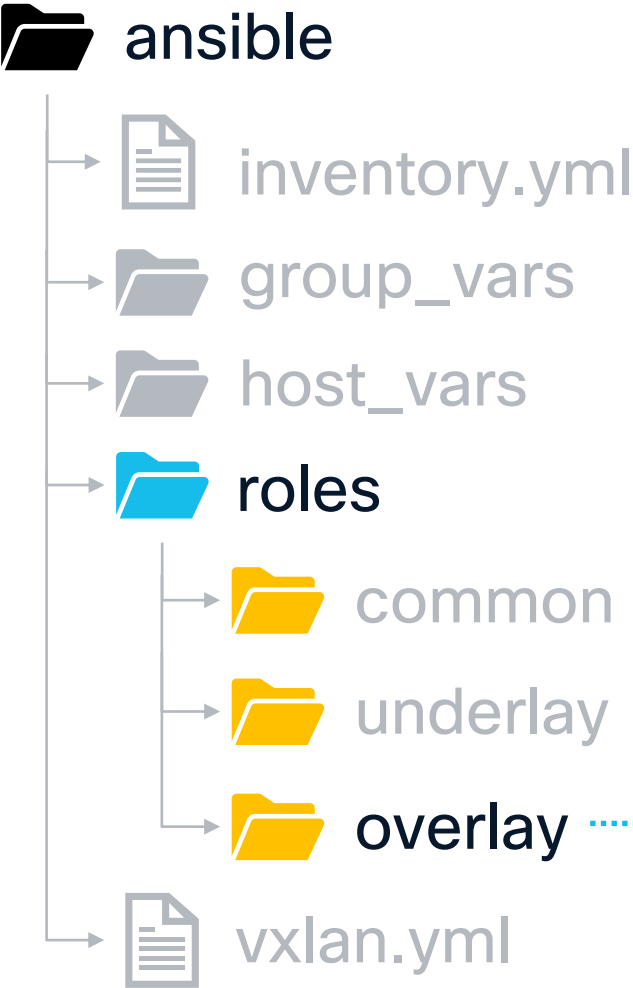
```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: Web Servers
        vlan_id: 101
        mapped_vni: 10101
      - name: DB Servers
        vlan_id: 102
        mapped_vni: 10102
      - name: vMotion
        vlan_id: 103
        mapped_vni: 10103
    state: overridden
<snip>
```

- Defines VLAN-to-VNI mappings in config list block
- Config block allows for YAML list of dictionary objects

Ansible Roles – Tasks

How to do it



```
roles/overlay/tasks/main.yml
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: Web Servers
        vl_id: 101
        mapped_vni: 10101
      - name: DB Servers
        vl_id: 102
        mapped_vni: 10102
      - name: vMotion
        vl_id: 103
        mapped_vni: 10103
    state: overridden
<snip>
```

Do not set task values and/or configurations inside of the role

Ansible Playbook

What to do



```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false
  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false
  roles:
    - role: overlay
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
<snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leaves:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leaves
  gather_facts: false
  roles:
    - role: common
    - role: underlay

- hosts: leaves
  gather_facts: false
  roles:
    - role: overlay
```

How to do it



roles/common/tasks/main.yml

```
---
# tasks file for roles/common

- name: Configure Hostname
  cisco.nxos.nxos_hostname:
    config:
      hostname: "{{ hostname }}"
    state: overridden
<snip>
```



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
    loop: "{{ networks }}"
<snip>
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
<snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/common/tasks/main.yml

```
---
# tasks file for roles/common

- name: Configure Hostname
  cisco.nxos.nxos_hostname:
    config:
      hostname: "{{ hostname }}"
    state: overridden
<snip>
```



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
      loop: "{{ networks }}"
<snip>
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
<snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leaves:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leaves
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leaves
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/common/tasks/main.yml

```
---
# tasks file for roles/common

- name: Configure Hostname
  cisco.nxos.nxos_hostname:
    config:
      hostname: "{{ hostname }}"
    state: overridden
<snip>
```



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
    loop: "{{ networks }}"
<snip>
```

Data to do it



host_vars/10.15.1.13.yml

```
---
# var file for 10.15.1.13

hostname: L1
<snip>
```



group_vars/leaves.yml

```
---
# var file for leaves group
<snip>
networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
  <snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leaves:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leaves
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leaves
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/common/tasks/main.yml

```
---
# tasks file for roles/common

- name: Configure Hostname
  cisco.nxos.nxos_hostname:
    config:
      hostname: "{{ hostname }}"
    state: overridden
  <snip>
```



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
    loop: "{{ networks }}"
  <snip>
```

Data to do it



host_vars/10.15.1.13.yml

```
---
# var file for 10.15.1.13

hostname: L1
<snip>
```



group_vars/leaves.yml

```
---
# var file for leaves group
<snip>
networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
  <snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leaves:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leaves
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leaves
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/common/tasks/main.yml

```
---
# tasks file for roles/common

- name: Configure Hostname
  cisco.nxos.nxos_hostname:
    config:
      hostname: "{{ hostname }}"
    state: overridden
  <snip>
```



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
    loop: "{{ networks }}"
  <snip>
```

Data to do it



host_vars/10.15.1.13.yml

```
---
# var file for 10.15.1.13

hostname: L1
<snip>
```



group_vars/leaves.yml

```
---
# var file for leaves group
<snip>
networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
<snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Generate VLAN Config Payload
  ansible.builtin.set_fact:
    nxos_vlans: |
      "{{ lookup('template', 'vlans.j2') }}"

- name: Configure VLANs
  cisco.nxos.nxos_vlans:
    config: "{{ nxos_vlans | from_yaml }}"
    state: overridden
<snip>
```



roles/overlay/templates/vlans.j2

```
{% for network in networks %}
- vlan_name: {{ network.vlan_name }}
  vlan_id: {{ network.vlan_id }}
  mapped_vni: {{ network.vni_id }}
{% endfor %}
```

Data to do it



host_vars/10.15.1.13.yml

```
---
# var file for 10.15.1.13

hostname: L1
<snip>
```



group_vars/leafs.yml

```
---
# var file for leafs group
<snip>
networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

Putting It All Together

Where to do it



inventory.yml

```
---
# main inventory file

all:
<snip>
  children:
    spines:
      hosts:
        10.15.1.11:
        10.15.1.12:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do



vxlan.yml

```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false

  roles:
    - role: overlay
```

How to do it



roles/overlay/tasks/main.yml

```
---
# tasks file for roles/overlay

- name: Generate VLAN Config Payload
  ansible.builtin.set_fact:
    nxos_vlans: |
      "{{ lookup('template', 'vlans.j2') }}"

- name: Configure VLANs
  cisco.nxos.nxos_vlans:
    config: "{{ nxos_vlans | from_yaml }}"
    state: overridden
<snip>
```



roles/overlay/templates/vlans.j2

```
{% for network in networks %}
- vlan_name: {{ network.vlan_name }}
  vlan_id: {{ network.vlan_id }}
  mapped_vni: {{ network.vni_id }}
{% endfor %}
```

Data to do it



host_vars/10.15.1.13.yml

```
---
# var file for 10.15.1.13

hostname: L1
<snip>
```



group_vars/leafs.yml

```
---
# var file for leafs group
<snip>
networks:
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet1
    vlan_id: 101
    vni_id: 10101
    ip: 10.1.101.1/24
  - vrf_name: AnsibleVRF
    vlan_name: AnsibleNet2
    vlan_id: 102
    vni_id: 10102
    ip: 10.1.102.1/24
```

A word about Ansible Variables

- Can be defined in many different places
 - Most commonly in group_vars and host_vars directory
- Created dynamically during runtime
- Used in task modules, conditional logic, templates, etc
- Jinja2 syntax used to reference
- Variable precedence is used

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#understanding-variable-precedence

Variables with Jinja2 Syntax

- Variable substitution in tasks
- Uses double curly braces wrapped in quotes: "{{ }}"

```
# tasks file for roles/overlay
```

```
- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
  config:
    - name: "{{ vlan_name }}"
      vlan_id: "{{ vlan_id }}"
      mapped_vni: "{{ vni_id }}"
```

```
# vars defined somewhere in Ansible
```

```
vlan_name: Web Servers
vlan_id: 101
vni_id: 10101
```

Set of
Key/Value Pairs

Ansible Variable Lists

Go from this...



To this...

```
# vars defined somewhere in Ansible
```

```
vlan_name: Web Servers  
vlan_id: 101  
vni_id: 10101
```

Sequential list of three
dictionary objects
containing VLAN
information that can be
referenced iteratively

```
# vars defined somewhere in Ansible
```

```
networks:
```

```
- vlan_name: Web Servers  
  vlan_id: 101  
  vni_id: 10101  
  
- vlan_name: DB Servers  
  vlan_id: 102  
  vni_id: 10102  
  
- vlan_name: vMotion  
  vlan_id: 103  
  vni id: 10103
```

Ansible Loop with Jinja2 Syntax

Use Ansible loop to iterate data for a tasks

```
# tasks file for roles/overlay

- name: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      - name: "{{ item.vlan_name }}"
        vlan_id: "{{ item.vlan_id }}"
        mapped_vni: "{{ item.vni_id }}"
    loop: "{{ networks }}"

# vars defined somewhere in Ansible
networks:
  - vlan_name: Web Servers
    vlan_id: 101
    vni_id: 10101
  - vlan_name: DB Servers
    vlan_id: 102
    vni_id: 10102
  - vlan_name: vMotion
    vlan_id: 103
    vni_id: 10103
```

The diagram illustrates the Jinja2 templating process in an Ansible task. On the left, the task configuration is shown with variables like `item.vlan_name`, `item.vlan_id`, `item.vni_id`, and `networks` highlighted in colored boxes. On the right, the `networks` variable is defined as a list of dictionaries. Arrows connect the highlighted variables in the task to their corresponding values in the `networks` list: `item.vlan_name` points to the `vlan_name` field, `item.vlan_id` points to the `vlan_id` field, `item.vni_id` points to the `vni_id` field, and `networks` points to the entire list of dictionaries.

A word about Jinja Templating

- Leverage module and filters from Ansible Builtin collection
- Create template file(s) in a role's template directory: .j2 file extension

tasks file for roles/overlay

```
- name: Generate VLAN Config Payload
  ansible.builtin.set_fact:
```

```
  nxos_vlans: "{{ lookup('template', 'vlans.j2') }}"
```

```
- name: Configure VLANs
```

```
  cisco.nxos.nxos_vlans:
```

```
    config: "{{ nxos_vlans | from_yaml }}"
    state: merged
```

config data passed to task

```
- vlan_id: 101
- Vlan_id: 102
```

group_vars/leafs.yml

```
networks:
```

```
- vlan_name: Web Servers
  vlan_id: 101
  vni_id: 10101
- vlan_name: DB Servers
  vlan_id: 102
  vni_id: 10102
```

roles/overlay/templates/vlans.j2

```
{% for network in networks %}
- vlan_id: {{ network.vlan_id }}
{% endfor %}
```

Additional Details for NXOS

Two Types of Modules for NXOS

Legacy Modules

- Inconsistent across different network devices
- Requires task loops for more than one configuration item
- Simple states, present or absent

Resource Modules

- Consistent across different network devices
- Can leverage task loops or Jinja2 templating for config blocks
- Introduces new states for Ansible to be the source of truth

A word about Resource Module States

- Merged - Ansible merges the on-device configuration with the provided configuration in the task.
- Replaced - Ansible replaces the on-device configuration subsection with the provided configuration subsection in the task.
- Overridden - Ansible overrides the on-device configuration for the resource with the provided configuration in the task. Use caution with this state as you could remove your access to the device (for example, by overriding the management interface configuration).
- Deleted - Ansible deletes the on-device configuration subsection and restores any default settings.

NXOS Config Fallback Module

- Allows passing direct cli configuration
- Can take full running-config backup, e.g. before a change operation
- Perform a save operation, i.e. "copy run start"

- name: Configure PIM Anycast RP

`cisco.nxos.nxos_config:`

lines:

- "ip pim anycast-rp {{ s1_loopback1 }} {{ s1_loopback0 }}"

- "ip pim anycast-rp {{ s2_loopback1 }} {{ s2_loopback0 }}"

save_when: modified

Options:

- always – copy always
- modified – copy only if changed since last save
- changed – copy only if the **task** made a change
- never – never copy

NXOS Config Fallback Module

- Allows sending arbitrary commands, e.g. show commands
- Supports prompt handling
- Can handle list of commands or prompts

- name: Get Show Commands

`cisco.nxos.nxos_command:`

commands:

- show version
- show ip ospf neighbor
- show ip pim neighbor

- name: Misc Commands

`cisco.nxos.nxos_command:`

commands: copy ftp://nxos.bin bootflash:

prompt:

- "Username:"
- "Password:"

answer:

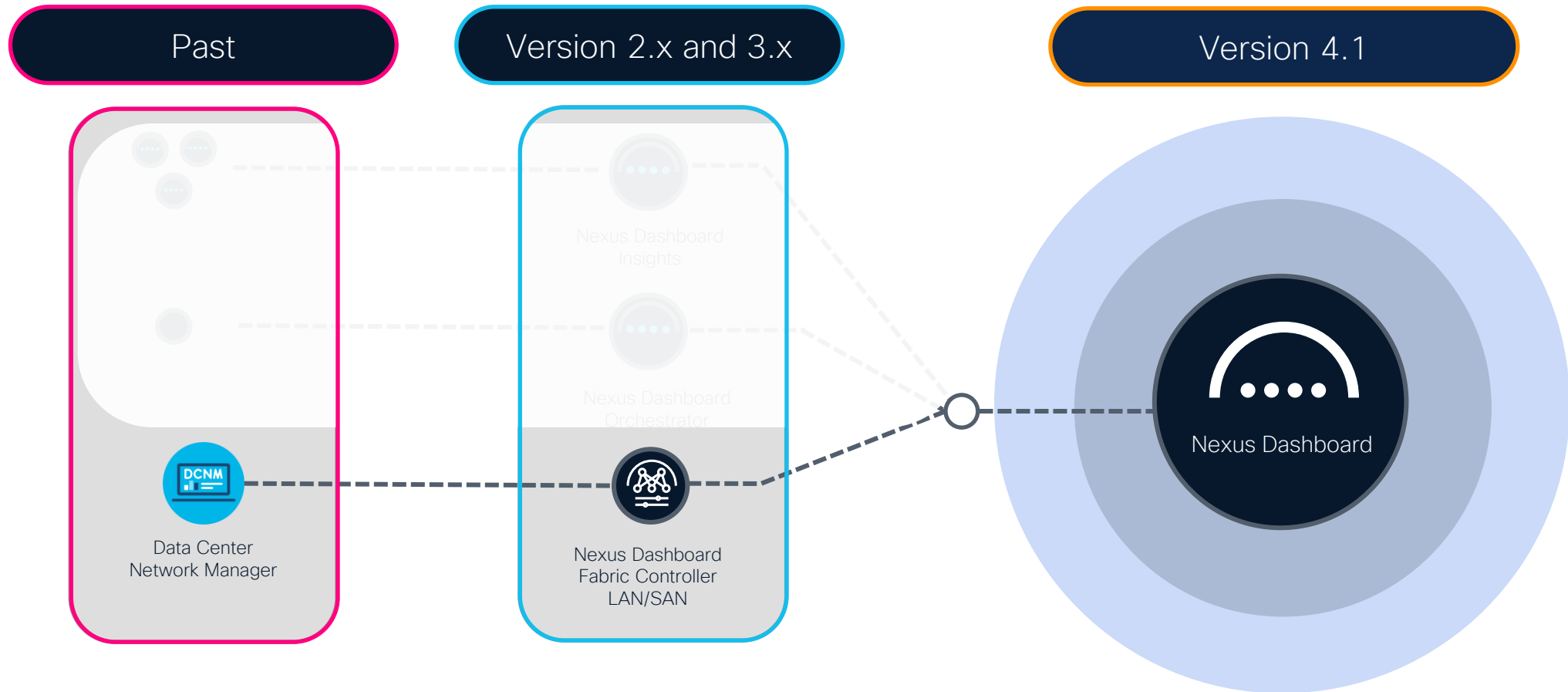
- <username>
- <password>

Infrastructure as Code with Nexus Dashboard and Ansible

Nexus Dashboard Recap

Cisco Nexus Dashboard

Formerly Called (NDFC), Formerly called (DCNM)



Nexus Dashboard

The Most **Powerful** Datacenter Platform

Management

Define the intent for a distributed fabric in a centralized orchestrator



Automation

Configure once centrally, deploy anywhere



Day2Ops

Insights, Assurance, Troubleshooting and Monitoring from a single pane of glass



Nexus Dashboard

The Most **Powerful** Datacenter Platform

Management

Define the intent for a distributed fabric in a centralized orchestrator



Automation

Configure once centrally, deploy anywhere



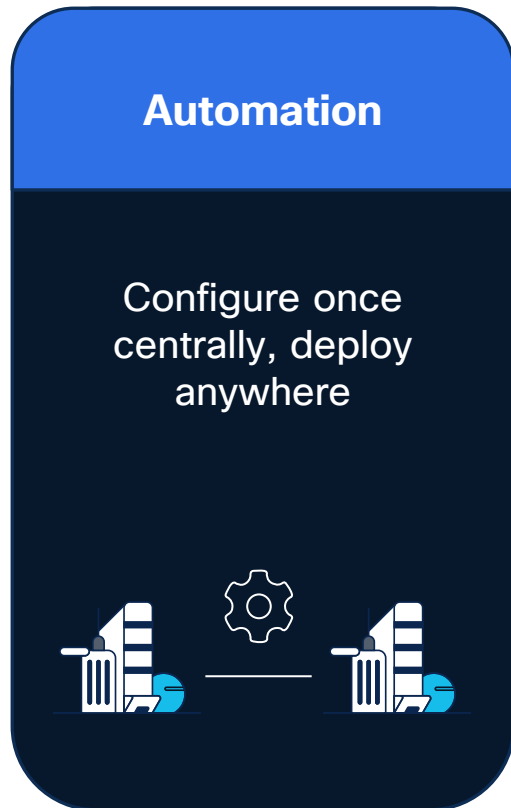
Day2Ops

Insights, Assurance, Troubleshooting and Monitoring from a single pane of glass



Nexus Dashboard

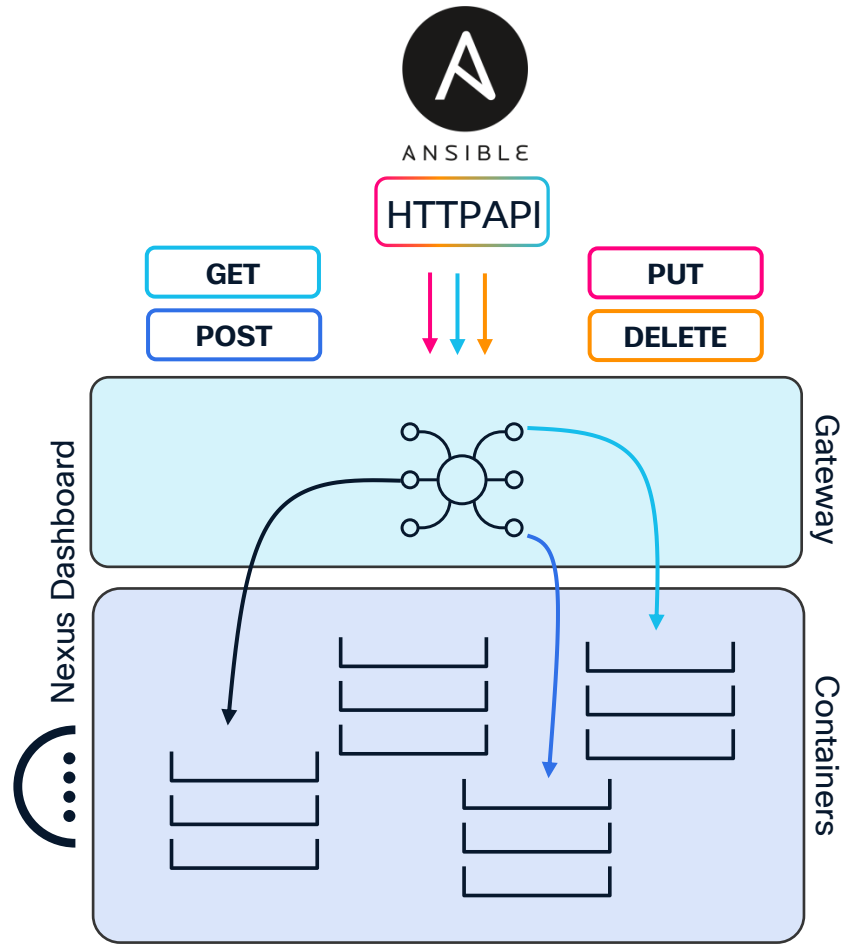
The Most **Powerful and Automated** Datacenter Platform



- ND native automation capabilities are calling out functionalities that allow large fabrics management via a single controller
- Additional automation capabilities can be integrated in the controller thanks to the API native architecture
 - Standardize deployment processes
 - Avoid ClickOps
 - Integrate additional applications effortlessly
 - Deploy using Infrastructure as Code(IaC)
 - Define and manage automation pipelines

Nexus Dashboard

An API Gateway For All Applications



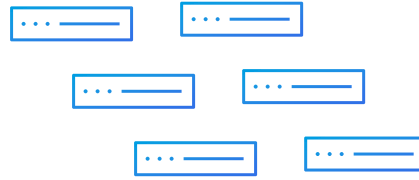
- APIs are the north-bound automation primer interface for Nexus Dashboard
 - They are Ansible's cisco.dcnm collection only interface to ND
- APIs are available by default, no need to enable them
- Exposed via the easy to consume REST framework
- Nexus Dashboard acts as central gateway for accessing the upper layer applications
 - Clients connect with ND, which routes the REST call to the appropriate component/container
- Authentication and RBAC available in Web UI

Bringing Up a VXLAN EVPN With Nexus Dashboard

Five Basic Steps



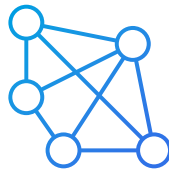
Define The Fabric



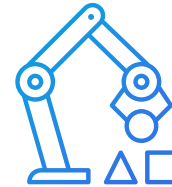
Onboard Devices



Configure Interfaces



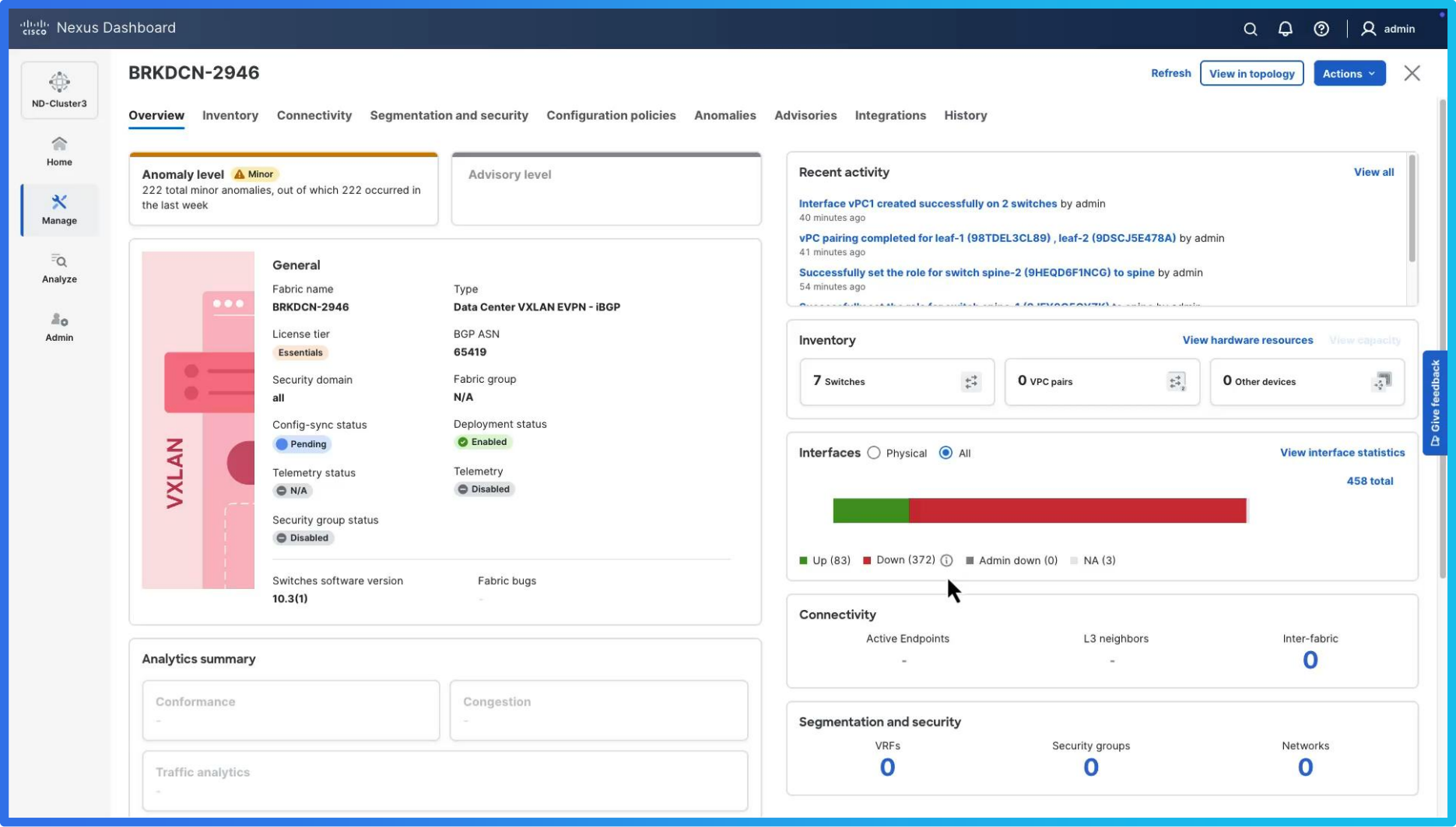
Create Overlays



Deploy

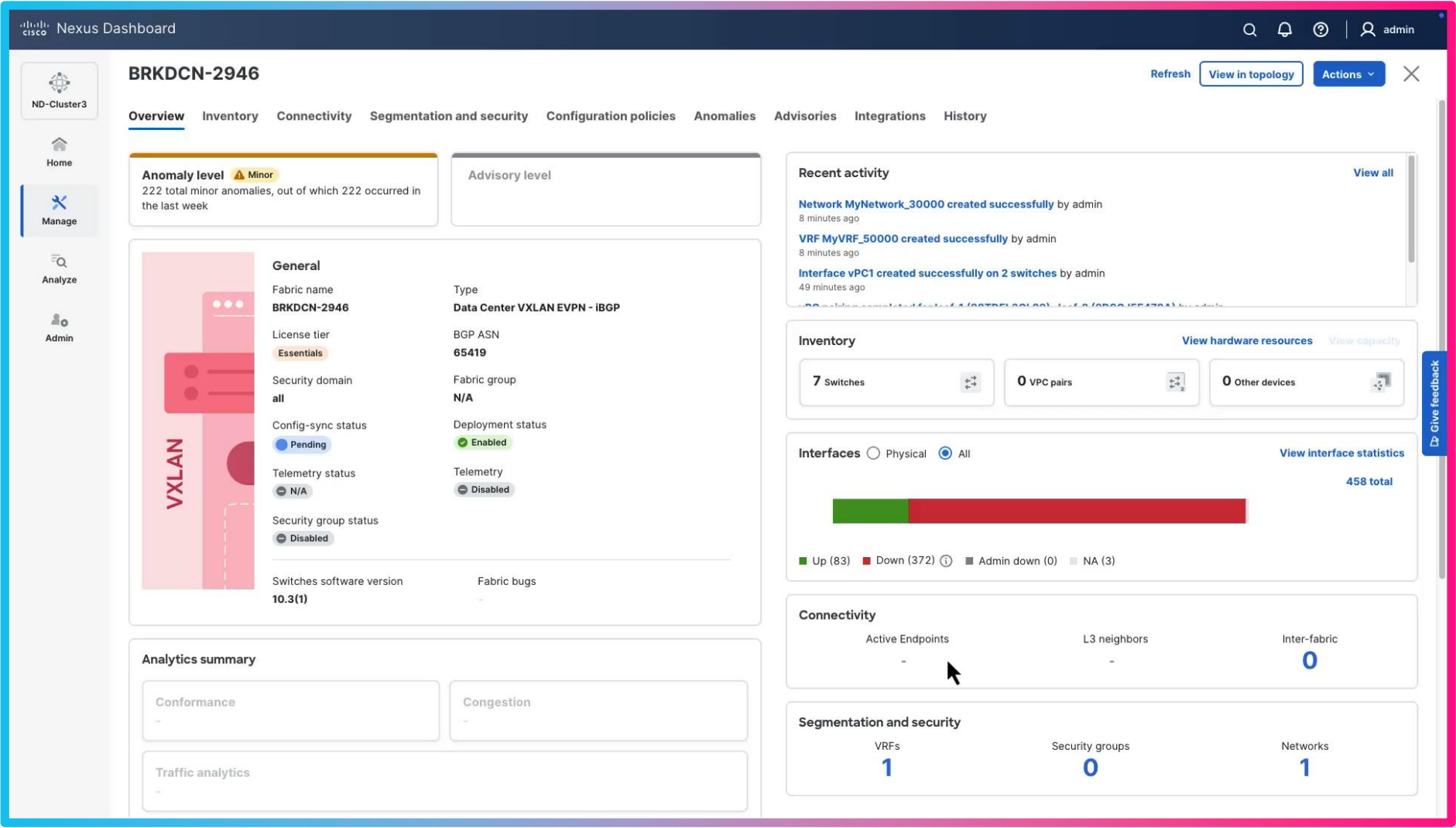
Bringing Up a VXLAN EVPN With Nexus Dashboard

Create Overlays



Bringing Up a VXLAN EVPN With Nexus Dashboard

Deploy



Ansible's DCNM Collection

DCNM Collection Modules – Toolbox

Ansible Galaxy

Search

Collections

Collections

Namespaces

Roles

Documentation

Terms of Use

Documentation (1)

Readme

Modules (20)

dcnm_bootflash

dcnm_fabric

dcnm_image_policy

dcnm_image_upgrade

dcnm_image_upload

dcnm_interface

dcnm_inventory

dcnm_links

dcnm_log

dcnm_maintenance_mode

dcnm_network

dcnm_policy

dcnm_resource_manager

dcnm_rest

dcnm_service_node

dcnm_service_policy

dcnm_service_route_peering

dcnm_template

dcnm_vpc_pair

dcnm_vrf

Synopsis

Parameters

• Synopsis

• Parameters

• Examples

Synopsis

• Add and remove Networks from a DCNM managed VXLAN fabric.

• In Multisite fabrics, Networks can be created only on Multisite fabric

Parameters

Parameter	Choices / Defaults	Comments
config list / elements=dictionary		List of details of networks being managed. Not required for state deleted
arp_suppress boolean	Choices: <div><div>true</div><div>false ←</div></div>	ARP suppression ARP suppression is only supported if SVI is present when Layer-2-Only is not enabled
attach list / elements=dictionary		List of network attachment details
deploy boolean	Choices: <div><div>true ←</div><div>false</div></div>	Per switch knob to control whether to deploy the attachment This knob has been deprecated from Ansible NDFC Collection Version 2.1.0 onwards. There will not be any functional impact if specified in playbook.
ip_address string / required		IP address of the switch where the network will be attached or detached
ports list / elements=string / required		List of switch interfaces where the network will be attached
tor_ports list / elements=dictionary		List of interfaces in the paired TOR switch for this leaf where the network will be attached Please attach the same set of TOR ports to both the VPC paired switches.
ip_address string / required		IP address of the TOR switch where the network will be attached

English

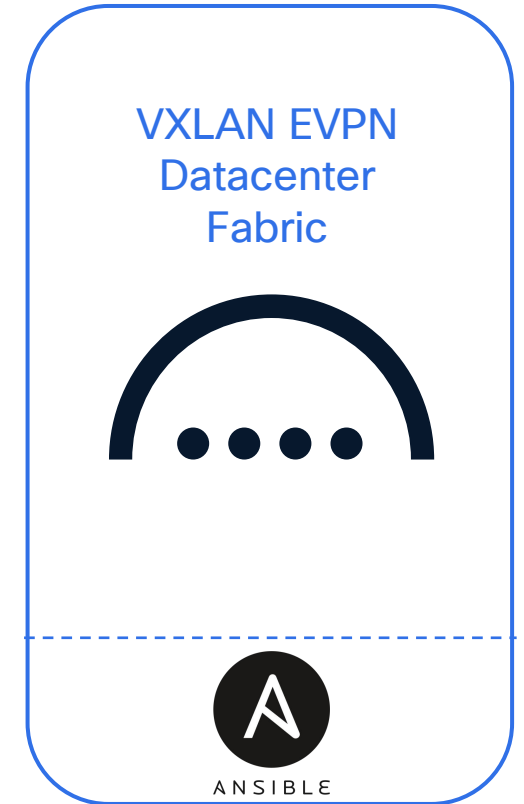
?

Login

20

Primary VXLAN EVPN Fabric Plugins

- ① `cisco.dcnm.dcnm_fabric`
- ② `cisco.dcnm.dcnm_inventory`
- ③ `cisco.dcnm.dcnm_vpc_pair`
- ④ `cisco.dcnm.dcnm_interface`
- ⑤ `cisco.dcnm.dcnm_vrf`
- ⑥ `cisco.dcnm.dcnm_network`
- ⑦ `cisco.dcnm.dcnm_policy`



Getting started

Ansible Collection Installation

Collection Location: <https://galaxy.ansible.com/cisco/dcnm>

Install Command:

```
* pip install ansible  
* ansible-galaxy collection install cisco.dcnm
```

Ansible uses the Fully Qualified Collection Name (FQCN)

Namespace: [cisco](#)

Collection Name: [dcnm](#)



**Let's Build Something With
ND and Ansible Together!**

Bringing Up a VXLAN EVPN With Nexus Dashboard

Clicking Around

 Nexus Dashboard

ND-Cluster3

Home

Manage

Analyze

Admin

← Fabrics

Create/Onboard Fabric

✓ Select a category
Create new LAN fabric

✓ Select a type
VXLAN

3 Settings
Default

4 Summary

5 Fabric creation

Settings

These are the recommended settings for configuring the parameters and capabilities of the new fabric.

Configuration mode ⓘ
☒ Default ☐ Advanced

Name *

Location *

BGP ASN *

1-4294967295 | 1-65535[.0-65535]

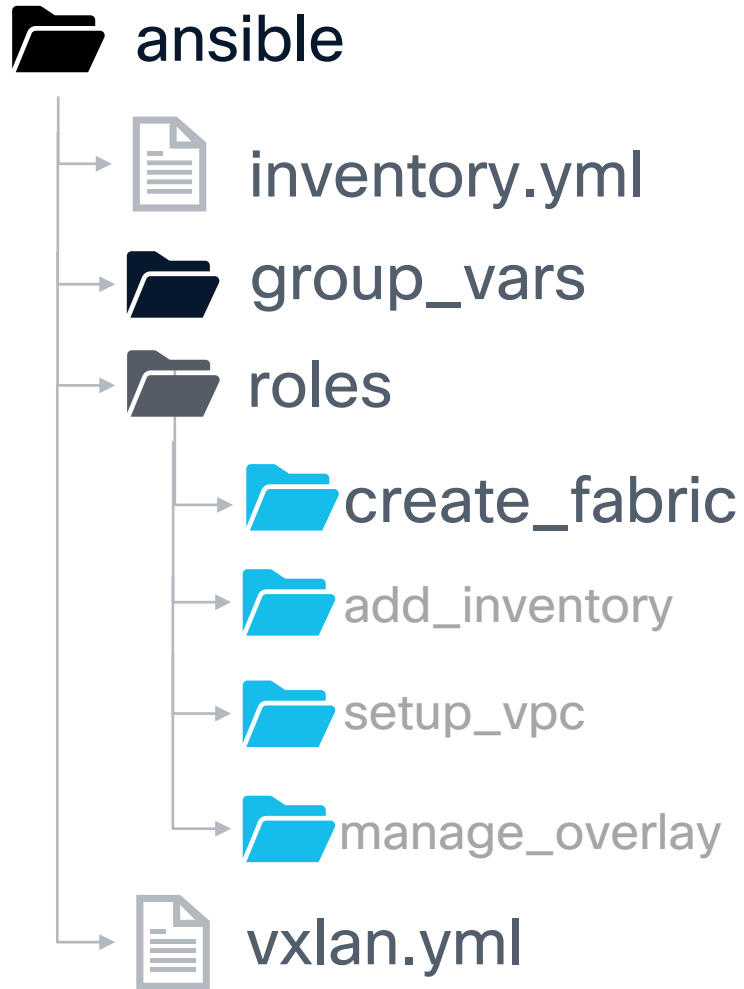
License tier for fabric ⓘ
☒ Essentials ☐ Advantage ☐ Premier

Enabled features
☐ Telemetry ⓘ

Cancel

Bringing Up a VXLAN EVPN With Nexus Dashboard

The IaC Way



```
roles/create_fabric/tasks/main.yml
---
# tasks file for roles/create_fabric

- name: Create Fabric with Module
  cisco.dcnm.dcnm_fabric:
    state: merged
    config:
      - FABRIC_NAME: "{{ fabric_name }}"
        FABRIC_TYPE: "{{ fabric_type }}"
        BGP_AS: "{{ fabric_as }}"
        ANYCAST_GW_MAC: "{{ dag_mac }}"
        ENABLE_PVLAN: "{{ pvlan_enable }}"
```

- Creates Fabric
- Handles Mutually Exclusive Properties
- Supports multiple types (VXLAN_EVPN, MSD, ISN, LAN_CLASSIC)

Bringing Up a VXLAN EVPN With Nexus Dashboard

The IaC Way

ansible

inventory.yml

group_vars

roles

create_fabric

add_inventory

setup_vpc

manage

vxlan.yml

roles/create_fabric/tasks/main.yml

```
---
# tasks file for roles/create

- name: Create Fabric
  cisco.dcnm.dcnm_fabric:
    state: merge
    config:
      fabric_name: "{{ fabric_name }}"
      fabric_type: "{{ fabric_type }}"
      fabric_as: "{{ fabric_as }}"
      dag_mac: "{{ dag_mac }}"
      pvlan_enable: "{{ pvlan_enable }}"
```

The Story Repeats

- Creates Fabric
- Handles Mutually Exclusive Properties
- Supports multiple types (VXLAN_EVPN, MSD, ISN, LAN_CLASSIC)

VXLAN as Code

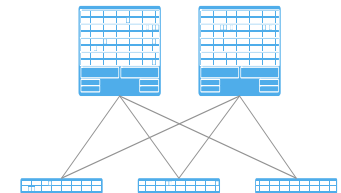
VXLAN as Code

Introduction

- **VXLAN as Code** speeds up network orchestration through simplification and abstraction
- Users focus on desired configurations without handling low-level device details
- Users deploy network fabrics in minutes with an easy, **opinionated data model**
- Data model is accessed by **VXLAN as Code** collection and that triggers the automation towards Nexus Dashboard

<https://netascode.cisco.com>

```
vxlan:
  overlay:
    networks:
      - name: NetworkDMZ
        is_l2_only: False
        vrf_name: VrfRed
        net_id: 33100
        vlan_id: 3100
        network_attach_group: dmz
```



VXLAN as Code

Data Model

- Becomes the fabric **source of truth**
- Built to represent network complexity in an easier structure
- Built by network engineers
- Can be used for **validation and testing**
- Definitions can be contained into a single file or spread across different ones
- Documentation available here: https://netascode.cisco.com/docs/data_models/vxlan/overview/

```
vxlan:  
  underlay:  
    general:  
      routing_protocol: ospf  
      enable_ipv6_underlay: false  
      replication_mode: multicast  
      fabric_interface_numbering: p2p  
      subnet_mask: 30  
      underlay_routing_loopback_id: 0  
      underlay_vtep_loopback_id: 1  
      underlay_routing_protocol_tag: UNDERLAY  
      underlay_rp_loopback_id: 254  
      intra_fabric_interface_mtu: 9216  
      layer2_host_interface_mtu: 9216  
      unshut_host_interfaces: True
```

VXLAN as Code - Building Blocks



Pre-Change
Validation

CI/CD



Automated
Testing

ChatOps



VXLAN Solution Data Model



VXLAN as Code Ansible Collection



ND Ansible Collection

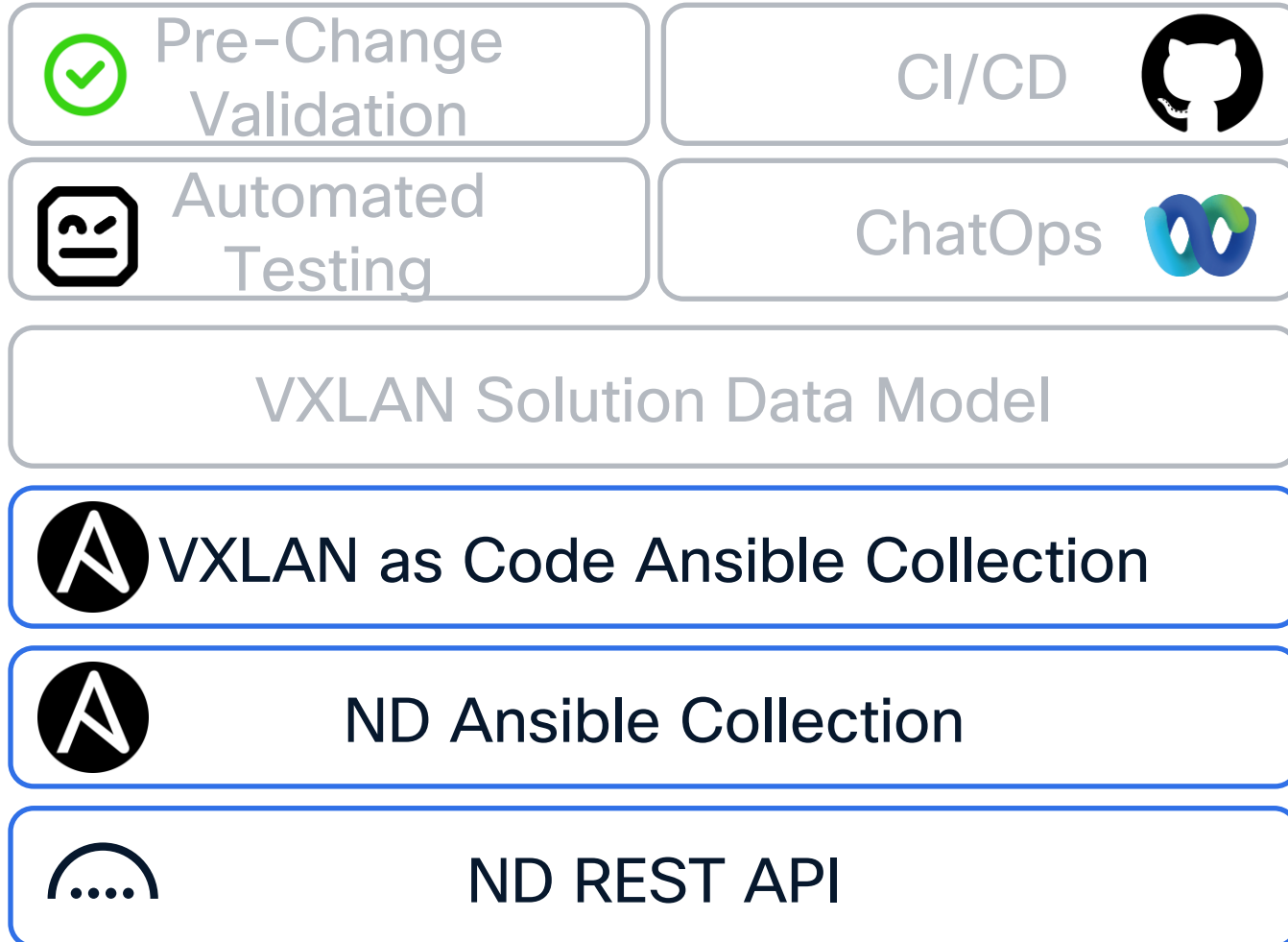
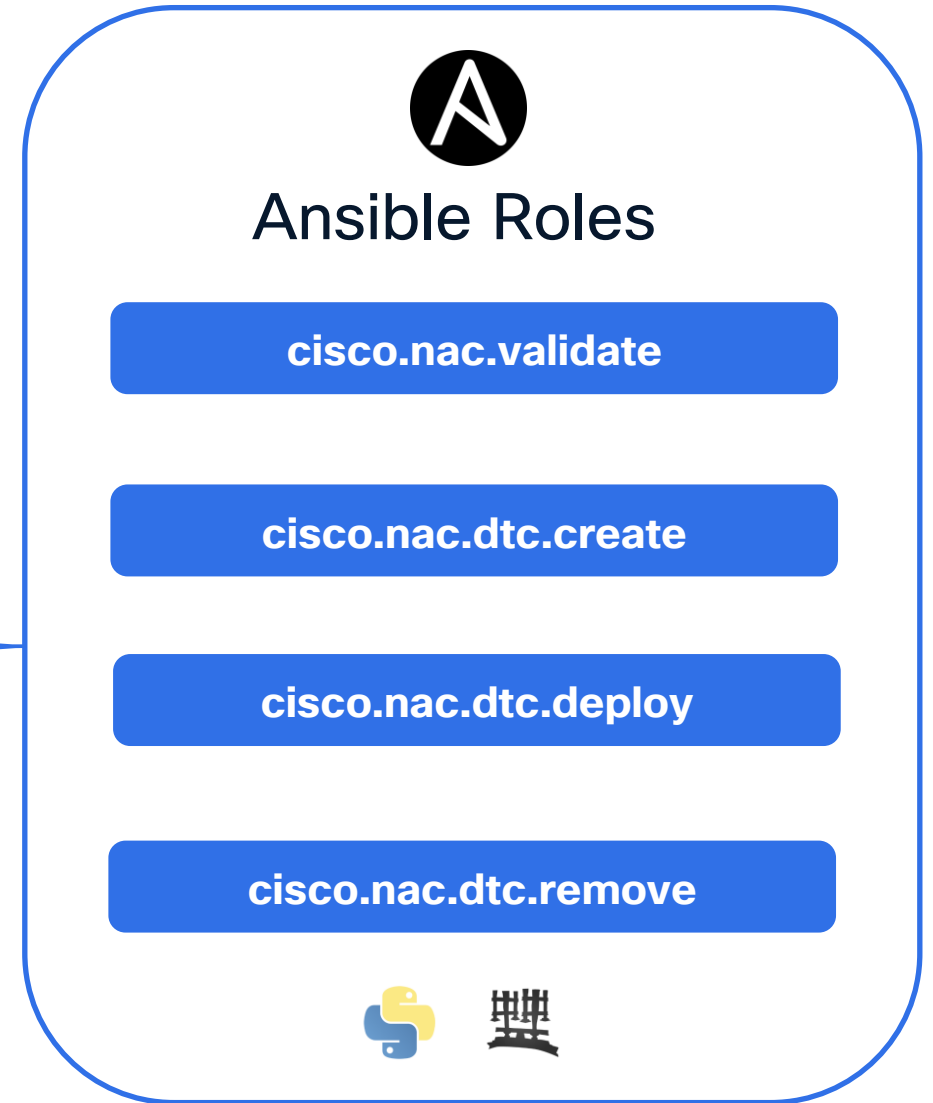


ND REST API

```
vxlan:
  underlay:
    general:
      routing_protocol: ospf
      enable_ipv6_underlay: false
      replication_mode: multicast
      fabric_interface_numbering: p2p
      subnet_mask: 30
      underlay_routing_loopback_id: 0
      underlay_vtep_loopback_id: 1
      underlay_routing_protocol_tag: UNDERLAY
      underlay_rp_loopback_id: 254
      intra_fabric_interface_mtu: 9216
      layer2_host_interfacde_mtu: 9216
      unshut_host_interfaces: True
```

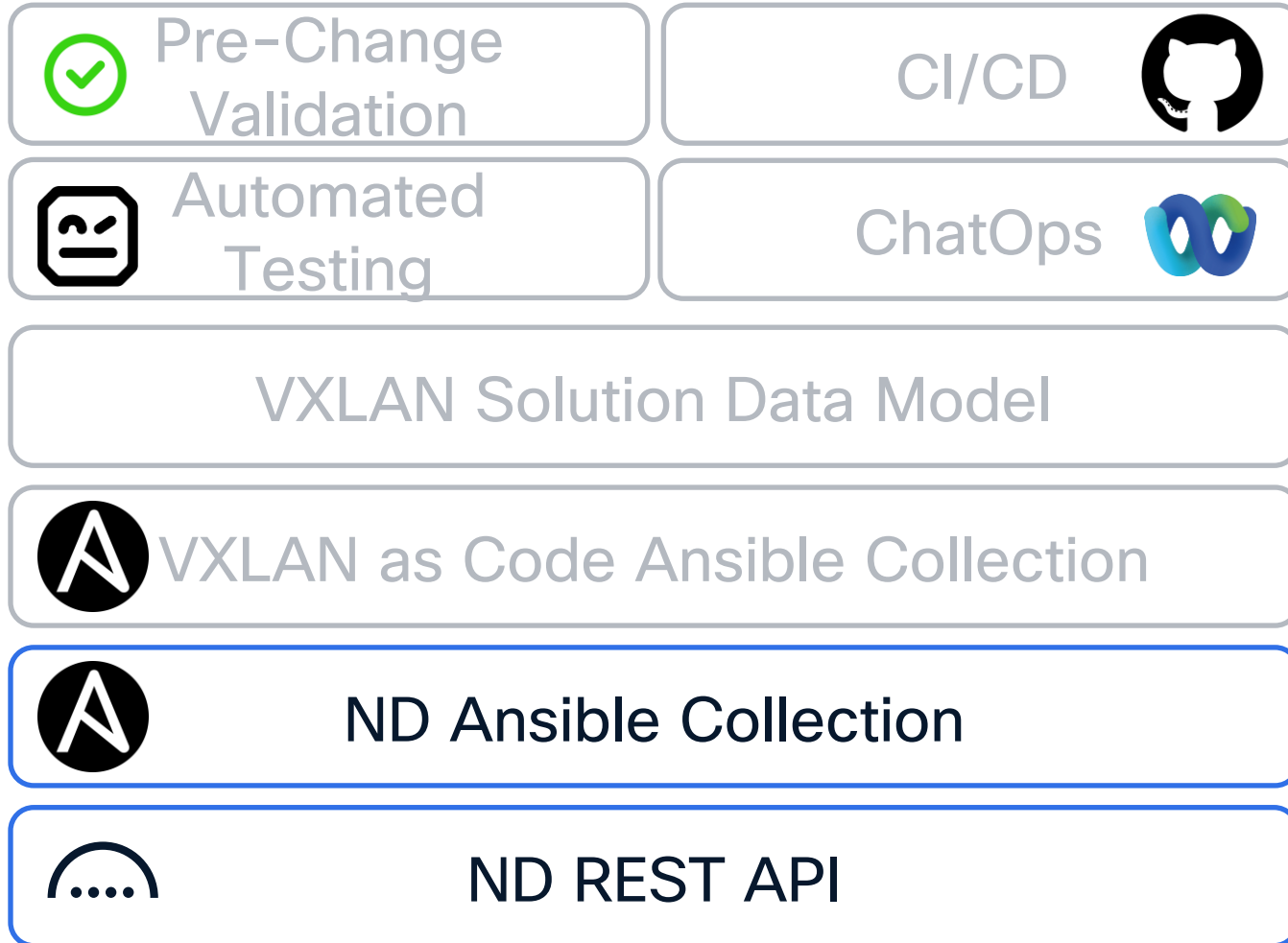
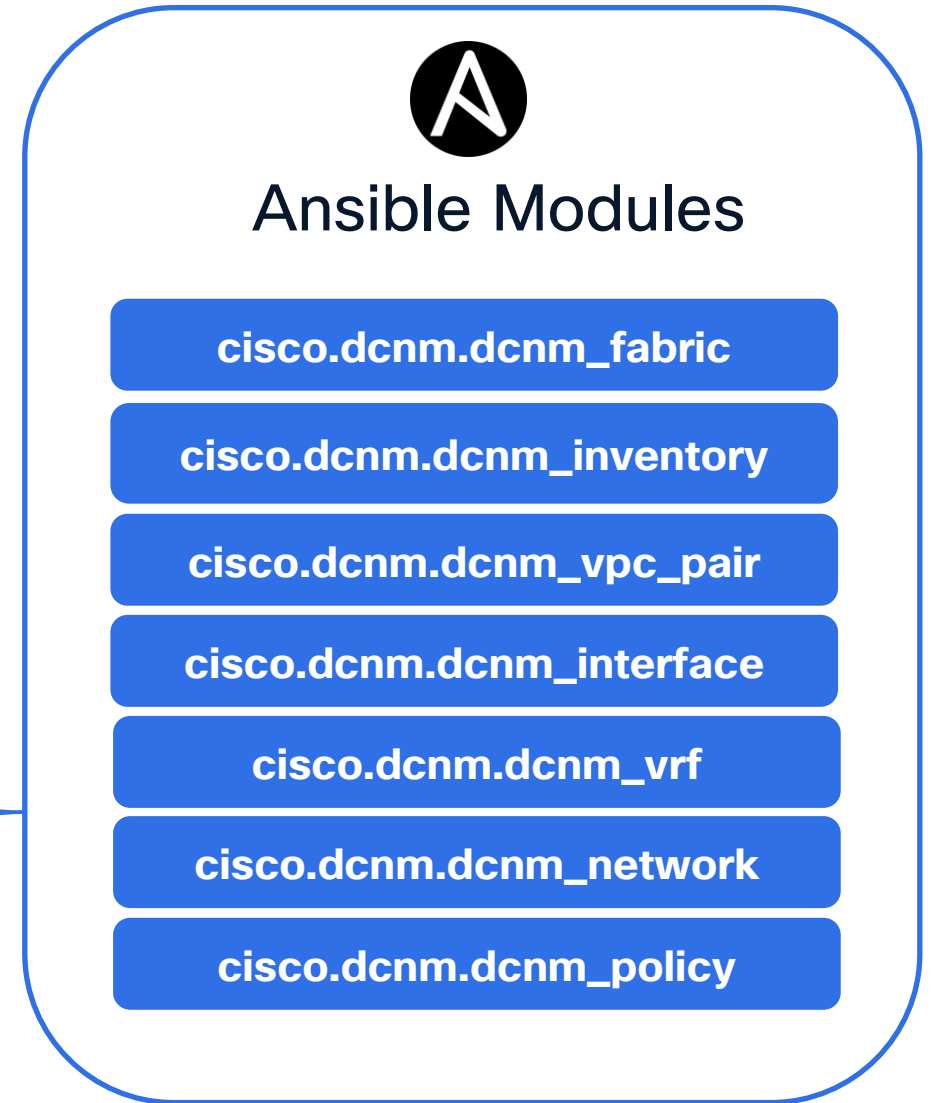
(VXLAN as Code Model Documentation) - https://netascode.cisco.com/docs/data_models/vxlan/overview/

VXLAN as Code - Building Blocks



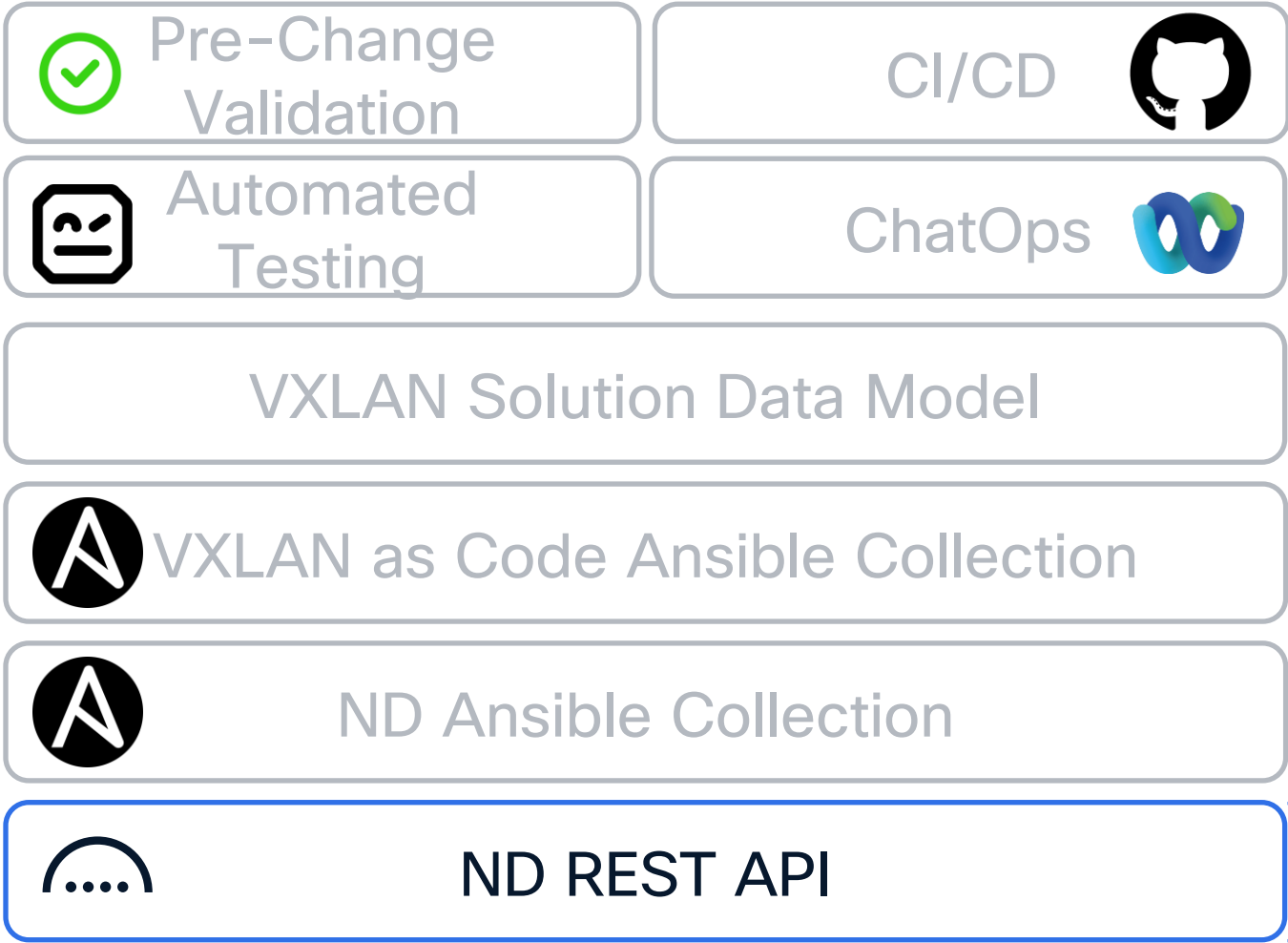
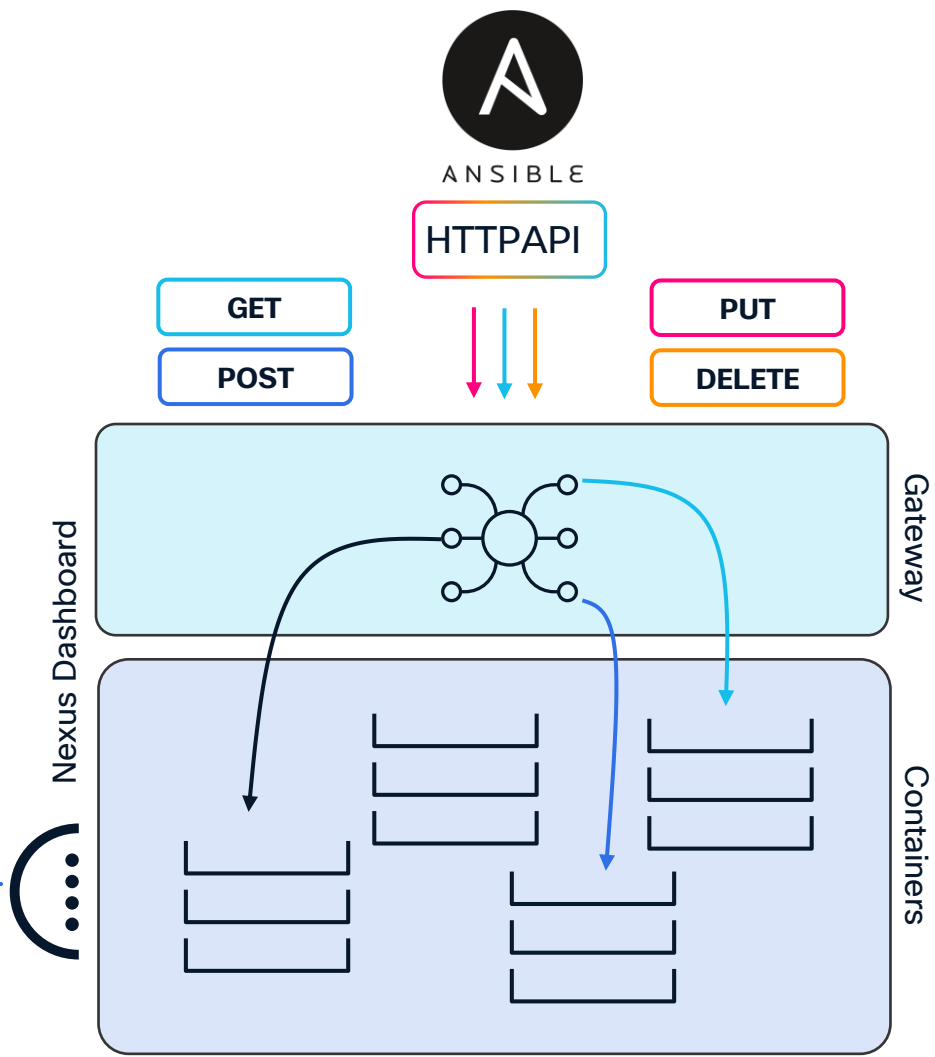
(VXLAN as Code Collection) - https://galaxy.ansible.com/cisco/nac_dc_vxlan

VXLAN as Code - Building Blocks



(Base ND Collection) - <https://galaxy.ansible.com/cisco/dcnm>

VXLAN as Code - Building Blocks



VXLAN as Code - Building Blocks



Pre-Change
Validation

CI/CD



Automated
Testing

ChatOps



VXLAN Solution Data Model



VXLAN as Code Ansible Collection



ND Ansible Collection



ND REST API

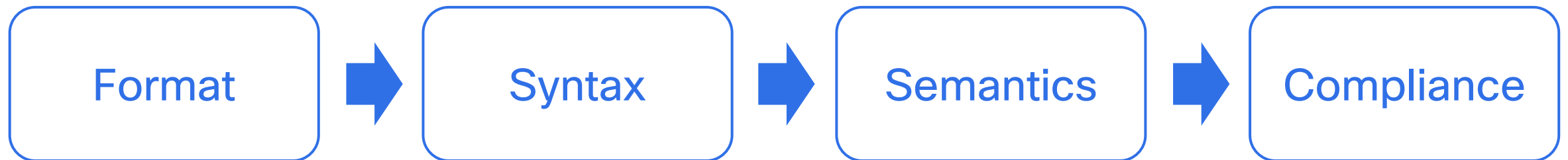
Automation Objectives

- Input and Rules Validation
- Pre and Post Change Testing
- Pipeline Integration
- ChatOps Integration

VXLAN as Code

Pre-Change Validation

- Automated validation is essential as configurations become increasingly complex.
- Validation tasks ensure data integrity and enforce best practices.



VXLAN as Code

Pre-Change Validation

Invalid anycast_gateway_mac

Early detection before sending any requests to the fabric

```
---
vxlan:
  global:
    name: nd_fabric1
    fabric_type: VXLAN_EVPN
    bgp_asn: 65001
    route_reflectors: 2
    anycast_gateway_mac: 12:34:56:78:90:00E99E
```

VXLAN as Code

Pre-Change Validation

Invalid anycast_gateway_mac

Early detection before sending any requests to the fabric

```
---
vxlan:
  global:
    name: nd_fabric1
    fabric_type: VXLAN_EVPN
    bgp_asn: 65001
    route_reflectors: 2
    anycast_gateway_mac: 12:34:56:78:90:00E99E
```

"msg":

"Syntax error '/home/cisco/Documents/nac-vxlan/host_vars/nac-ndfc1/global.nac.yaml':
vxlan.global.anycast_gateway_mac: '12:34:56:78:90:00E99E' is not a mac match"

VXLAN as Code

Pre-Change Validation

Network referencing an undefined VRF

Early detection before sending any requests to the fabric

```
---  
vxlan:  
  overlay:  
    vrfs:  
      # VRF Commented out  
      # - name: NaC-VRF01
```

```
---  
vxlan:  
  overlay:  
    networks:  
      - name: NaC-Net01  
        is_l2_only: False  
        vrf_name: NaC-VRF01
```

VXLAN as Code

Pre-Change Validation

Network referencing an undefined VRF

Early detection before sending any requests to the fabric

```
---
vxlan:
  overlay:
    vrfs:
      # VRF Commented out
      # - name: NaC-VRF01
```

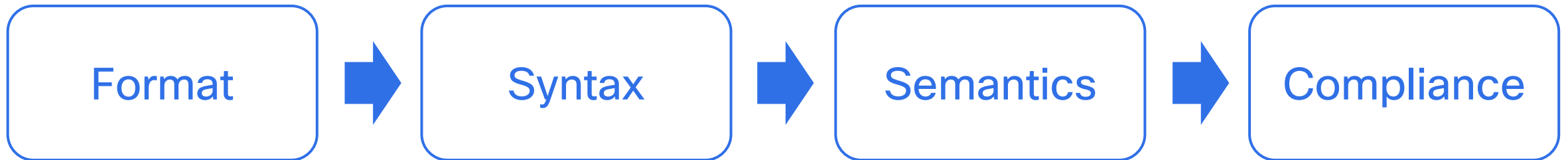
```
---
vxlan:
  overlay:
    networks:
      - name: NaC-Net01
        is_l2_only: False
        vrf_name: NaC-VRF01
```

"msg": "Semantic error, rule 401: Cross Reference VRFs and Networks items in the Service Model (['Network (NaC-Net01) is referencing VRF (NaC-VRF01) which is not defined in the service model. Add the VRF to the service model or remove the network from the service model and re-run the playbook.]"

VXLAN as Code

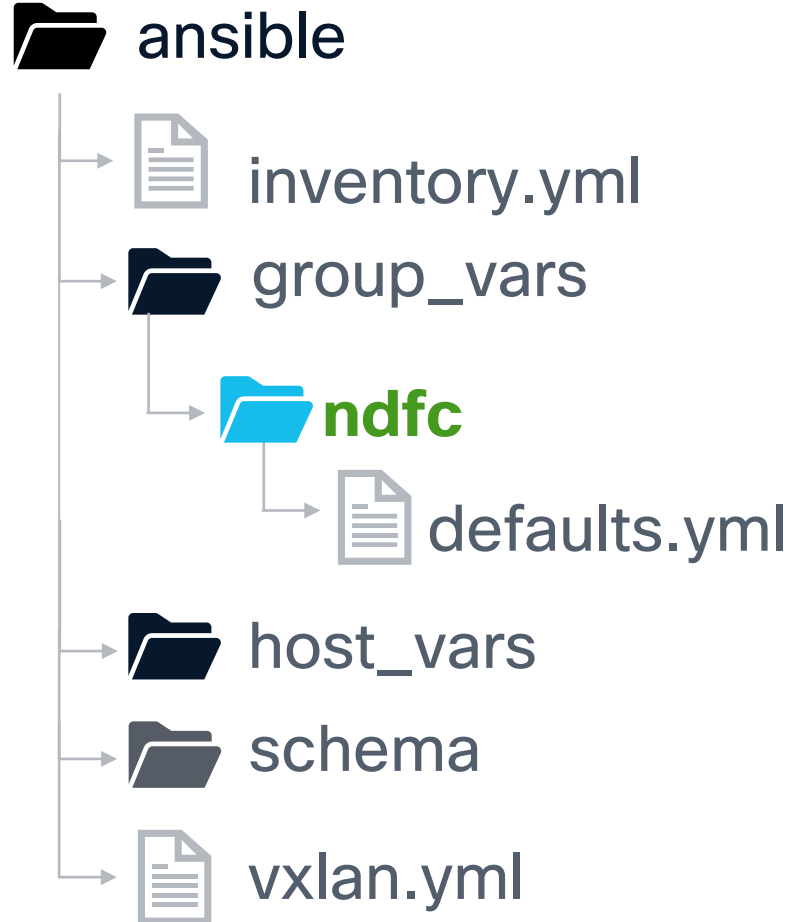
Pre-Change Validation

- Automated validation is essential as configurations become increasingly complex.
- Validation tasks ensure data integrity and enforce best practices.
- When required data is missing, default values are automatically applied.



VXLAN as Code

Data Module Defaults



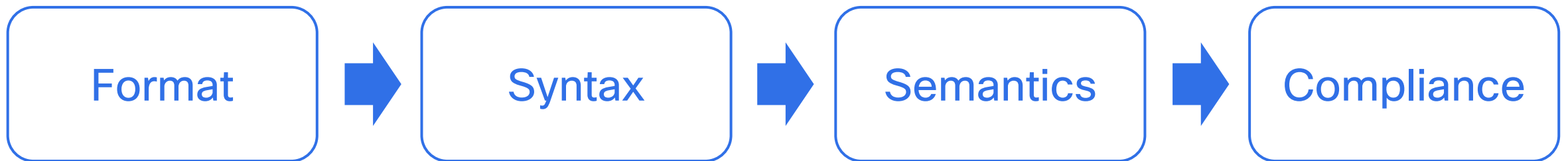
```
defaults:
  vxlan:
    overlay:
      vrfs:
        max_bgp_paths: 1
        max_ibgp_paths: 2
        vrf_int_mtu: 9216
```

```
vxlan:
  overlay:
    vrfs:
      - name: CiscoLive
        ...
      - name: BlueNet
        ...
```

VXLAN as Code

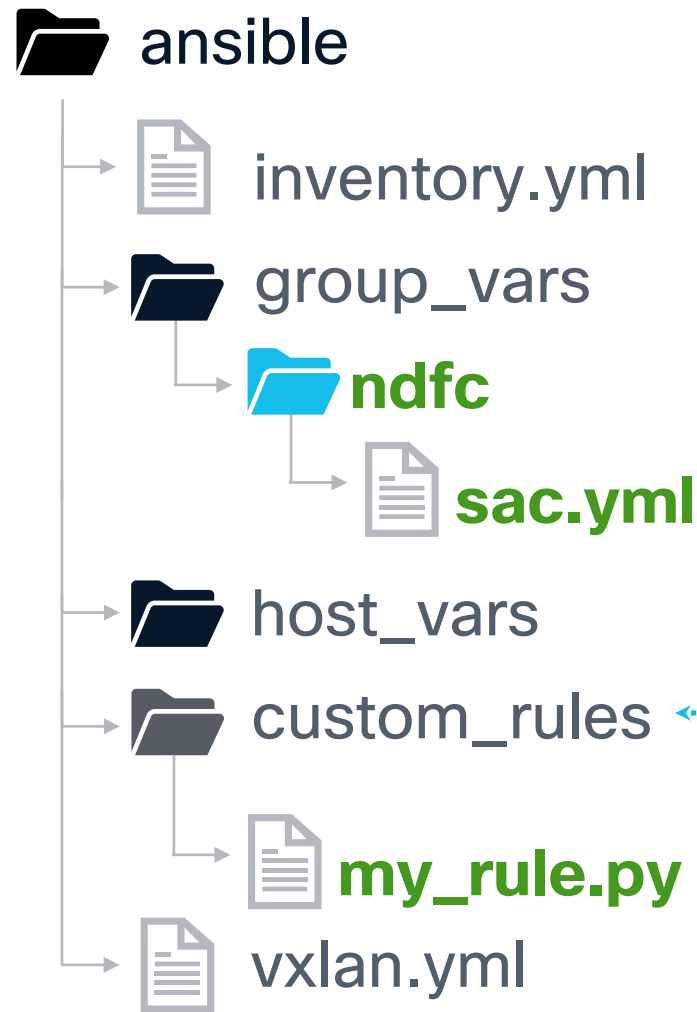
Pre-Change Validation

- Automated validation is essential as configurations become increasingly complex.
- Validation tasks ensure data integrity and enforce best practices.
- When required data is missing, default values are automatically applied.
- Users can add custom validation rules by providing Python scripts.



VXLAN as Code

Data Module Custom Rules



```
import os
import ipaddress
import pynetbox

class Rule:
    id = "546"
    description = "Validate that all routed subnets exist in NetBox"
    severity = "HIGH"
    ...
```

`enhanced_rules_path: "custom_rules"`

Ansible Inventory

Where to do it

 ansible

 inventory.yml

 group_vars

 host_vars

 schema

 vxlan.yml

main inventory file

all:
 children:

 ndfc:

 hosts:

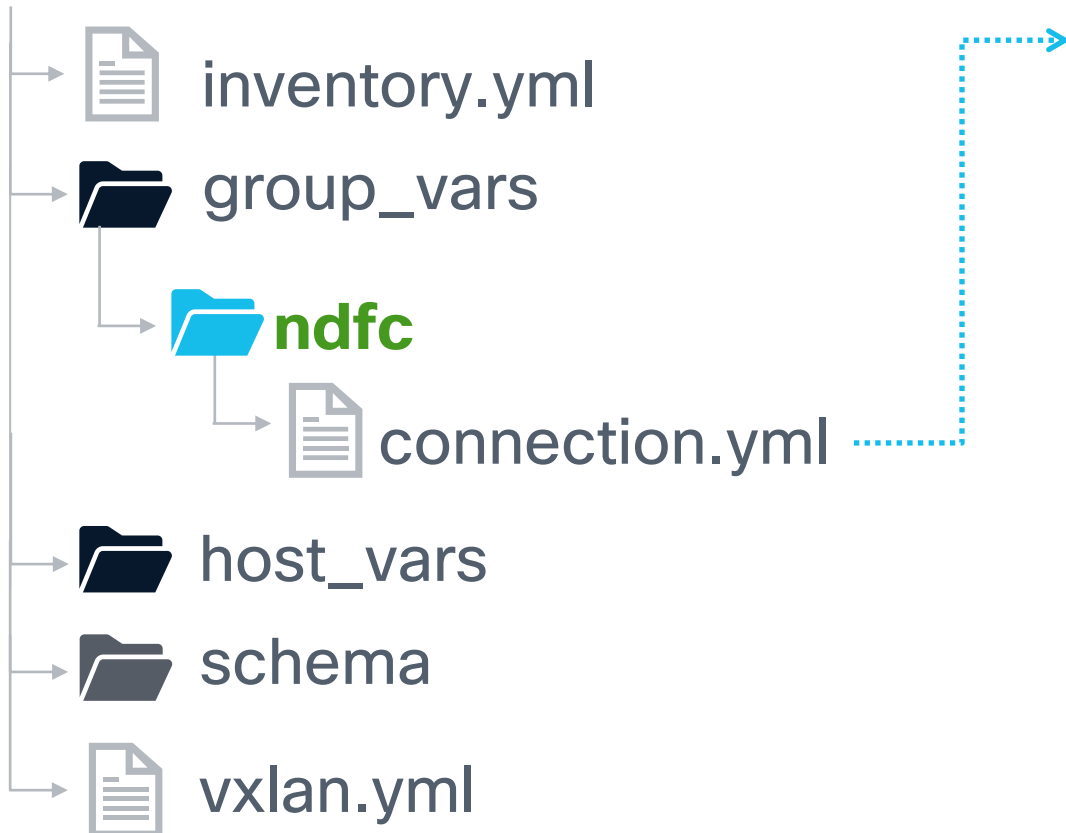
 nac-fabric1:
 ansible_host 10.15.0.20
 nac-fabric2:
 ansible_host 10.15.0.20



Ansible Group Vars

Where to do it

ansible

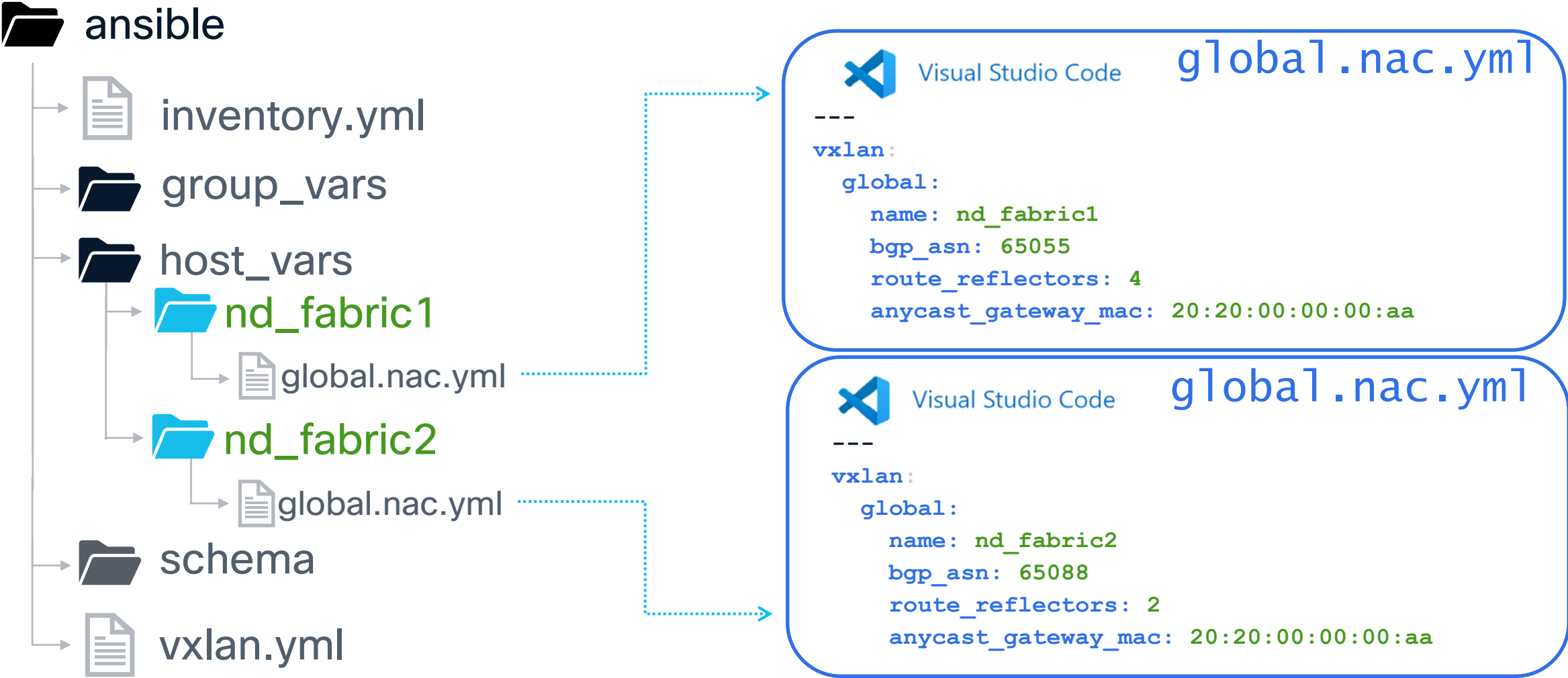


```
---  
# group_vars ndfc connection information  
  
ansible_connection: ansible.netcommon.httpapi  
ansible_user: fc_username  
ansible_password: !vault | (fc_password)  
ansible_network_os: cisco.dcnm.dcnm  
  
ansible_user: "{{ lookup('env', 'ND_USERNAME') }}"  
ansible_password: "{{ lookup('env', 'ND_PASSWORD') }}"  
  
nd_switch_username: "{{ lookup('env', 'ND_SW_USERNAME') }}"  
nd_switch_password: "{{ lookup('env', 'ND_SW_PASSWORD') }}"
```

Connection information for ND

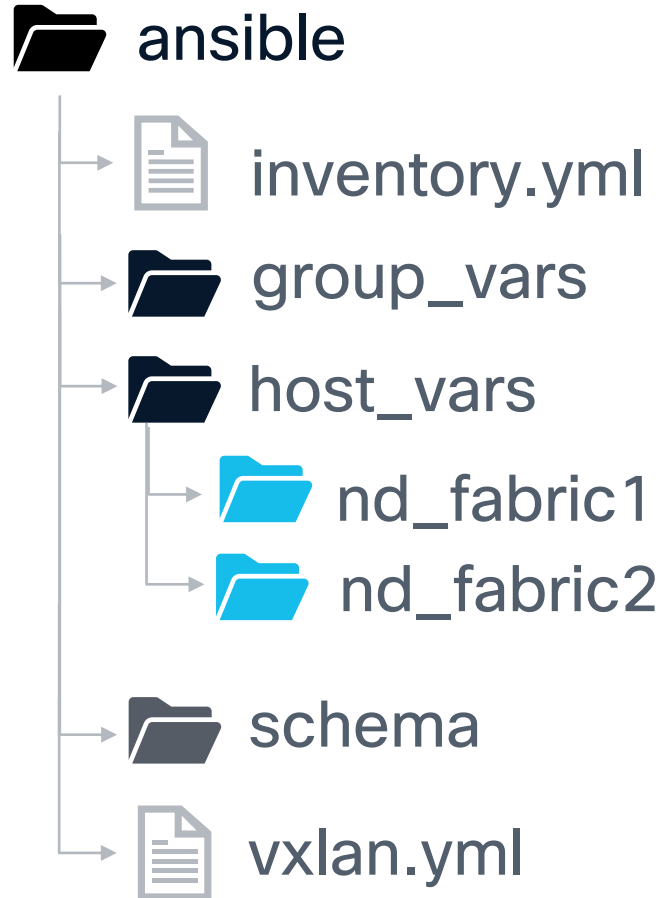
Ansible Host Vars

Data to do It



Ansible Host Vars

Data to do it



Visual Studio Code

global.nac.yml

```
---
vxlan:
  global:
    name: nd_fabric1
    bgp_asn: 65055
    route_reflectors: 4
    anycast_gateway_mac: 20:20:00:00:00:aa
```

Service Model Data

https://netascode.cisco.com/docs/data_models/vxlan/overview/

- global.nac.yml
- underlay.nac.yml
- interfaces.nac.yml
- overlay.nac.yml
- topology.nac.yml
- vrf.nac.yml
- networks.nac.yml
- policy.nac.yml

Ansible Playbook

What to Do

ansible



```
---  
# main NaC playbook  
- name: Build VXLAN EVPN Fabric on ND  
  hosts: nd_fabric1, nd_fabric2  
  gather_facts: false
```

```
roles:
```

```
#- role: cisco.nac_dc_vxlan.validate
```

```
- role: cisco.nac_dc_vxlan.dtc.create  
  tags: 'role_create'
```

```
- role: cisco.nac_dc_vxlan.dtc.deploy  
  tags: 'role_deploy'
```

```
- role: cisco.nac_dc_vxlan.dtc.remove  
  tags: 'role_remove'
```

*roles will be installed in the collection directory. Here are defined just for a visual reference

Workflow Step1: Fabric Management

Clicking Around

Nexus Dashboard

ND-Cluster3

Home

Manage

Analyze

Admin

← Fabrics

Create/Onboard Fabric

✓ Select a category
Create new LAN fabric

✓ Select a type
VXLAN

3 Settings
Default

4 Summary

5 Fabric creation

Settings

These are the recommended settings for configuring the parameters and capabilities of the new fabric.

Configuration mode ⓘ
☒ Default ☐ Advanced

Name *

BRKDCN-2946

Location *

San Jose, US

BGP ASN *

65419

1-4294967295 | 1-65535[.0-65535]

License tier for fabric ⓘ
☒ Essentials ☐ Advantage ☐ Premier

Enabled features
☐ Telemetry ⓘ

Cancel

© 2025 Cisco and/or its affiliates. All rights reserved.

BRKDCN-2946

Model – Fabric Management

Data to do it

How to do it

ansible

host_vars

nd_fabric1

roles

validate

create

deploy

remove

vxlan.yml



Visual Studio Code

global.nac.yaml

vxlan:

global:

name: BRKDCN-2946

bgp_asn: 65419

route_reflectors: 2

anycast_gateway_mac: 20:20:00:00:00:aa

validate

create

deploy

remove

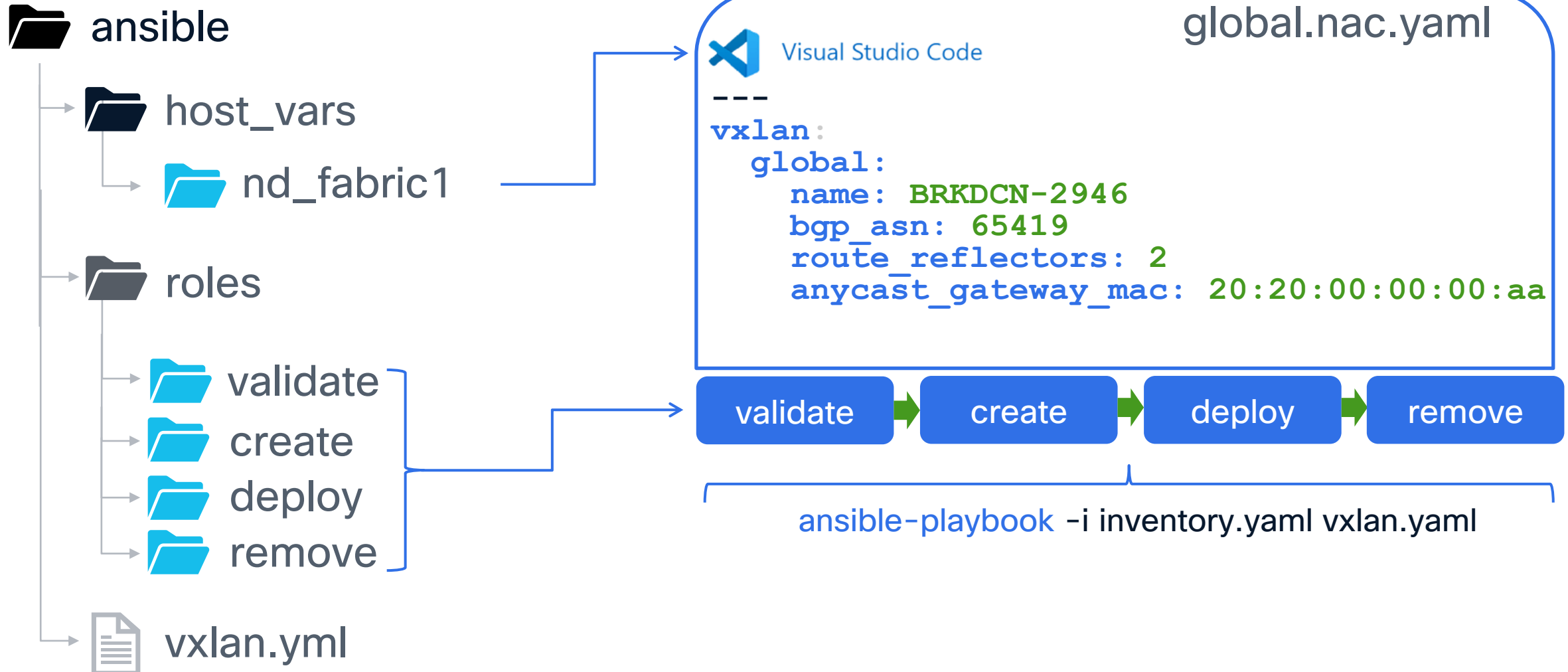
START HERE:

<https://github.com/netascode/ansible-dc-vxlan>

Model – Fabric Management

Data to do it

How to do it



Model – Fabric Management

Data to do it

How to do it

ansible

host_vars

nd_fabric1

roles

validate

create

deploy

remove

vxlan.yml



Visual Studio Code

global.nac.yaml

```
---
vxlan:
  global:
    name: BRKDCN-2946
    bgp_asn: 65419
    route_reflectors: 2
    anycast_gateway_mac: 20:20:00:00:00:aa
```

validate

create

deploy

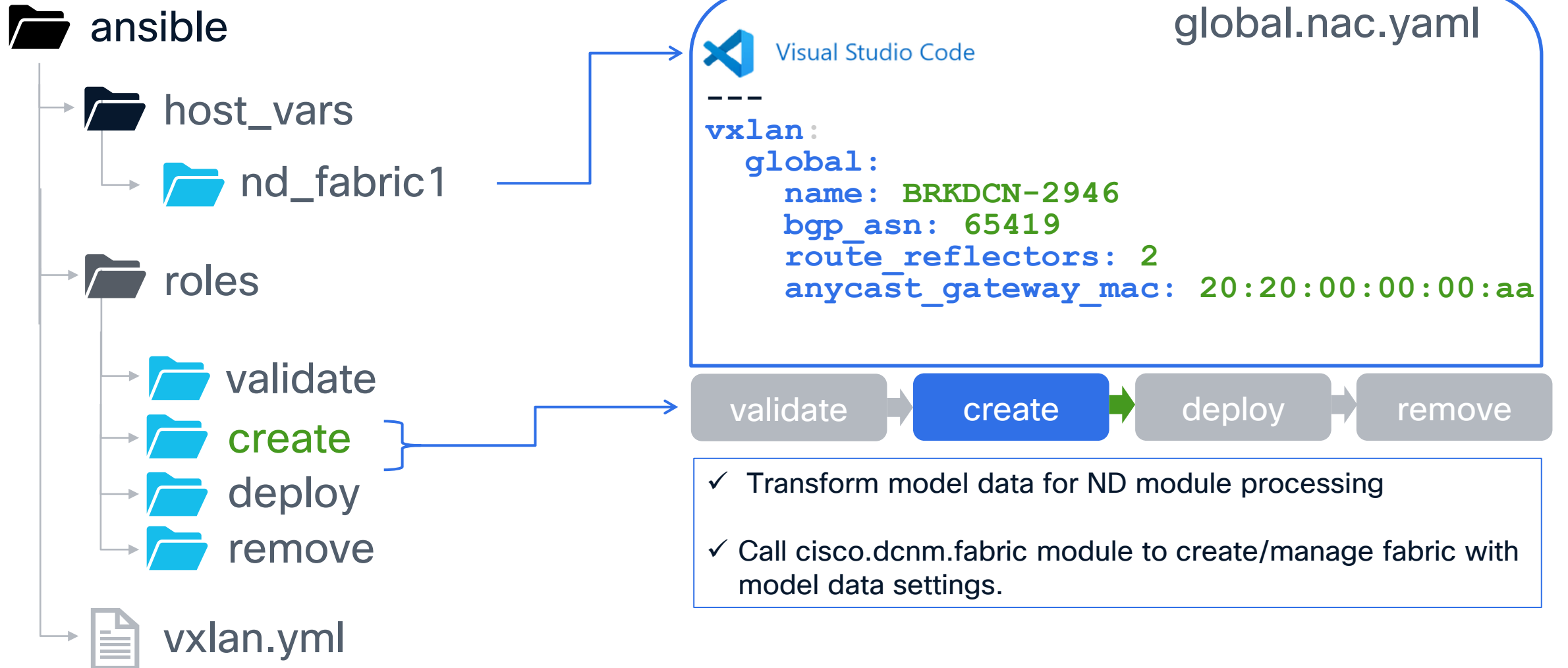
remove

- ✓ Validate vxlan.global data against model schema
 - ✓ Semantic Validation – model structure
 - ✓ Input Validation – anycast_gateway_mac valid?
- ✓ Process base and custom rules
 - ✓ Example: spanning tree vlan and mst ranges

Model – Fabric Management

Data to do it

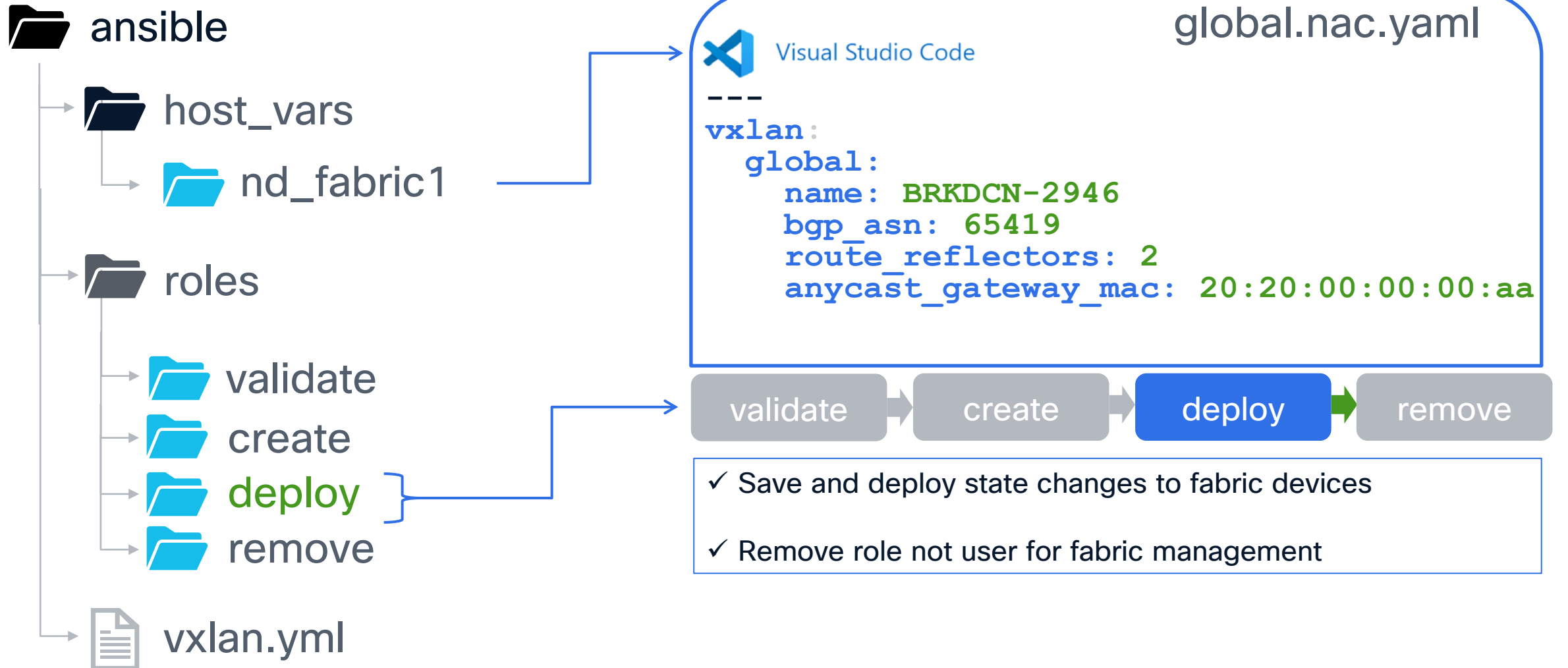
How to do it



Model – Fabric Management

Data to do it

How to do it



Workflow Step2: Inventory Management

Clicking Around

 Nexus Dashboard

ND-Cluster3

Home

Manage

Analyze

Admin

Add switches - Fabric: BRKDCN-2946

Switch Addition Mechanism*
☒ Discover

Seed Switch Details

Seed IP*

172.23.254.25

i

Ex: "2.2.2.20" or "10.10.10.40-60" or "2.2.2.20, 2.2.2.21"

Authentication / Privacy*

MD5

Username*

admin

Max Hops*

2

Preserve Config
☐

Password*

.....

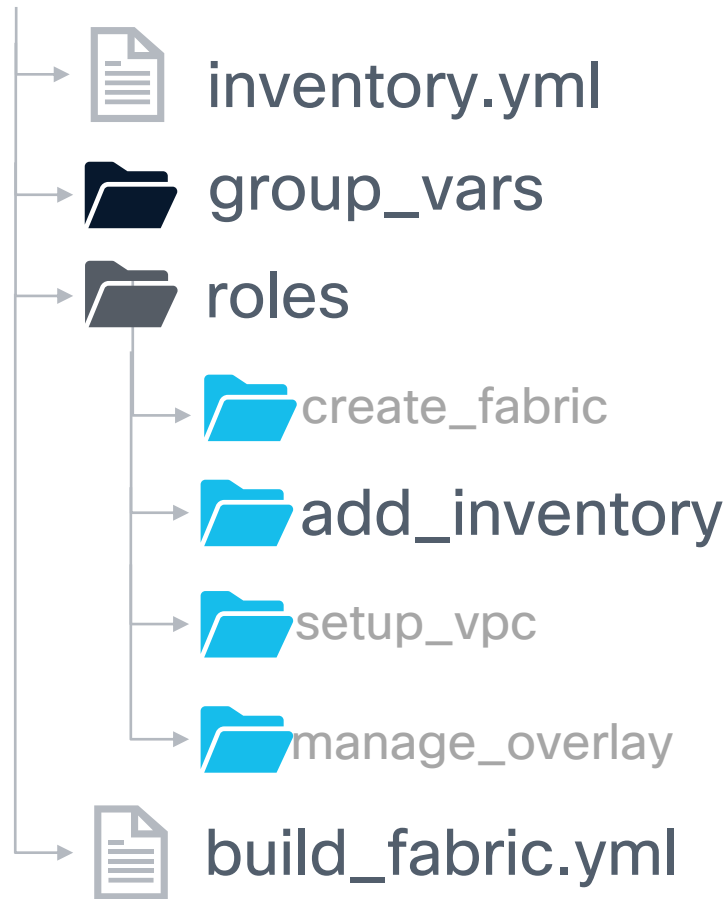
Show

☐ Set as individual device write credential

Ansible Roles – Inventory Tasks

How to do it

ansible



roles/add_inventory/tasks/main.yml

```
---
# tasks file for roles/add_inventory

- name: Add Fabric Devices
  cisco.dcnm.dcnm_inventory:
    fabric: "{{ fabric.name }}"
    state: merged
  config:
    - seed_ip: 192.168.1.1
      role: spine
      < credentials >
        - seed_ip: 192.168.1.2
          role: leaf
          < credentials >
    - seed_ip: 192.168.1.4
      role: border
      < credentials >
    poap:
      - serial_number: 2A3BCDEFJKL
```

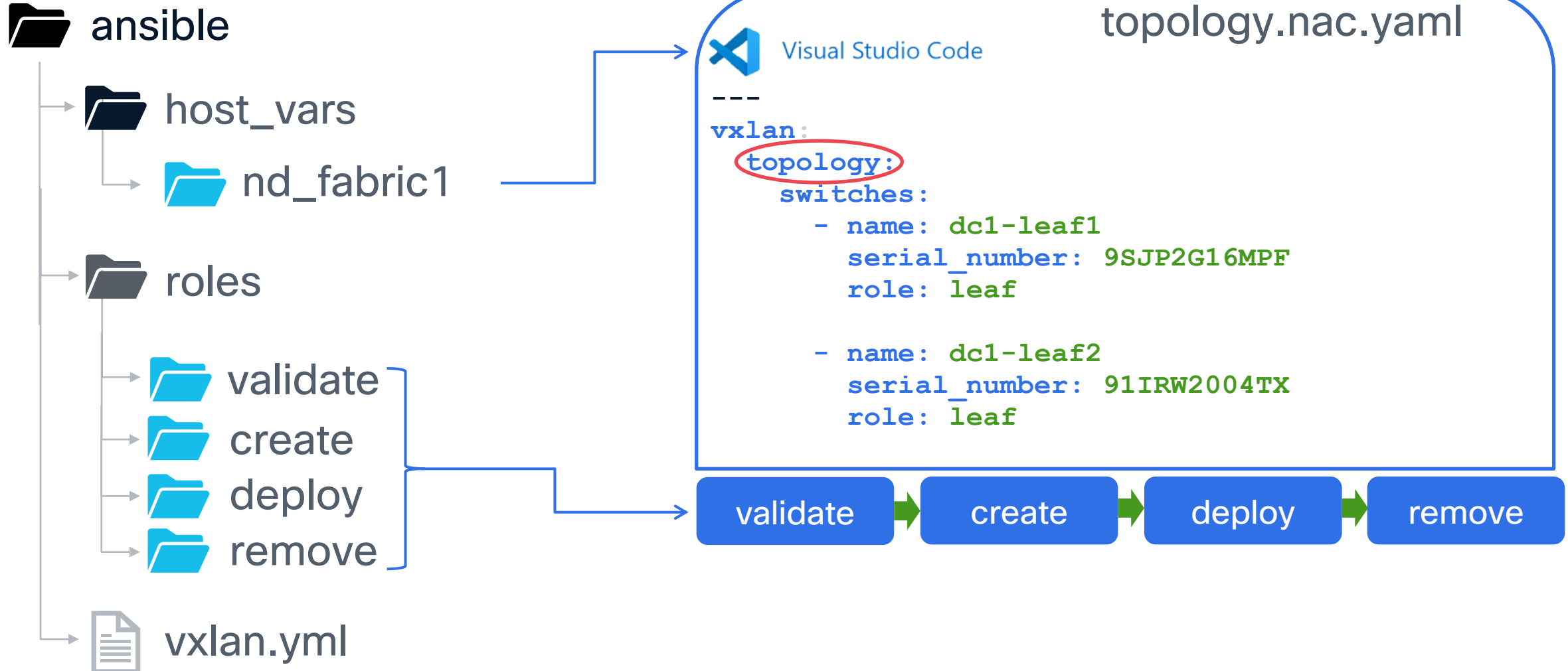
<snip>

- Adds devices to the fabric
- Defines role in the fabric
- Deploy control

Model – Inventory Management

Data to do it

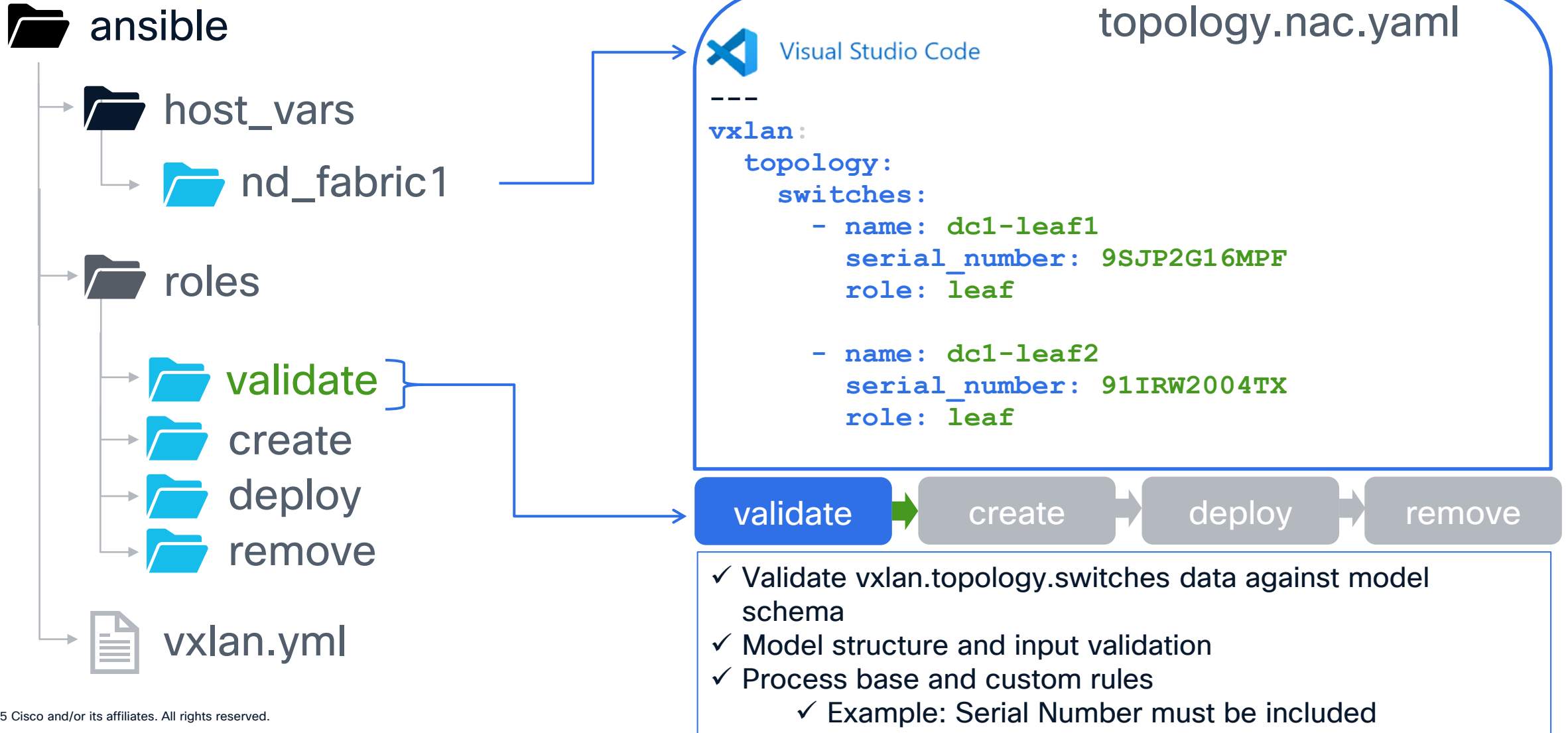
How to do it



Model – Inventory Management

Data to do it

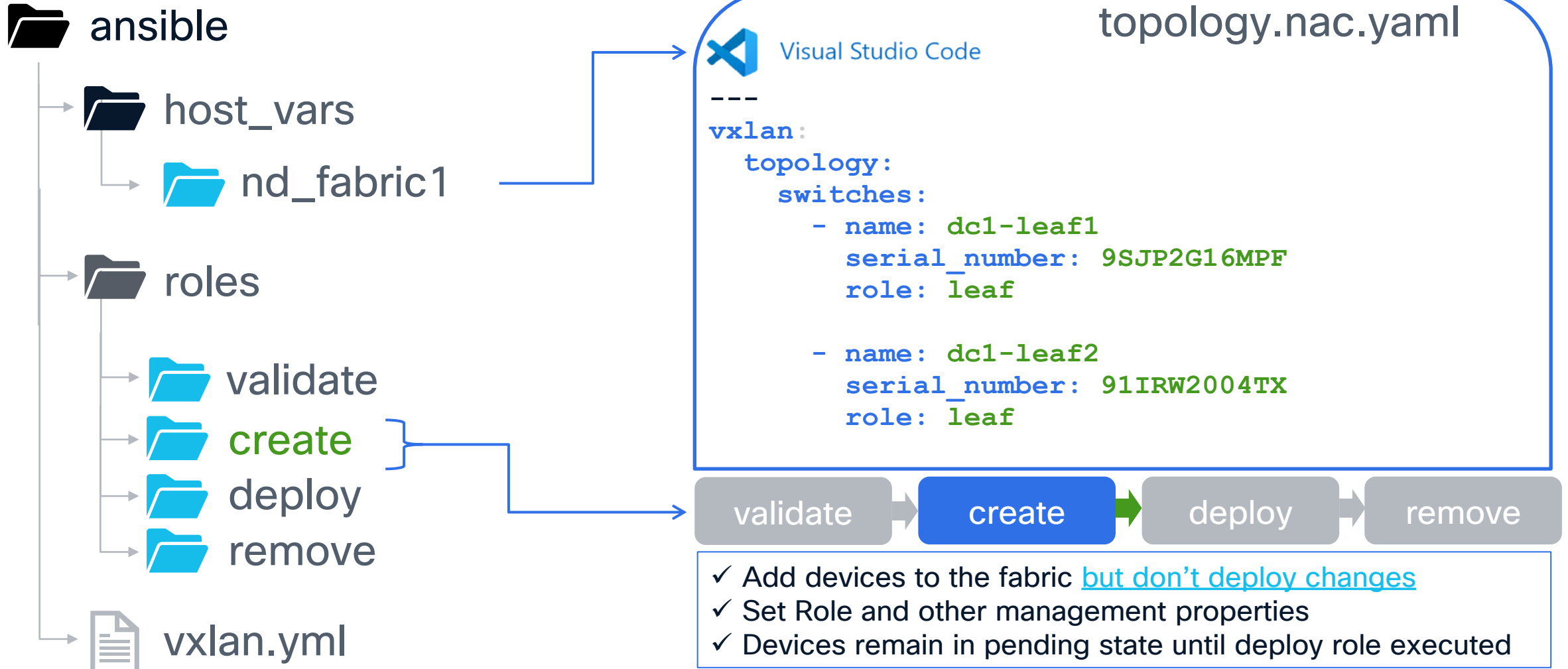
How to do it



Model – Inventory Management

Data to do it

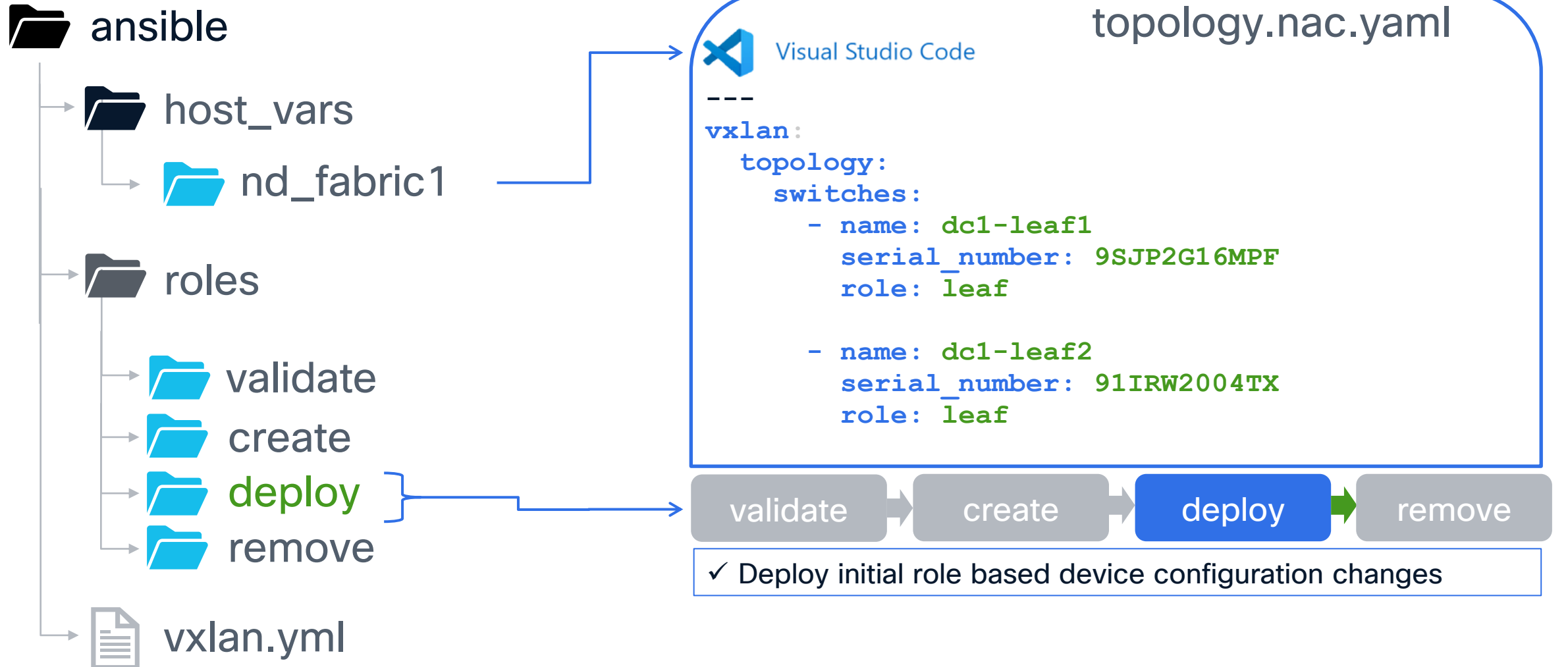
How to do it



Model – Inventory Management

Data to do it

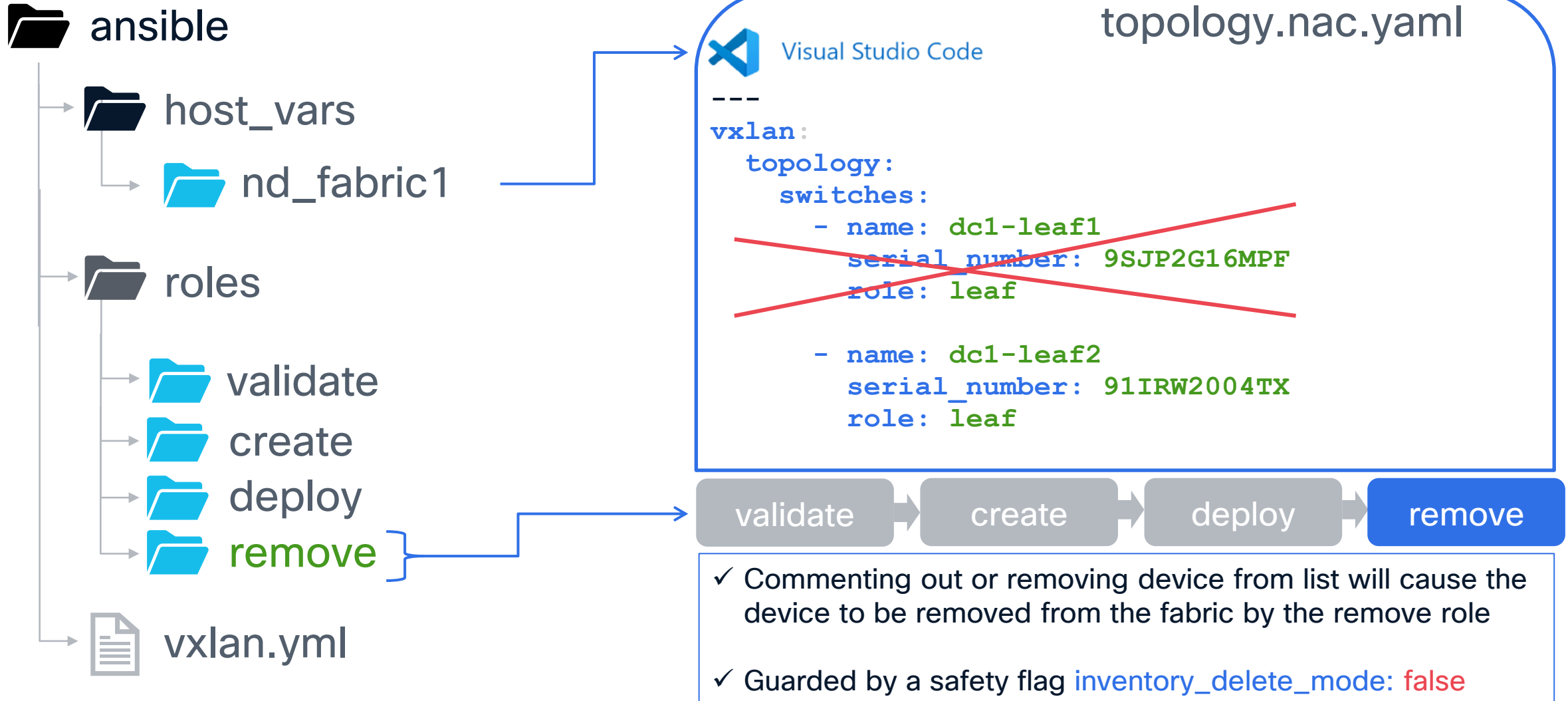
How to do it



Model – Inventory Management

Data to do it

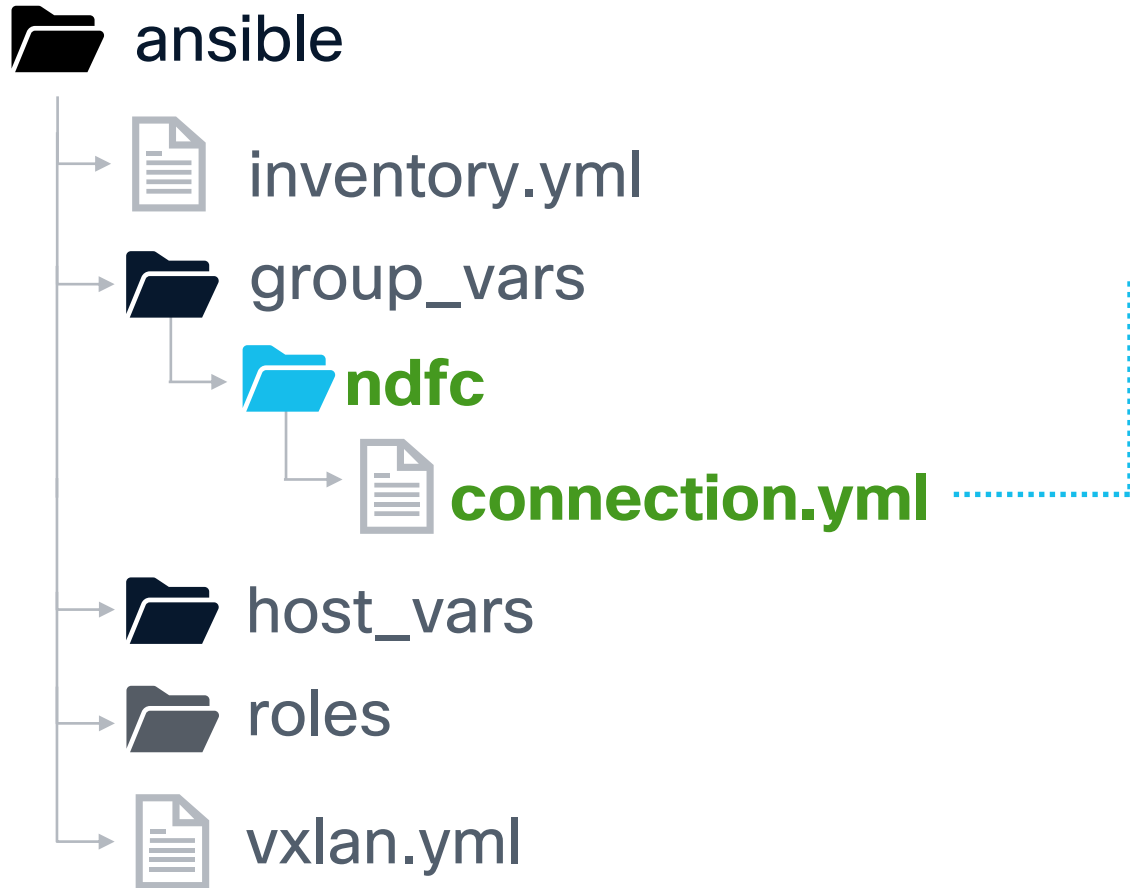
How to do it



Model – Inventory Management

Data to do it

How to do it



```
# Parameter to ignore diff-run
force_run_all: false

# Parameters for the tasks in the 'Remove' role
interface_delete_mode: true
inventory_delete_mode: true
link_vpc_delete_mode: true
network_delete_mode: true
policy_delete_mode: true
vpc_delete_mode: true
vrf_delete_mode: true
```

Flag	Default Value
force_run_all	false
*_delete_mode	false

Workflow Step3: Interface Configuration

Clicking Around

ND-Cluster3

Home

Manage

Analyze

Admin

Nexus Dashboard

Create interface

Type*

virtual Port Channel (vPC)

Select a vPC pair*

leaf-1---leaf-2

vPC ID*

2

Policy*

[int_vpc_trunk_host](#) >

Policy Options

General Parameters

Storm Control

Peer-1 Port-Channel ID*

2

Peer-1 VPC port-channel number (Min:1, Max:4096)

Peer-2 Port-Channel ID*

2

Peer-2 VPC port-channel number (Min:1, Max:4096)

☐ Enable Config Mirroring

If enabled, Peer-1 config will be copied to Peer-2

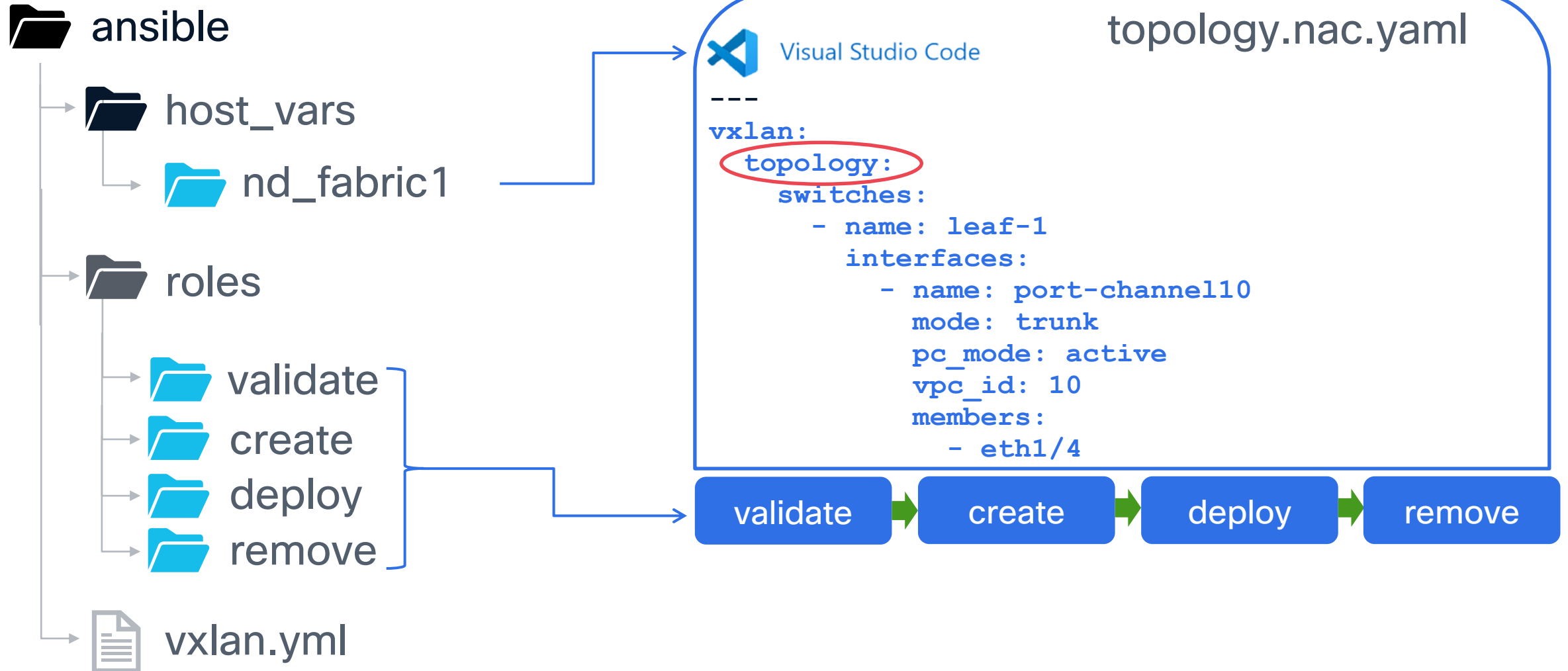
© 2025 Cisco and/or its affiliates. All rights reserved.

BRKDCN-2946

Model – Interface Configuration

Data to do it

How to do it



Workflow Step4: Overlay Management

Clicking Around

Create VRF

!

VRF Name*

CL-VRF1

VRF ID*

470000

VLAN ID

2055

Propose VLAN

VRF Template*

Default_VRF_Universal >

VRF Extension Template*

Default_VRF_Extension_Universal >

General Parameters

Advanced

Route Target

Create Network

!

Network Name*

CL-NET7000

Layer 2 Only

☐

VRF Name*

CL-VRF1

Create VRF

Network ID*

7000

VLAN ID

88

Propose VLAN

VRF Overview - CL-VRF1

Overview

VRF Attachments

Networks

VRF Info

VRF Name

CL-VRF1

VRF ID

470000

VLAN ID

2055

Status

DEPLOYED

Fabric Name

fabric-stage

VRF Template

Default_VRF_Universal

VRF Extension Template

Default_VRF_Extension_Universal

VRF Description

NA

L3VniMcastGroup

NA

VRF Status

3

Status

DEPLOYED 3

Attached Roles Association

3

Role

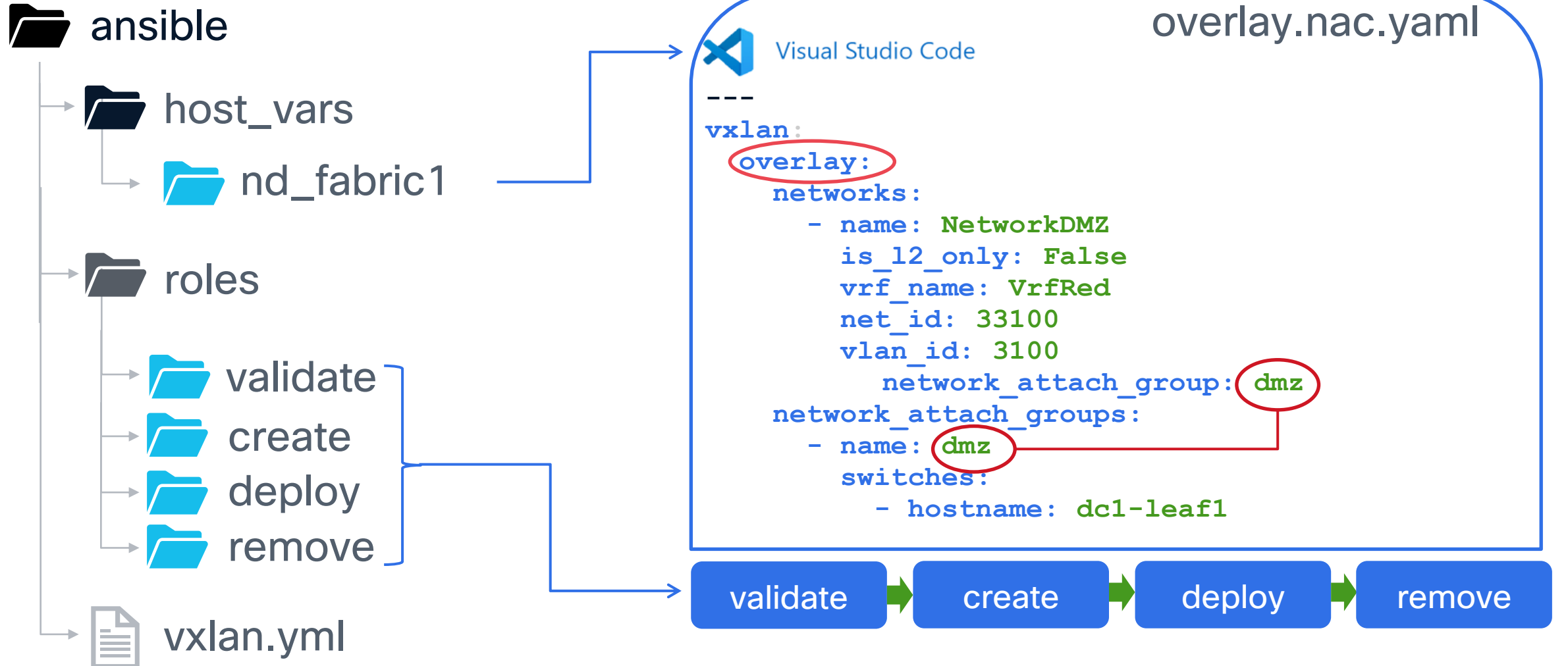
border 1

leaf 2

Model – Overlay Management

Data to do it

How to do it



ND as Code – Ansible Tags

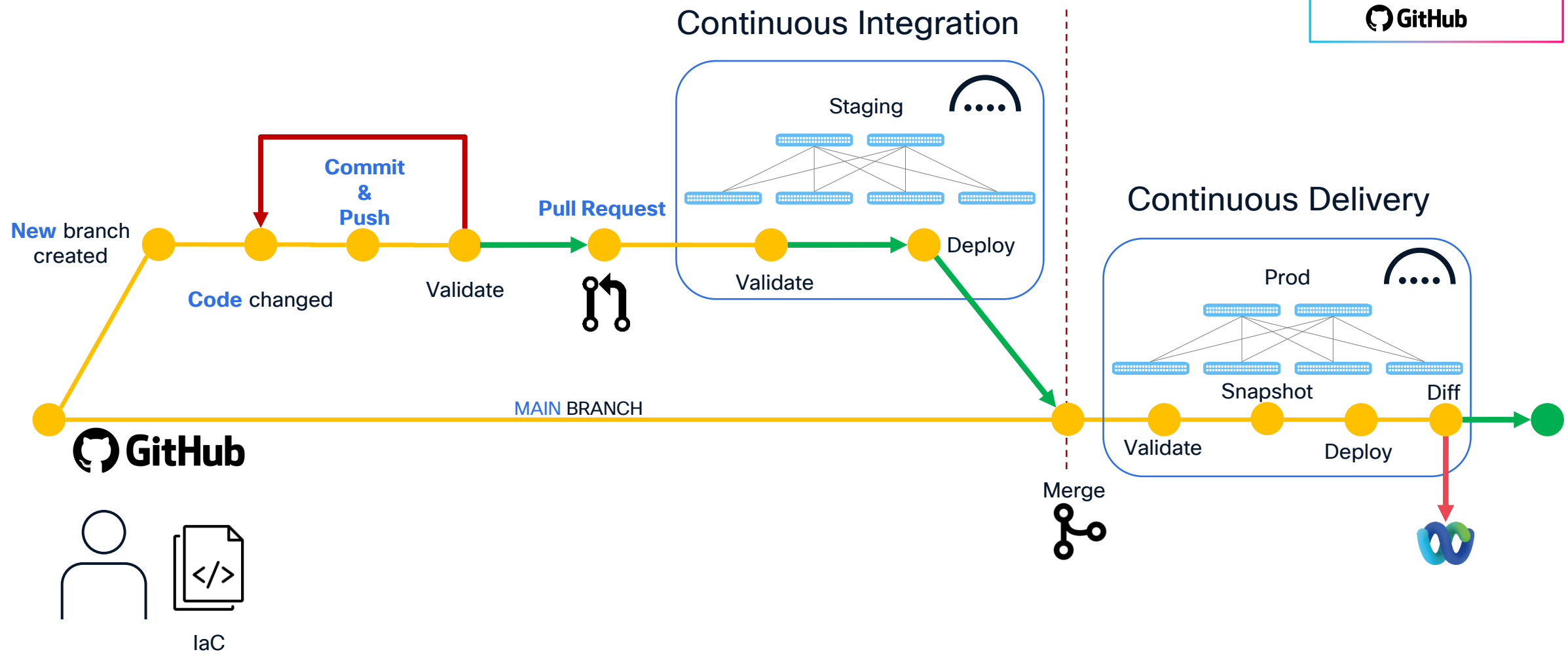
Tags for selective execution control

```
ansible-playbook -i inventory.yaml vxlan.yaml --tags role* -vvvv
```

```
ASK [cisco.nac_dc_vxlan.common_global : Verify User Tags]
*****
*****
task path: /home/cisco/Documents/nac-
vxlan/collections/ansible_collections/cisco/nac_dc_vxlan/roles/common_global/tasks/main.yml:23
Saturday 25 January 2025  20:16:41 +0000 (0:00:01.694)          0:00:01.694 *****
Saturday 25 January 2025  20:16:41 +0000 (0:00:01.693)          0:00:01.693 *****
fatal: [nac-ndfc1 -> localhost]: FAILED! => {
  "changed": false,
  "msg": "Tag 'role*' not found in list of supported tags",
  "supported_tags": [
    "cc_verify",
    "cr_manage_fabric",
    "cr_manage_switches",
    "cr_manage_vpc_peers",
    "cr_manage_interfaces",
    "cr_manage_vrfs_networks",
    "cr_manage_policy",
    "cr_manage_links", >>>
  ]
}
(More tags supported like role_verify, role_create, role_deploy, role_remove)
```

CI/CD Pipeline Demo

Infrastructure as Code for Network Ops



References to Start Your Journey

Ansible VXLAN as Code Repos



VXLAN NaC Collection
Repo

<https://github.com/netascode/ansible-dc-vxlan>



VXLAN NaC Collection
Example Repo

<https://github.com/netascode/ansible-dc-vxlan-example>

Ansible for NXOS and ND Repos



BRKDCN-2946
Session Repo

<https://github.com/mtarking/BRKDCN-2946>



BRKDCN-2946
CI/CD Demo

<https://github.com/alessandro-deprato/BRKDCN-2946/>

Complete your session evaluations



Complete a minimum of 4 session surveys and the Overall Event Survey to claim a Cisco Live T-Shirt.



Earn up to 800 points by completing all surveys and climb the Cisco Live Challenge leaderboard.



Level up and earn exclusive prizes!



Complete your surveys in the Cisco Live Events app.

Continue your education



Visit the Cisco Stand for related demos



Book your one-on-one Meet the Expert meeting



Attend the interactive education with Capture the Flag, and Walk-in Labs



Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Thank you

CISCO Live !

