

Infrastructure as Code for NXOS and NDFC with Ansible

CISCO Live !

Matt Tarkington
Technical Leader

Mike Wiebe
Technical Leader

Cisco Webex App

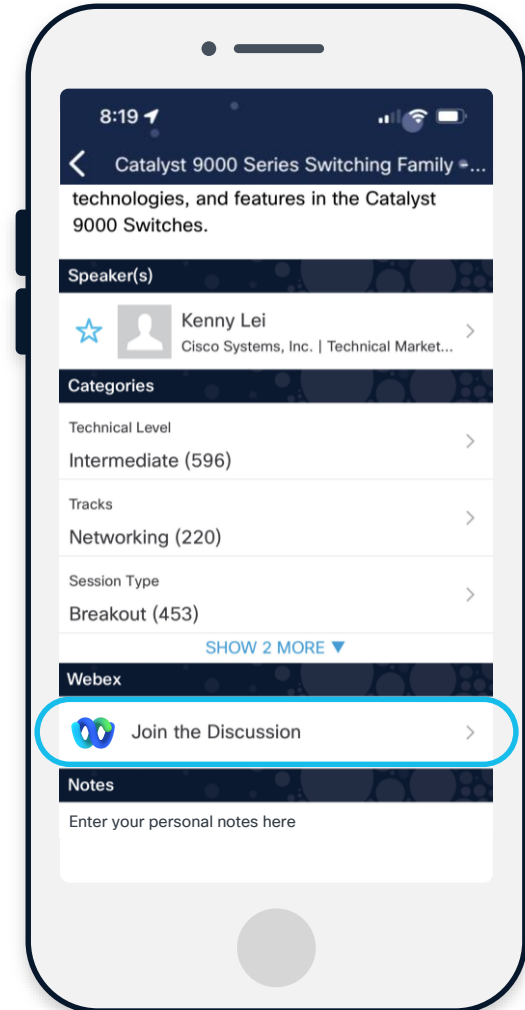
Questions?

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 June 13, 2025.



Agenda

- 01 What is Infrastructure as Code?
- 02 Infrastructure as Code with NXOS and Ansible
- 03 Infrastructure as Code with NDFC and Ansible
- 04 Start your IaC Journey!

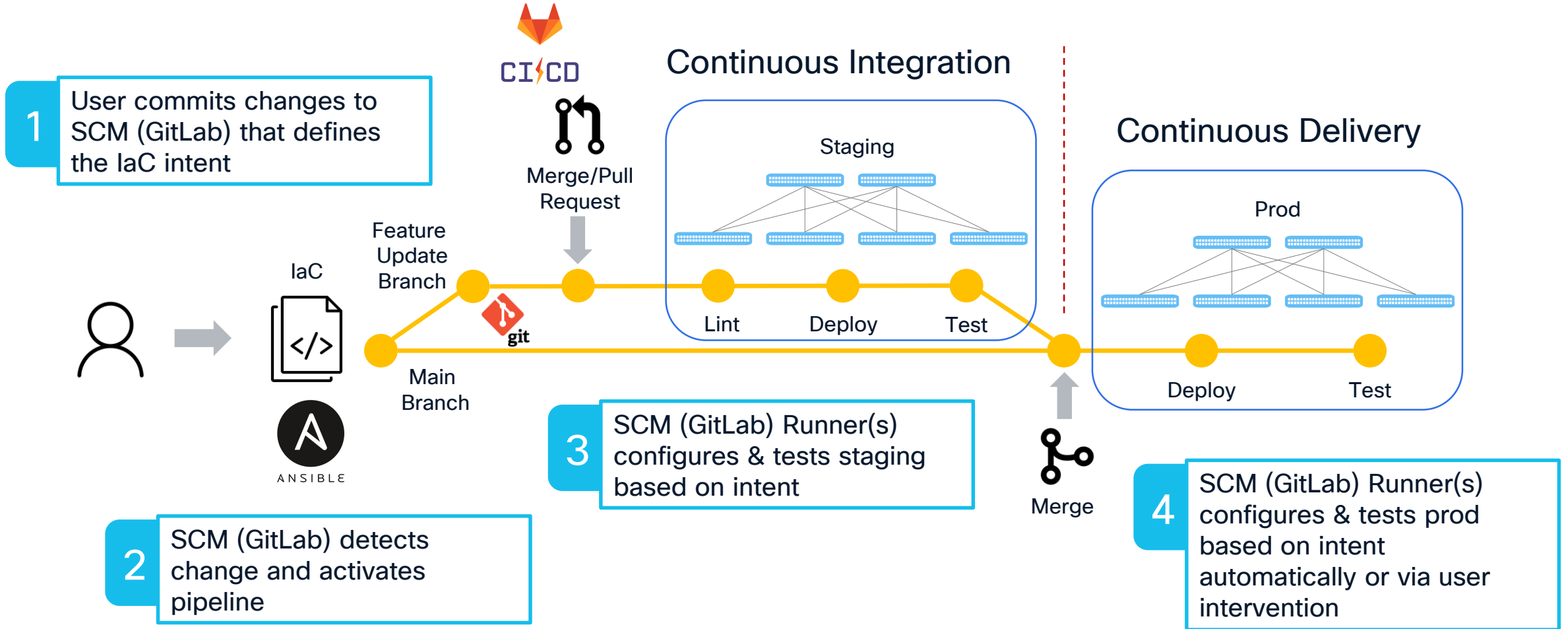
What is Infrastructure as Code?

Infrastructure as Code for Network Ops

- 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



Infrastructure as Code for Network Ops



Infrastructure as Code with NXOS and Ansible

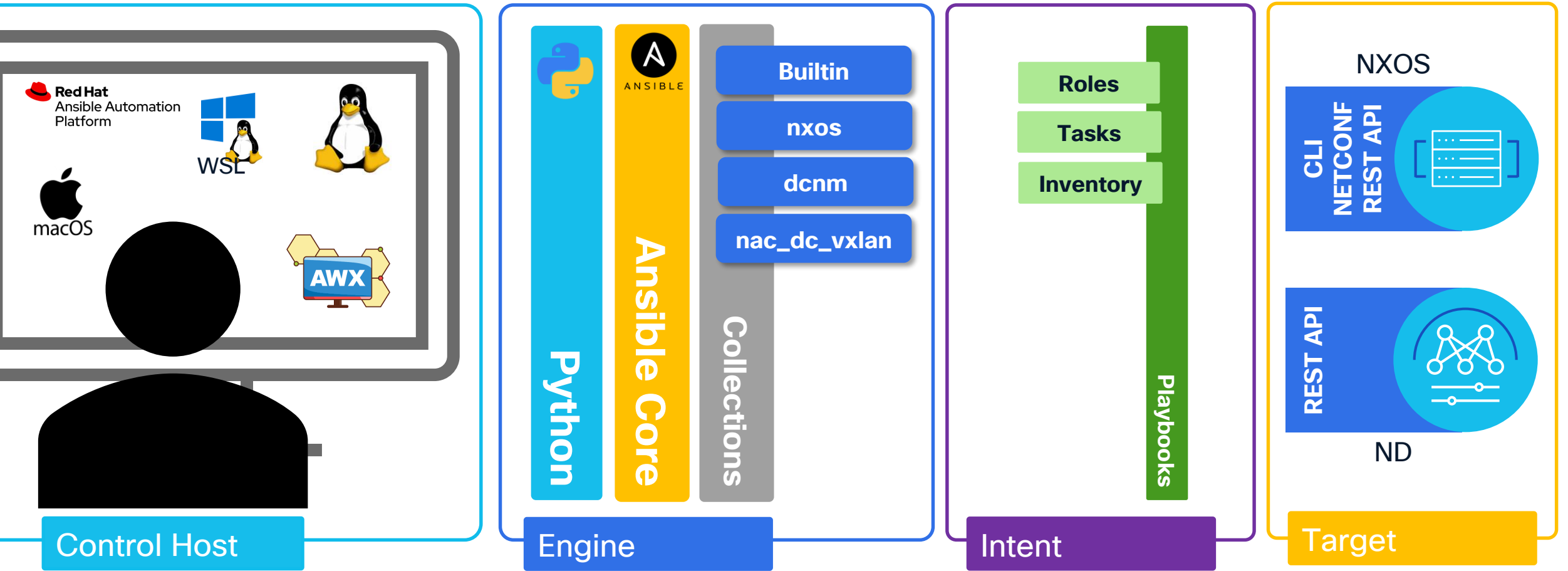
What is Ansible?



- Automation / Configuration / Orchestration tool
- Open Source
- Agentless Push Model
- Produces the same results no matter how many times it is executed*
- No programming knowledge required
- Requires only data-structure manipulation knowledge
- Network CLI and REST API interaction

*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
- Uses Ansible Galaxy as the delivery vehicle
- Contains modules, plugins, filters
- Collections not related to Ansible release schedules
 - Allows vendor flexibility in relation to product releases

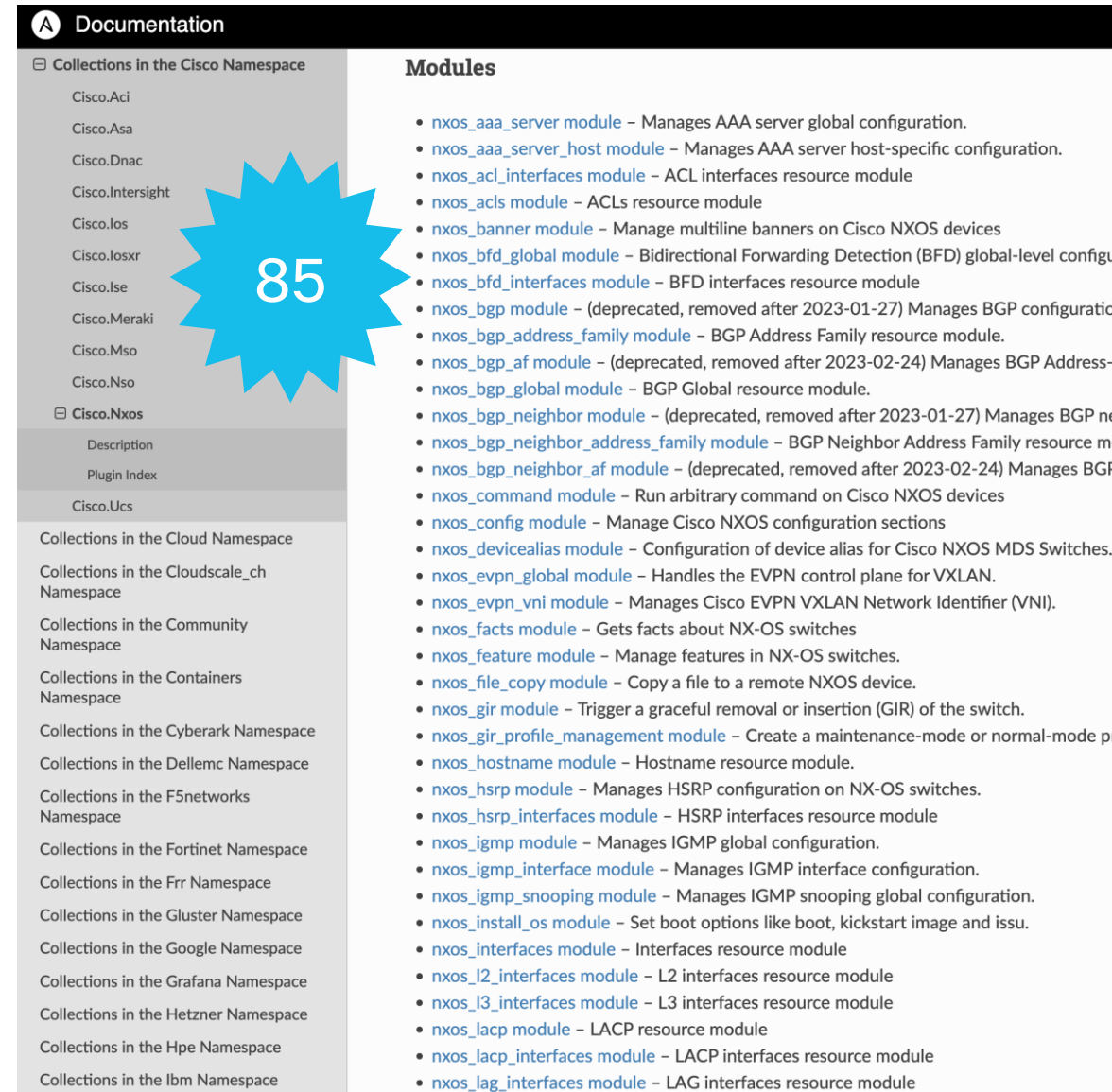
```
% ansible-galaxy collection install cisco.(nxos|dcnm|nac_dc_vxlan)
```

(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



Documentation

Collections in the Cisco Namespace

- Cisco.Aci
- Cisco.Asa
- Cisco.Dnac
- Cisco.Intersight
- Cisco.Ios
- Cisco.Iosxr
- Cisco.Ise
- Cisco.Meraki
- Cisco.Mso
- Cisco.Nso
- Cisco.Nxos**

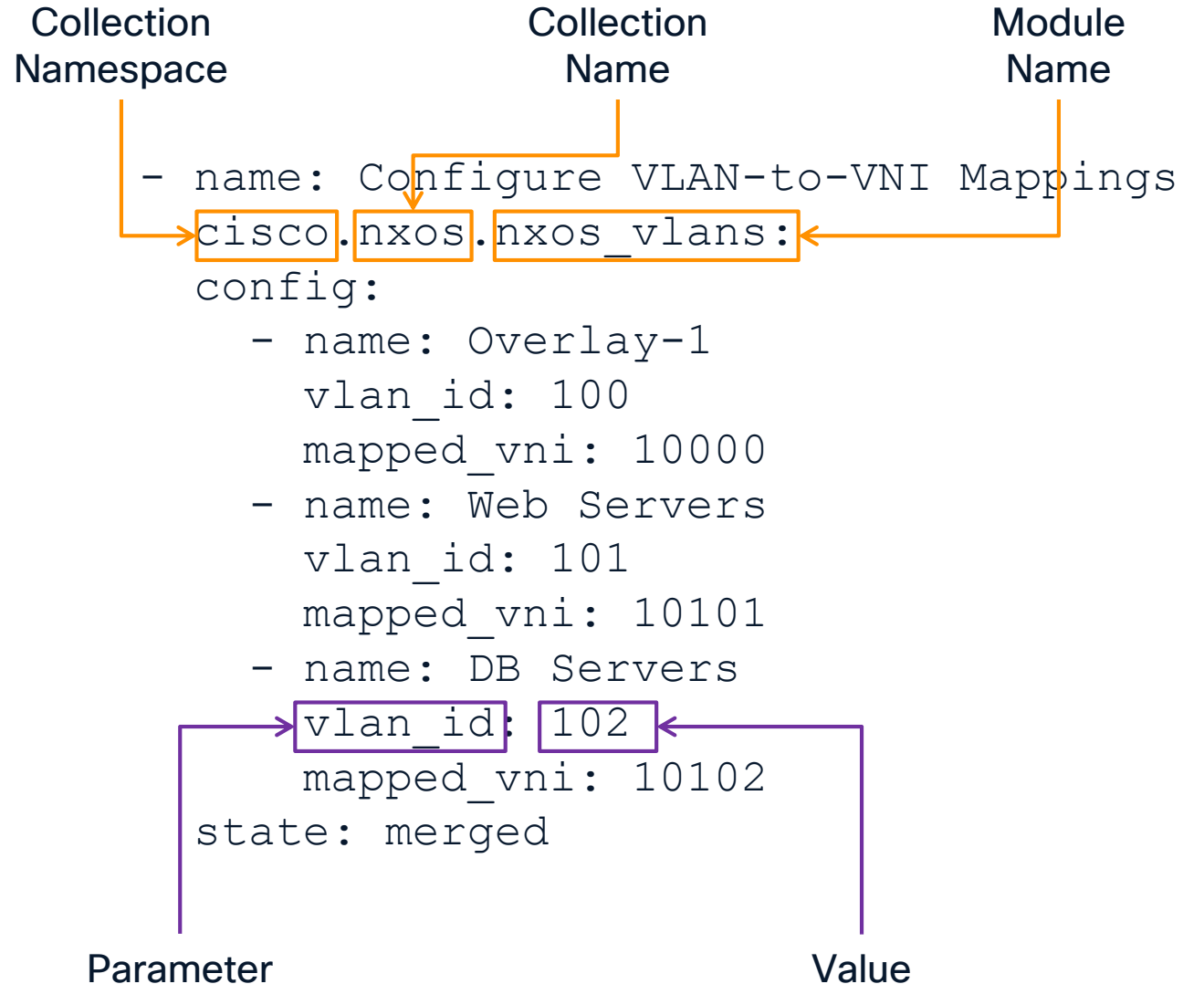
Modules

- [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 config
- [nxos_bfd_interfaces module](#) – BFD interfaces resource module
- [nxos_bgp module](#) – (deprecated, removed after 2023-01-27) Manages BGP configuration
- [nxos_bgp_address_family module](#) – BGP Address Family resource module.
- [nxos_bgp_af module](#) – (deprecated, removed after 2023-02-24) Manages BGP Address-
- [nxos_bgp_global module](#) – BGP Global resource module.
- [nxos_bgp_neighbor module](#) – (deprecated, removed after 2023-01-27) Manages BGP ne
- [nxos_bgp_neighbor_address_family module](#) – BGP Neighbor Address Family resource m
- [nxos_bgp_neighbor_af module](#) – (deprecated, removed after 2023-02-24) Manages BGP
- [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_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 p
- [nxos_hostname module](#) – Hostname resource module.
- [nxos_hsrp module](#) – Manages HSRP configuration on NX-OS switches.
- [nxos_hsrp_interfaces module](#) – HSRP interfaces resource module
- [nxos_igmp module](#) – Manages IGMP global configuration.
- [nxos_igmp_interface module](#) – Manages IGMP interface configuration.
- [nxos_igmp_snooping module](#) – Manages IGMP snooping global configuration.
- [nxos_install_os module](#) – Set boot options like boot, kickstart image and issu.
- [nxos_interfaces module](#) – Interfaces resource module
- [nxos_l2_interfaces module](#) – L2 interfaces resource module
- [nxos_l3_interfaces module](#) – L3 interfaces resource module
- [nxos_lacp module](#) – LACP resource module
- [nxos_lacp_interfaces module](#) – LACP interfaces resource module
- [nxos_lag_interfaces module](#) – LAG interfaces resource module

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

Ansible Modules

- Always use the fully qualified collection name (FQCN) for the module
- The modules require parameters with values assigned that define your configuration intent
- Documentation provides details on default values and required values



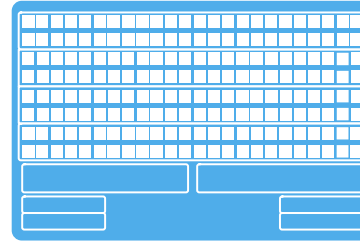
Ansible Concepts

VXLAN EVPN Fabric

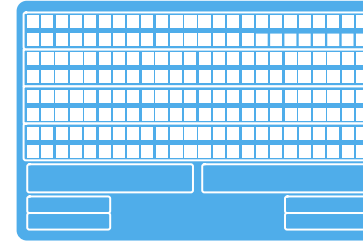
IaC – Nexus as Code

Common configuration
(Hostnames, Features, etc)

10.15.1.11 - S1



S2 - 10.15.1.12



10.15.1.13 - L1



10.15.1.14 - L2

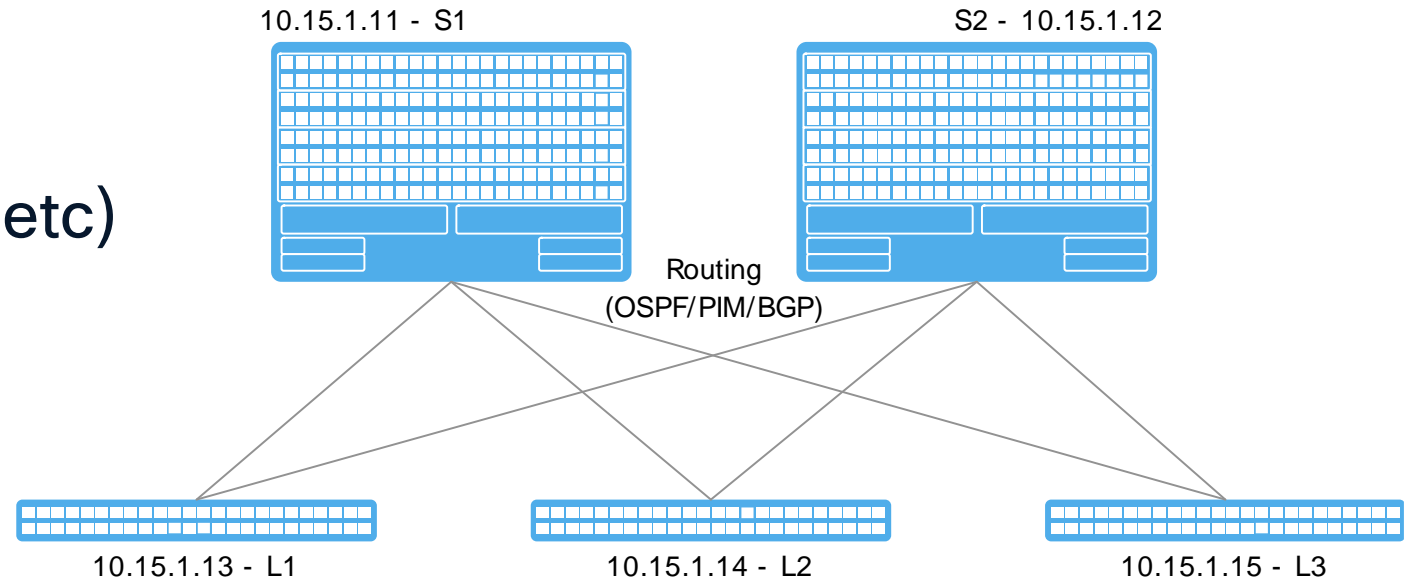


10.15.1.15 - L3

VXLAN EVPN Fabric

laC – Nexus as Code

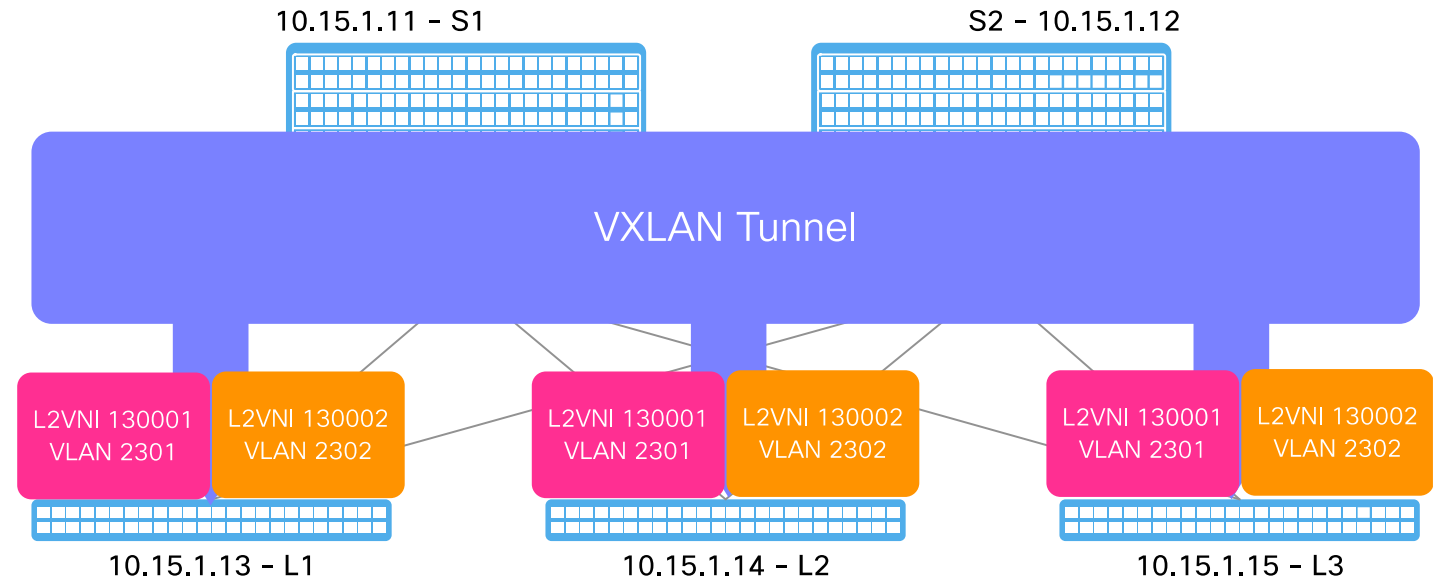
Underlay Configuration (Interfaces, Routing protocols, etc)



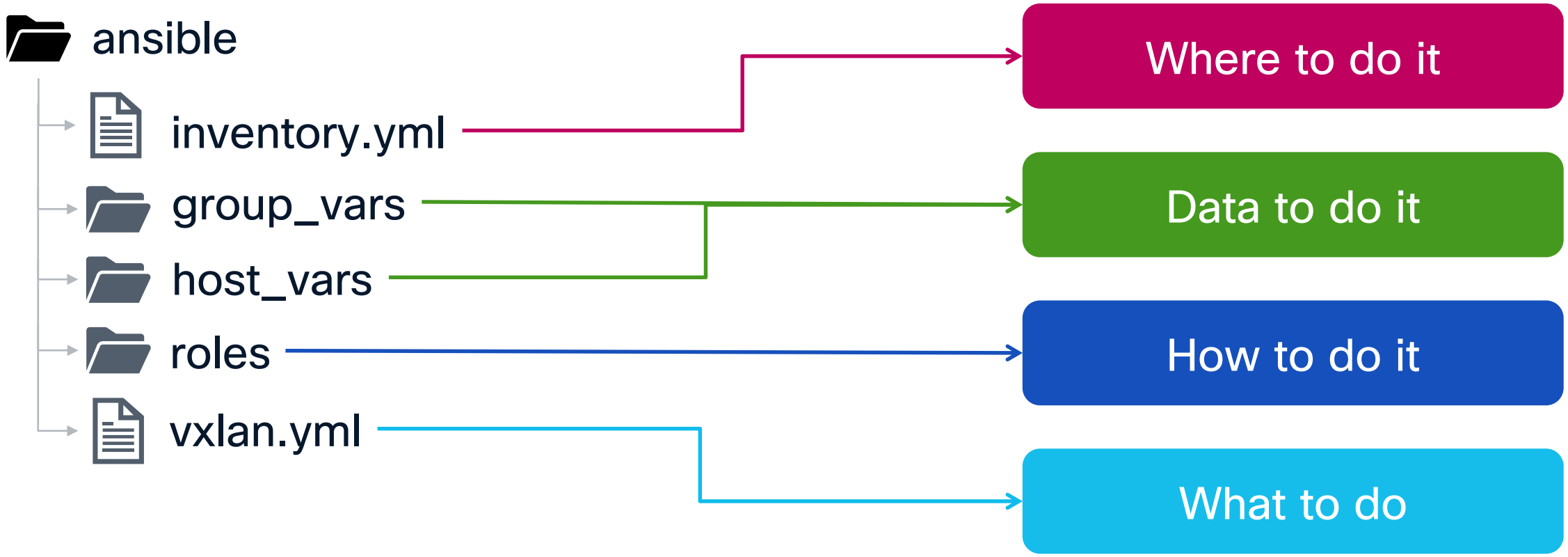
VXLAN EVPN Fabric

laC – Nexus as Code

Overlay Configuration (VRFs, VLANs, SVI, etc)

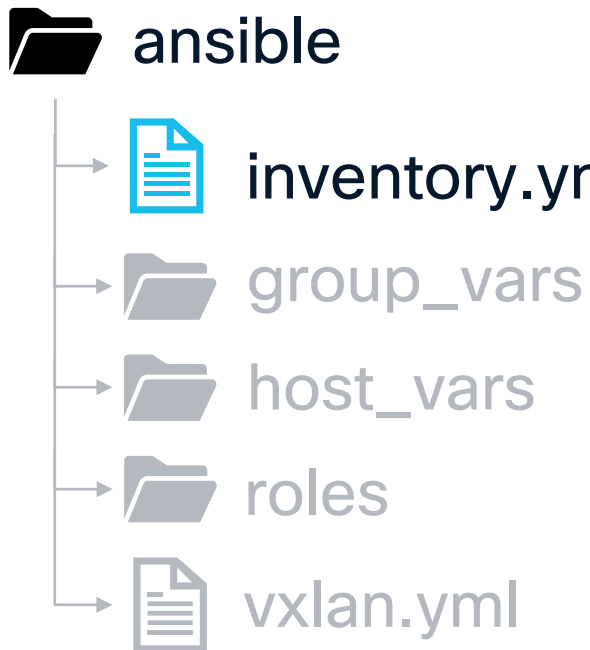


Ansible Directory Structure



Ansible Inventory

Where to do it



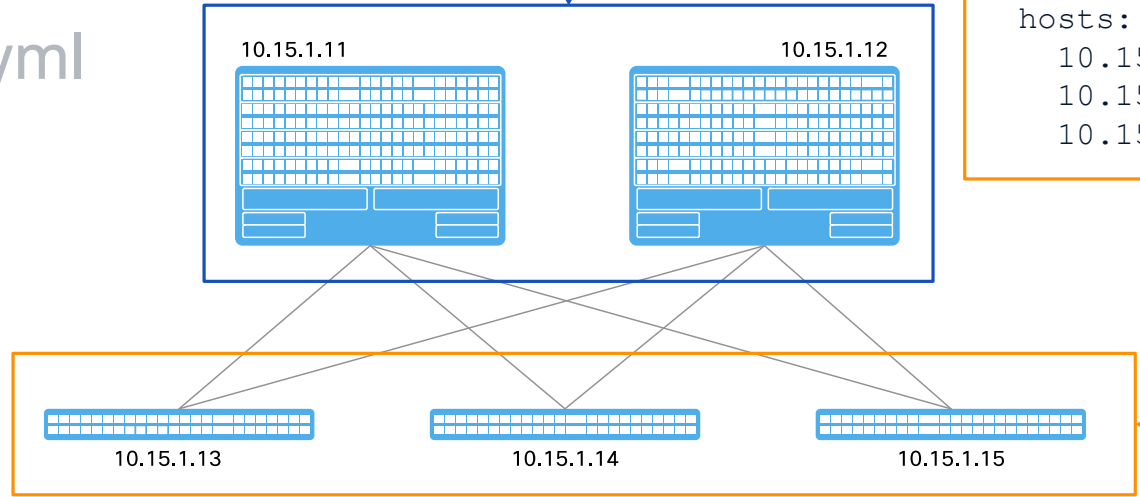
```
---
# main inventory file

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:
```

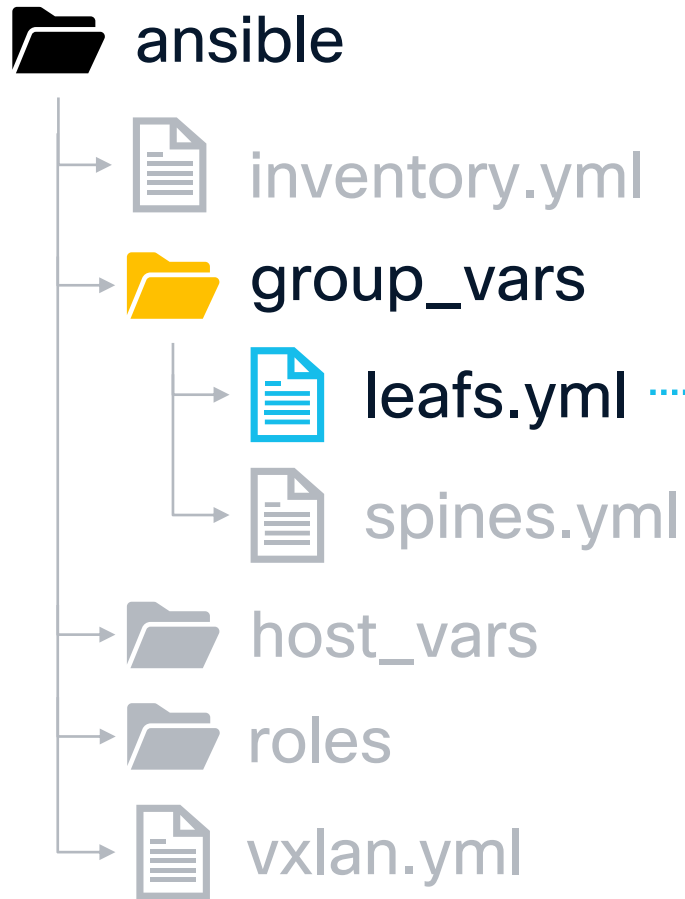
```
  leaves:
    hosts:
      10.15.1.13:
      10.15.1.14:
      10.15.1.15:
```

Connection information for switches



Ansible Group Vars

Data to do it



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

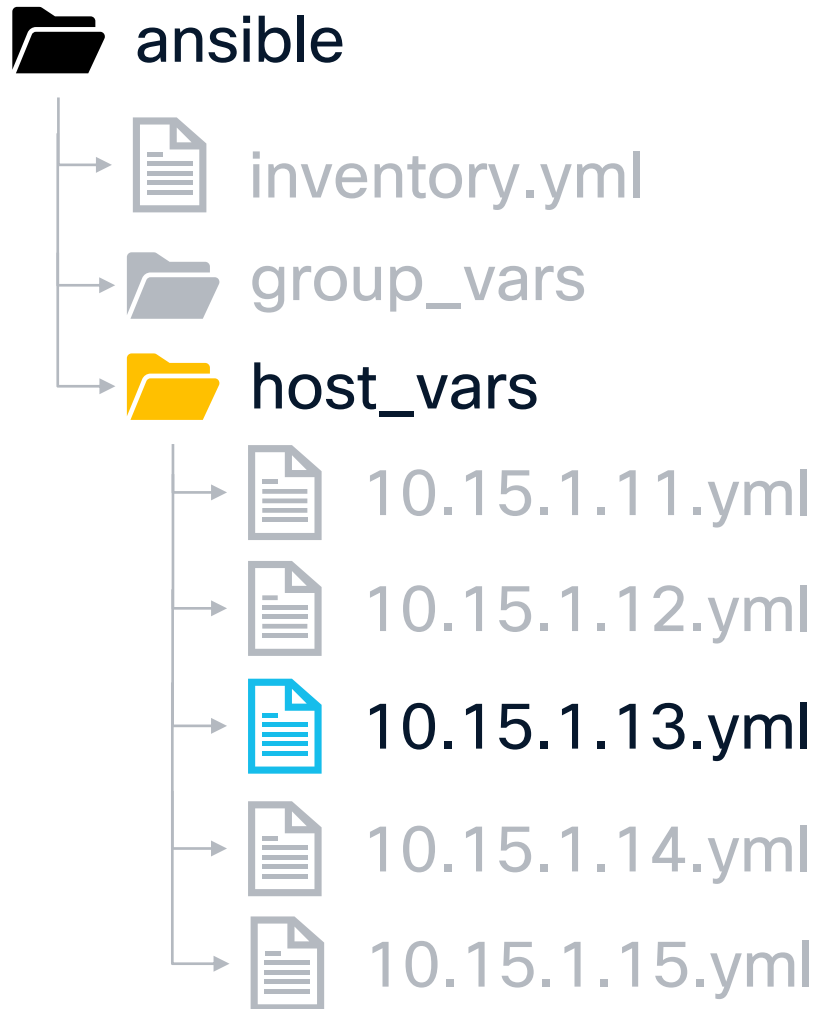
features:
  - ospf
  - pim
  - bgp
  - nv overlay
  - vn-segment-vlan-based
  - interface-vlan

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
```

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

Ansible Host Vars

Data to do it



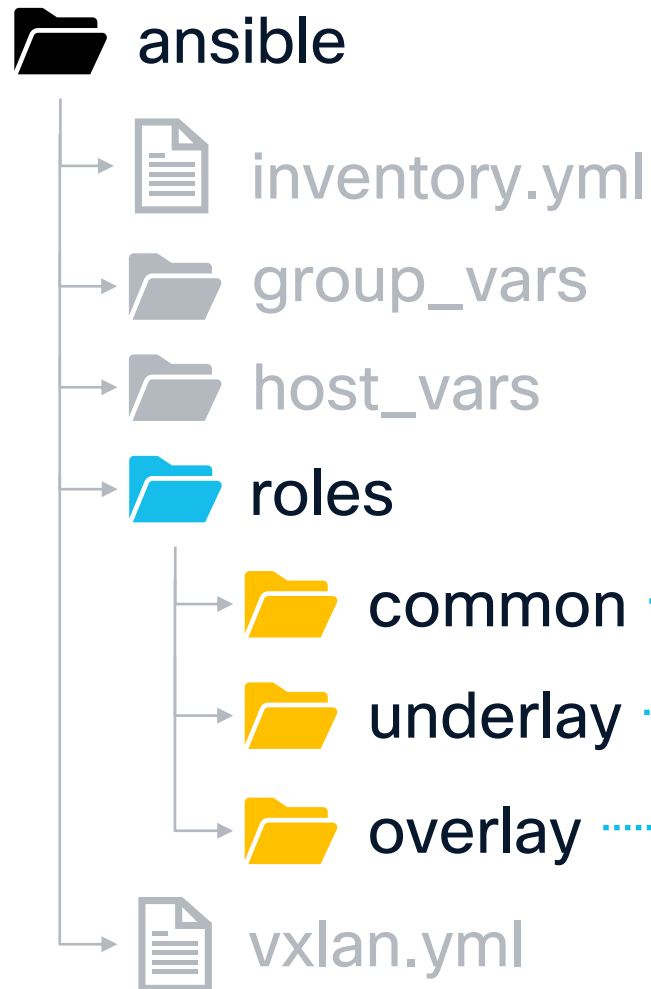
```
---
> # vars file for L1

hostname: L1

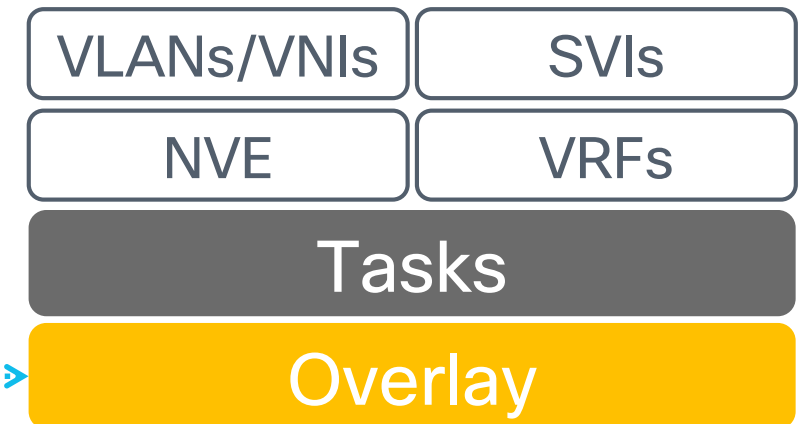
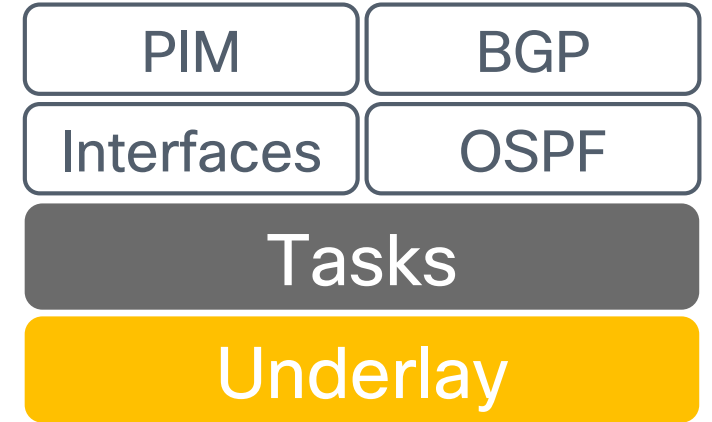
layer3_physical_interfaces:
  - interface: ethernet1/11
    description: To S1 Eth1/1
    mode: layer3
    ip_address: 10.1.1.1
    mask: 31
    mtu: 9216
  - interface: ethernet1/12
    description: To S2 Eth1/1
    mode: layer3
    ip_address: 10.2.2.1
    mask: 31
    mtu: 9216
```

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

Ansible Roles - Tasks

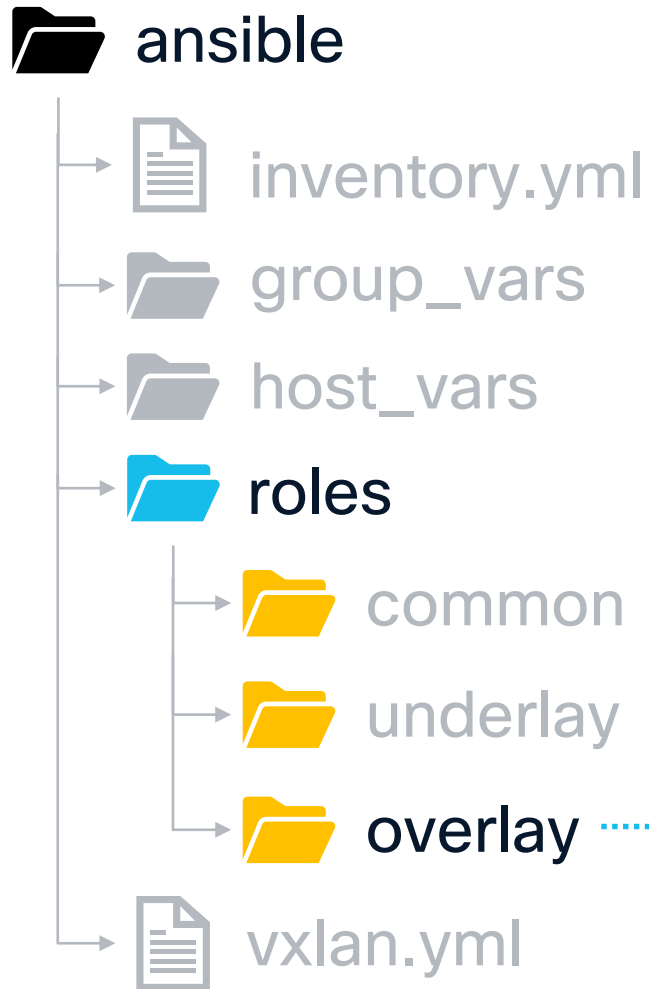


How to do it

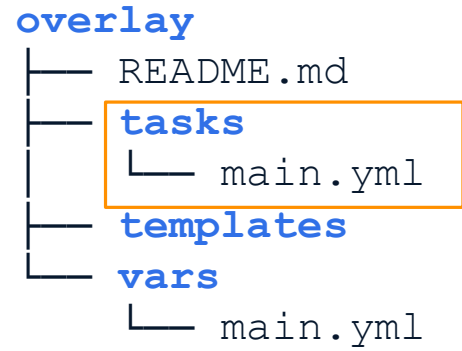


Ansible Roles

How to do it



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

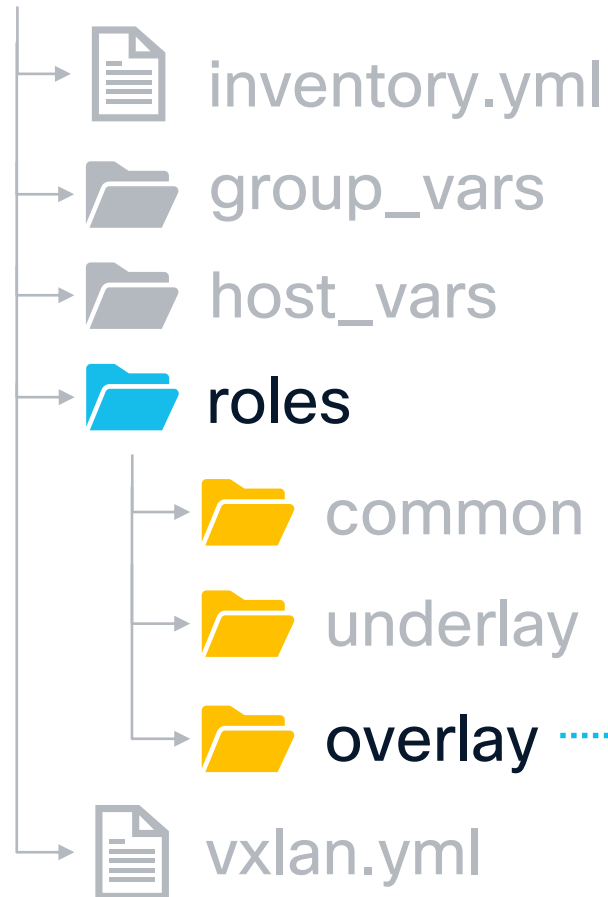


This is
how we
do it

Ansible Roles - Tasks

How to do it

ansible



roles/overlay/tasks/main.yml

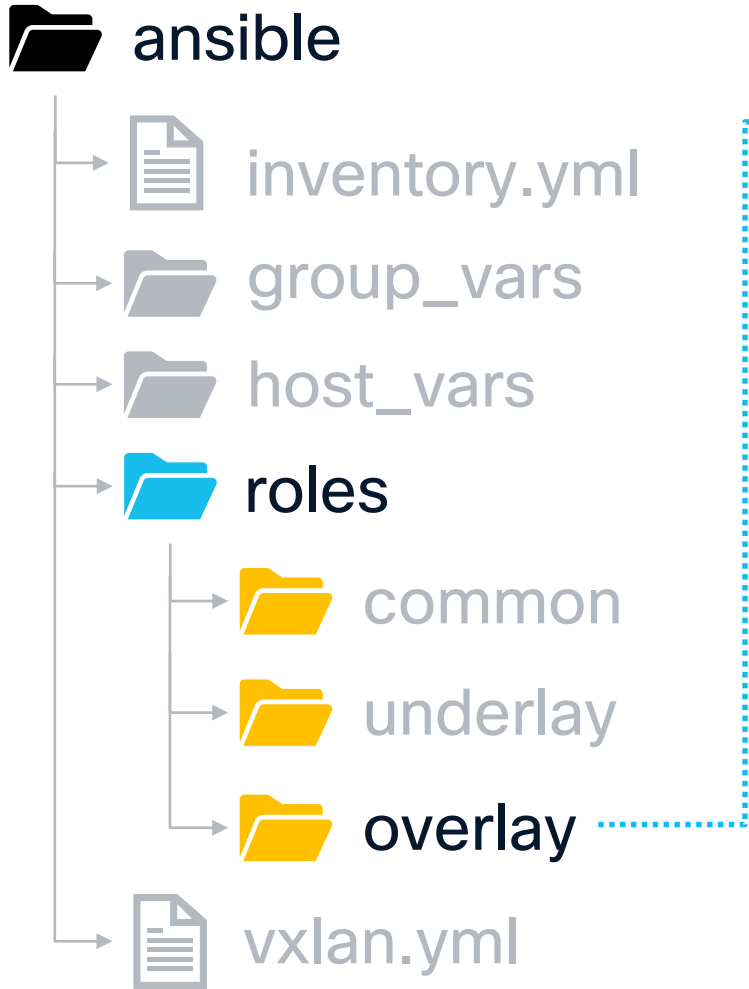
```
---
# tasks file for roles/overlay

- 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 do Ansible task parameters this way!

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

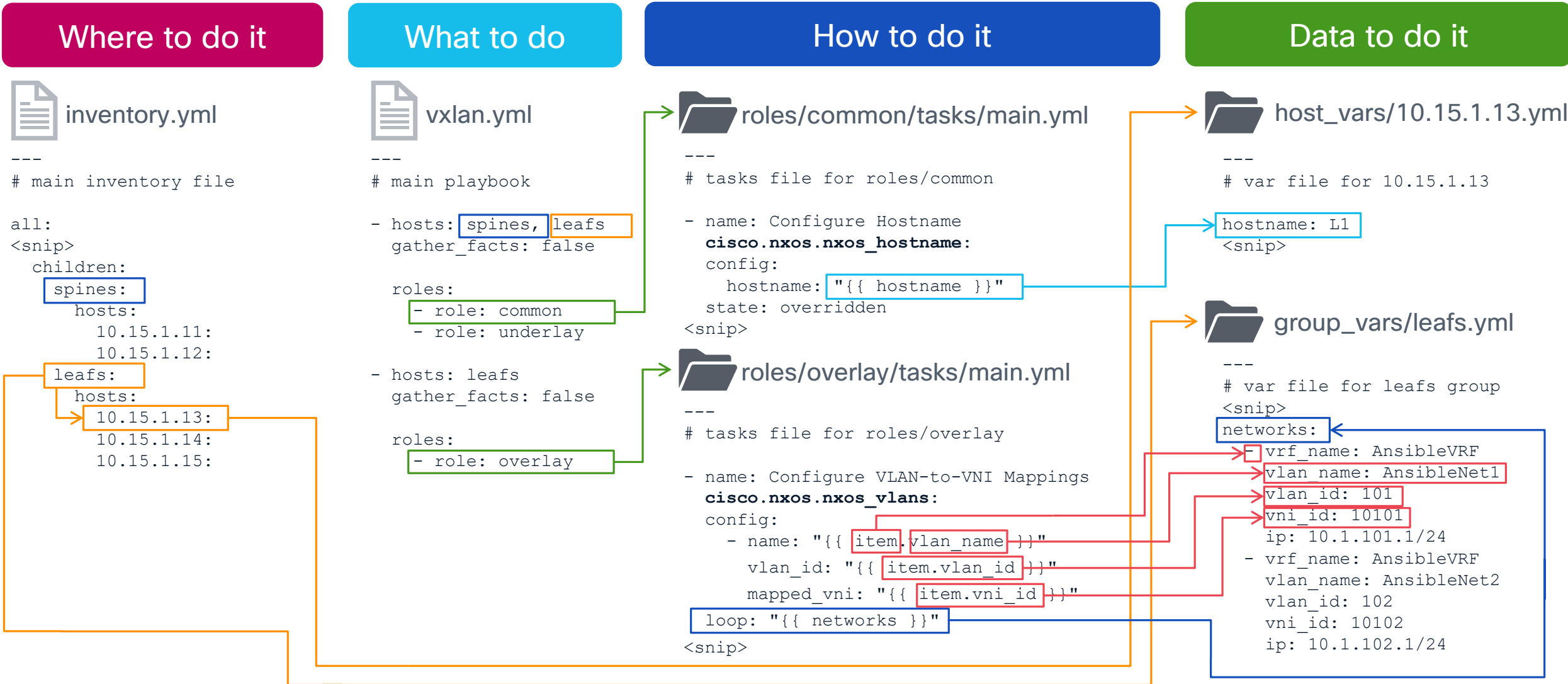
```
---
# var file for 10.15.1.13

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:
    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/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 }}
  vni_id: {{ 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/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 }}
  vni_id: {{ 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
```

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

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 Command 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>
```

An Example

Executing Ansible Playbooks

```
ansible-playbook -i inventory.yml vxlan.yml
```

```
~/development/ansible/BRKDCN-2946 > main mtarking@ansible-ubuntu-01 < 03:59:50 PM
brkdcn-2946 3.11.2 > ansible-playbook -i inventory.yml vxlan.yml

PLAY [spines, leafs] *****
TASK [common : Configure Hostname] *****
```

Infrastructure as Code with NDFC and Ansible

Automation



Accelerate provisioning from days to minutes

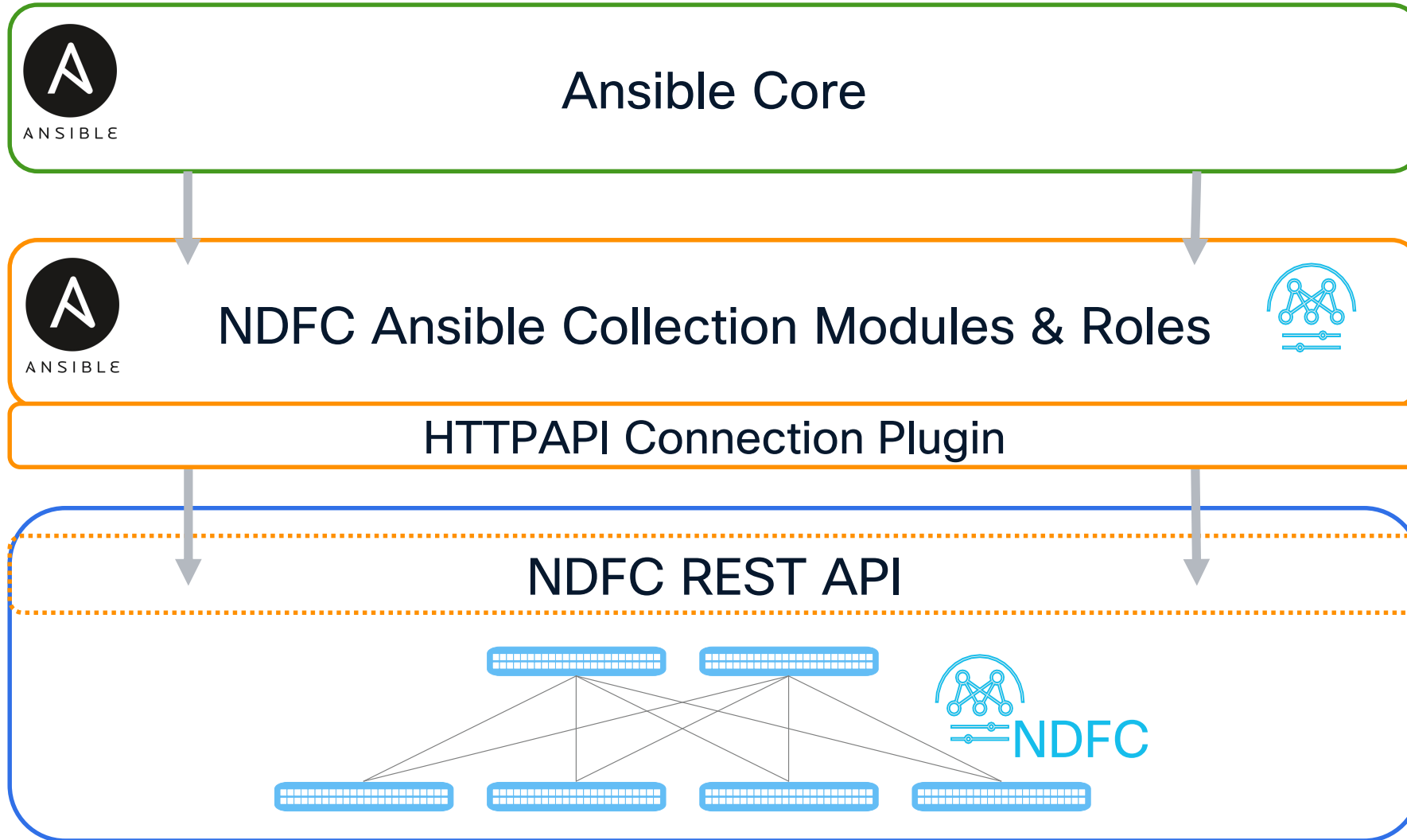
Easy to understand approach to auto-bootstrapping of entire fabric

Rapid Deployment with Fabric Builder
best practice templates for VXLAN-EVPN

DevOps friendly

Enhanced Programmability

NDFC DevOps “Friendliness”



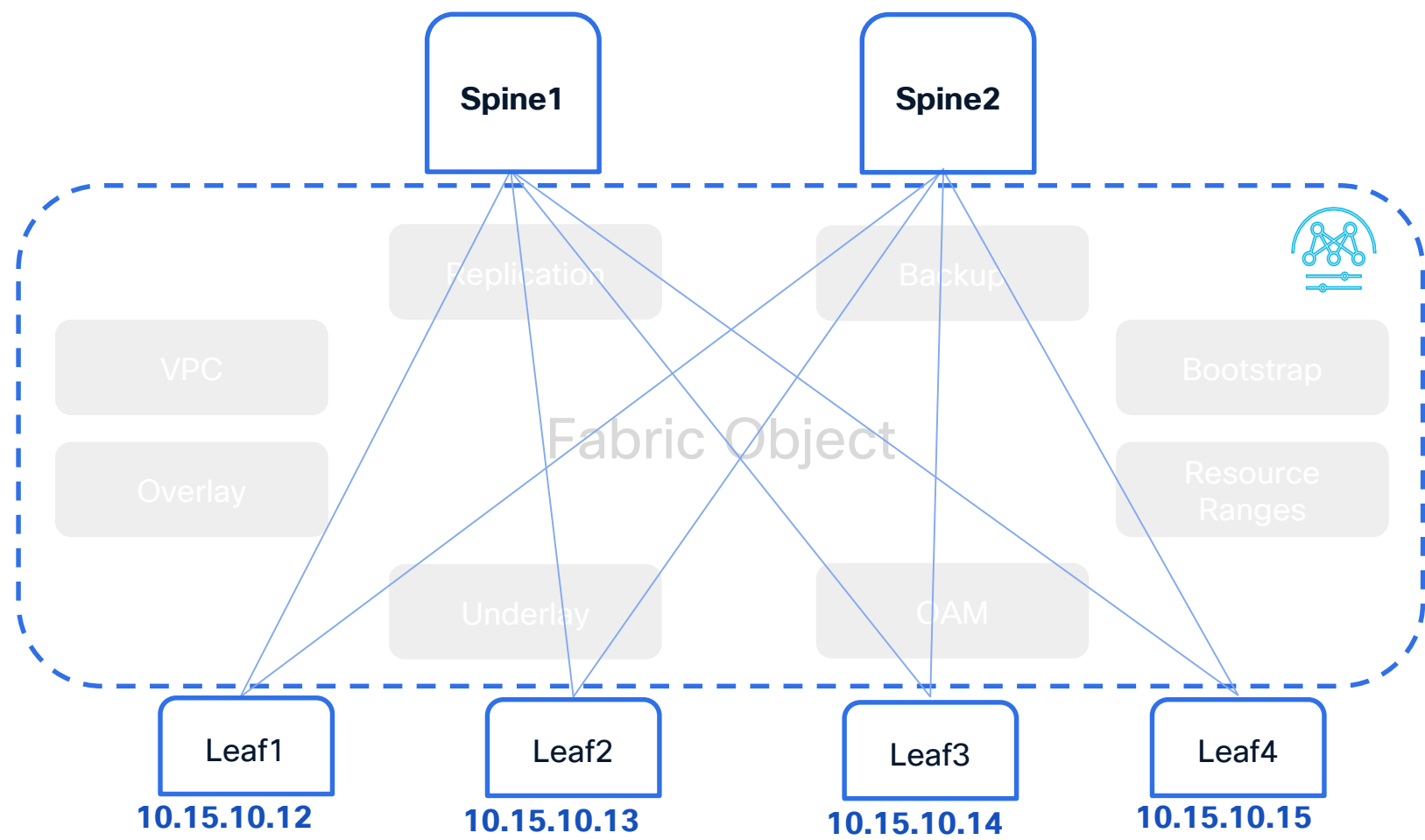


Workflow Step1: Fabric Management



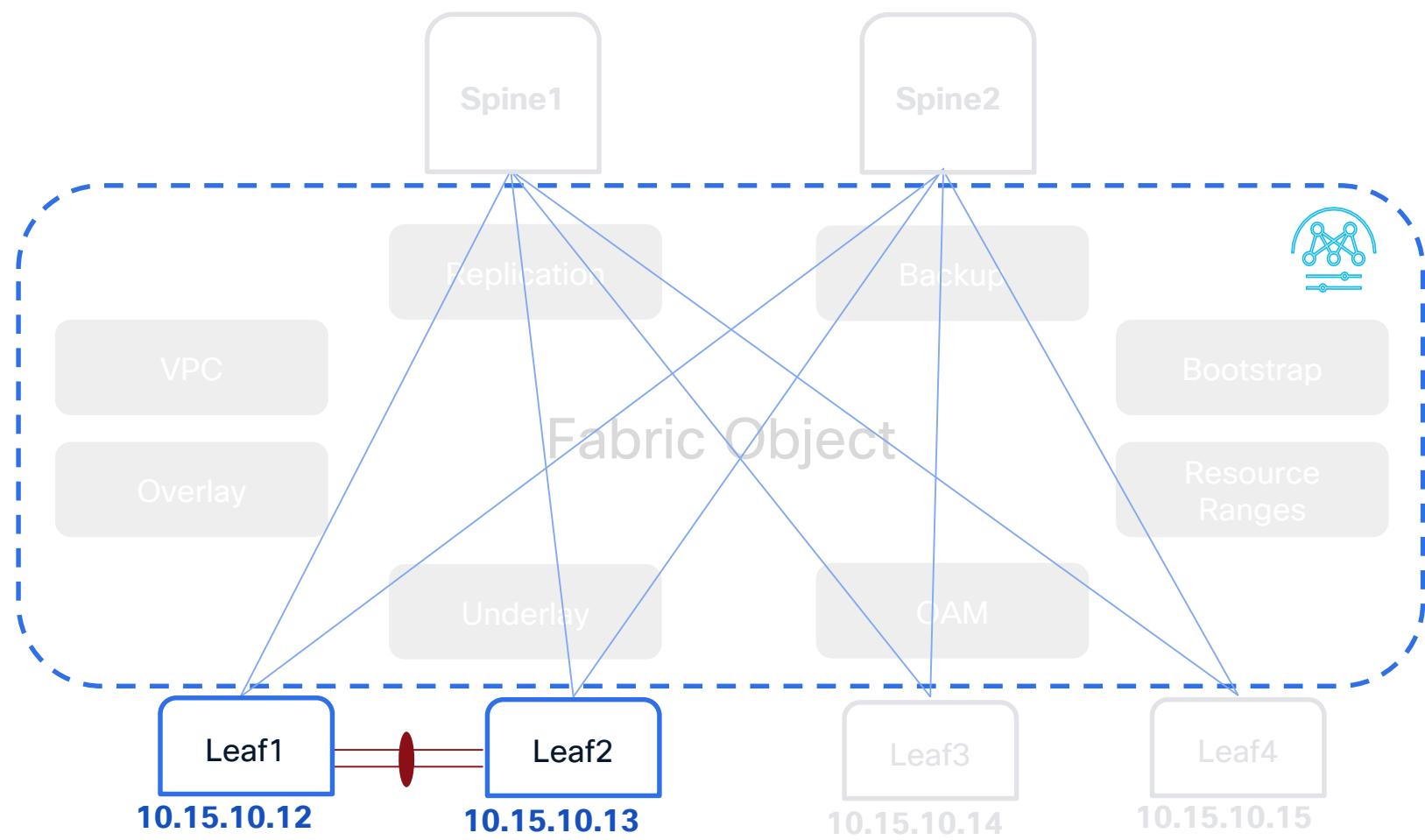


Workflow Step2: Inventory Management



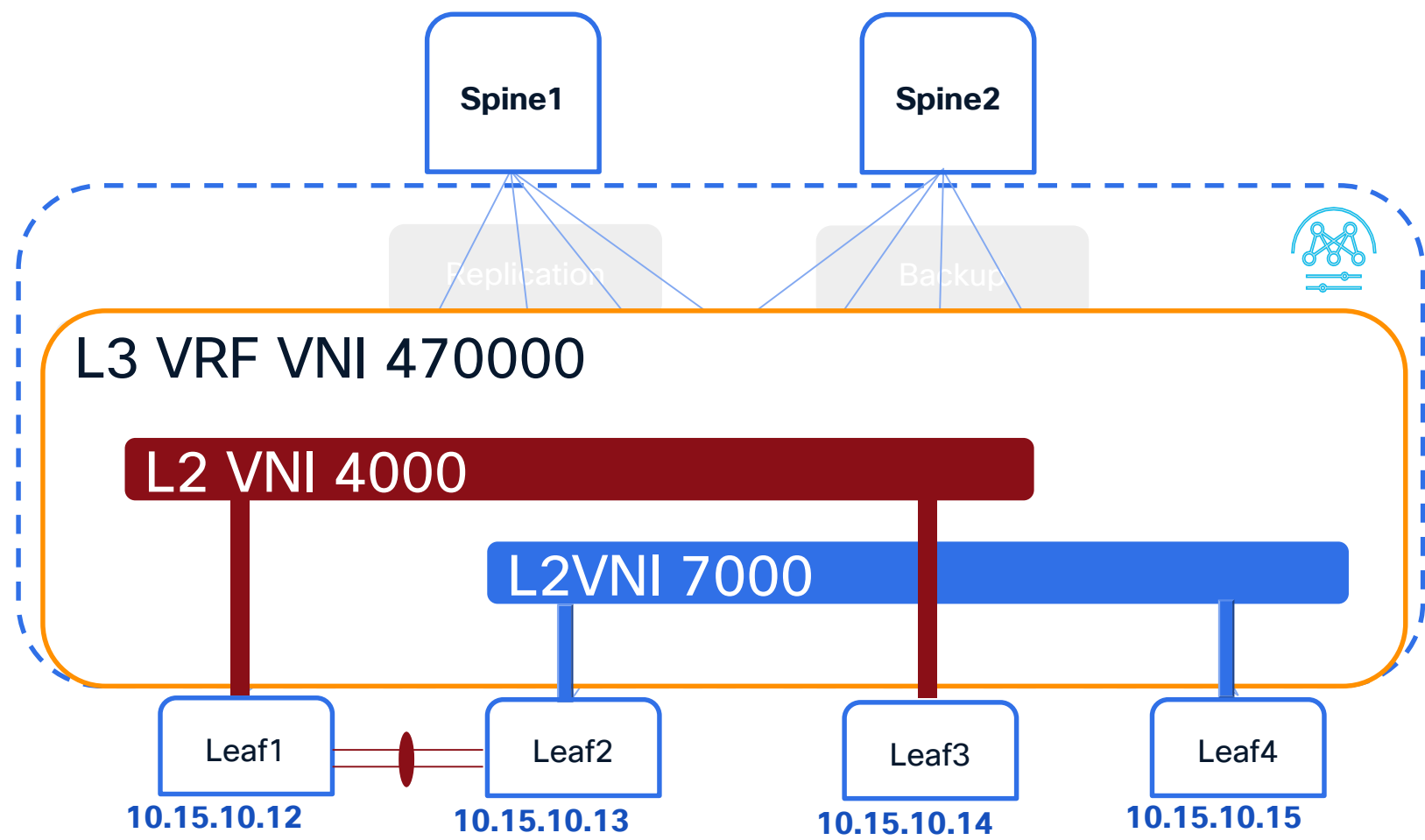


Workflow Step3: vPC Pair Management





Workflow Step4: Overlay Management



NDFC Collection Modules - Toolbox

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



20 Modules

Module Name	Purpose
cisco.dcnm.dcnm_rest	General Do Anything Module
cisco.dcnm.fabric (Ver 3.5.0)	Manage creation and configuration of NDFC fabrics
cisco.dcnm.dcnm_inventory	Add Devices To Fabric
cisco.dcnm.dcnm_vpc_pair (Ver 3.5.0)	Manage vPC switch pairs
cisco.dcnm.dcnm_interface	Configure Fabric Interfaces
cisco.dcnm.dcnm_vrf	Add Overlay VRFs
cisco.dcnm.dcnm_network	Add Overlay Networks / VLANs
cisco.dcnm.dcnm_template	Create Custom Templates
cisco.dcnm.dcnm_policy	Create Policies Based On Templates
cisco.dcnm.dcnm_links	Manage Fabric Links
cisco.dcnm.dcnm_resource_manager	Manage Fabric Resources
cisco.dcnm.dcnm_image_upload (Ver 3.5.0)	Manage Switch Images
cisco.dcnm.dcnm_image_policy (Ver 3.5.0)	Manage Image Policies
cisco.dcnm.dcnm_image_upgrade (Ver 3.5.0)	Manage Images for Nexus Switches
cisco.dcnm.dcnm_bootflash (Ver 3.6.0)	Bootflash management for Nexus Switches
cisco.dcnm.dcnm_maintenance_model (Ver 3.6.0)	Manage maintenance mode configuration of Nexus Switches
cisco.dcnm.dcnm_service_node	Manage Service Nodes
cisco.dcnm.dcnm_service_policy	Manage Service Policy
cisco.dcnm.dcnm_service_route_peering	Manage Service Route Peering

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`





+



ANSIBLE

Let's Build Something With NDFC and Ansible Together!



Workflow Step 1: Create VXLAN Fabric

Actions ^

- Create Fabric
- Edit Fabric
- Delete Fabric

1

Create Fabric

2

Fabric Name
fabric-stage

Pick Fabric
[Data Center VXLAN EVPN >](#)

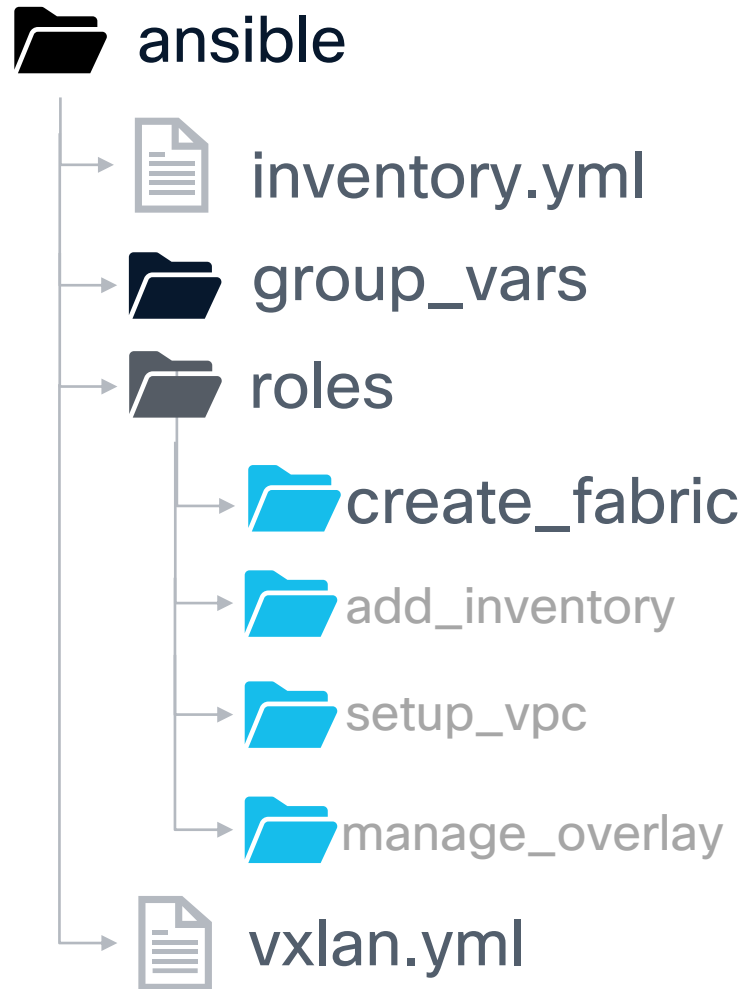
General Parameters | Replication | vPC | Protocols | Advanced | Resources | Manageability | Bootstrap | Configuration Backup | Flow Monitor

BGP ASN*
 1-4294967295 | 1-65535[.0-65535] It is a good practice to have a unique ASN for each Fabric.

Enable IPv6 Underlay
 If not enabled, IPv4 underlay is used

Enable IPv6 Link-Local Address
 If not enabled, Spine-Leaf interfaces will use global IPv6 addresses

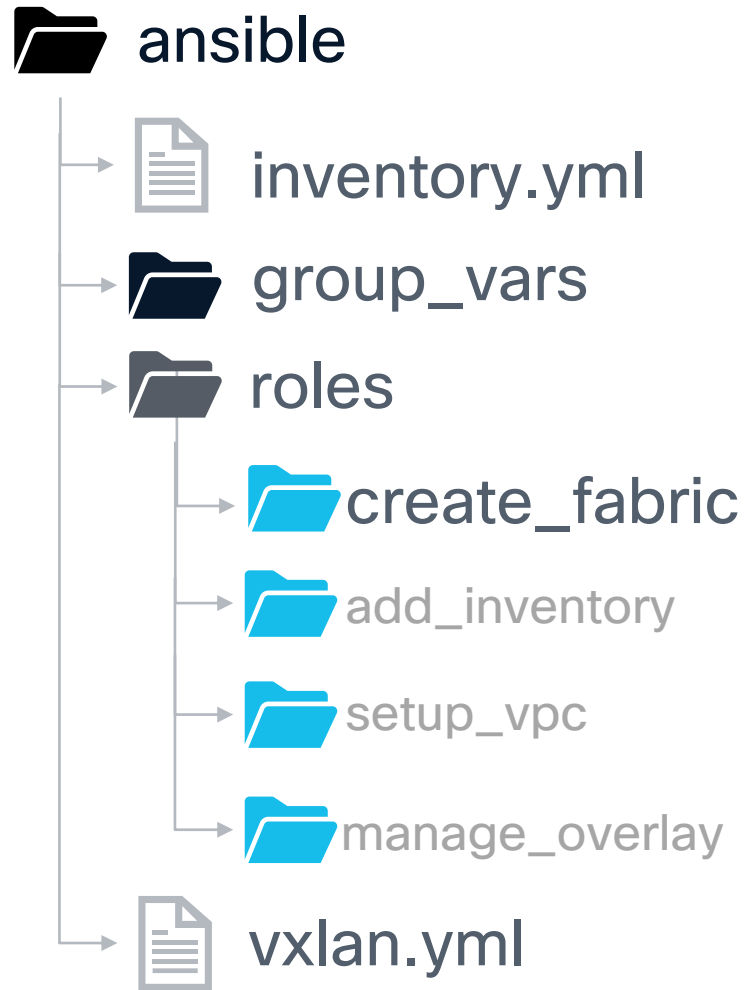
Fabric Interface Numbering*
 Numbered(Point-to-Point) or Unnumbered



```
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: CL_STAGING
        FABRIC_TYPE: VXLAN_EVPN
        BGP_AS: 65000
        ANYCAST_GW_MAC: 0001.aabb.ccdd
        UNDERLAY_IS_V6: false
```

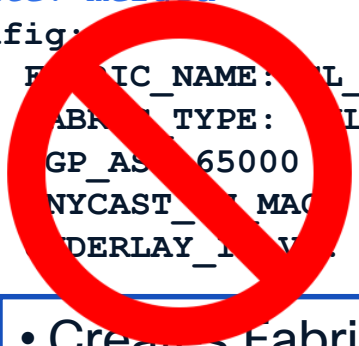
- Creates Fabric
- Handles Mutually Exclusive Properties
- Supports multiple types (VXLAN_EVPN, MSD, ISN, LAN_CLASSIC)



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_TYPE: ...
      GP_AS: 65000
      UNICAST ... MAC ...
      DERLAY ...
```



Let's discuss a new and better way!

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

VXLAN as Code

VXLAN as Code

VXLAN as Code aims to reduce time to value by lowering the barrier of entry to network orchestration through simplification, abstraction, and curated examples.

It allows users to instantiate network fabrics in minutes using an easy to use, **opinionated data model**. It takes away the complexity of having to deal with references, dependencies or loops.

Users can focus on **describing the intended configuration** while using a set of maintained and tested Ansible Modules without the need to understand the low-level network device configuration.

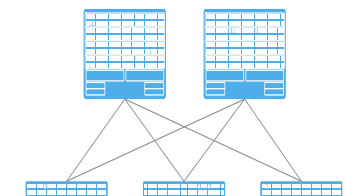
<https://netascode.cisco.com>



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



ANSIBLE



Data Model

Built by **network engineers**

Built to allow us to represent network complexity in an easier structure where we can incorporate validation and testing

We use Yamale* to construct schema and insert rule enhancements

```
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  
  
  bgp:  
    authentication_enable: true  
    authentication_key_type: 3  
    authentication_key: 014903c077689077e5b4548432c3d91d
```

23 and Me
open-source project <https://github.com/23andMe/Yamale>

Default Values

VXLAN as Code comes with **pre-defined default values** based on common best practices.

In some cases, those default values might not be the best choice for a particular deployment and can be overwritten if needed.

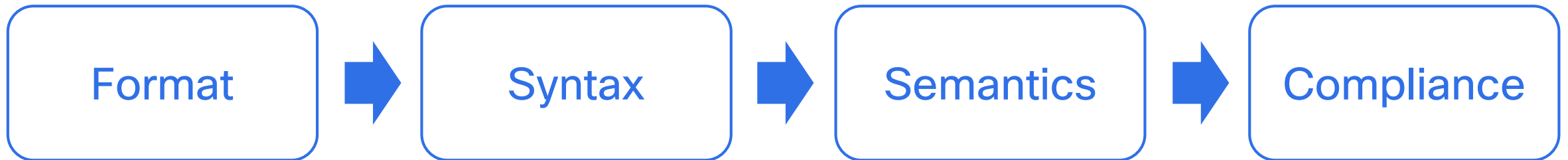
```
defaults:  
  vxlan:  
    overlay:  
      vrfs:  
        loopback_route_tag: 12345  
        vrf_int_mtu: 9216
```

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

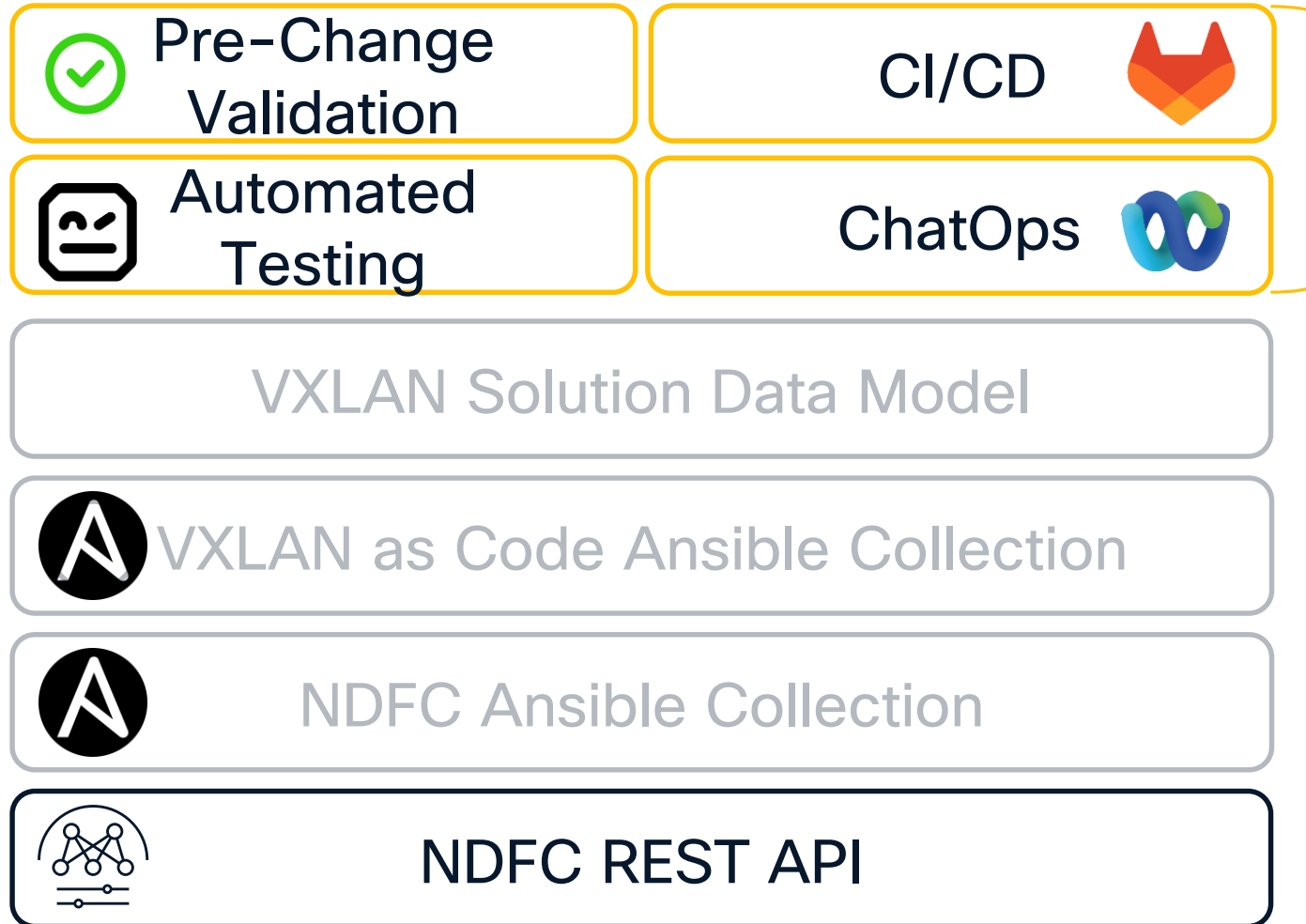
Pre-Change Validation

As the complexity of the configuration and the underlying data model increases automated validation before deploying anything in a production environment becomes a critical aspect.

Several tools can be used to ensure that the provided input data is valid, but also that common best practices and formatting guidelines are being followed.



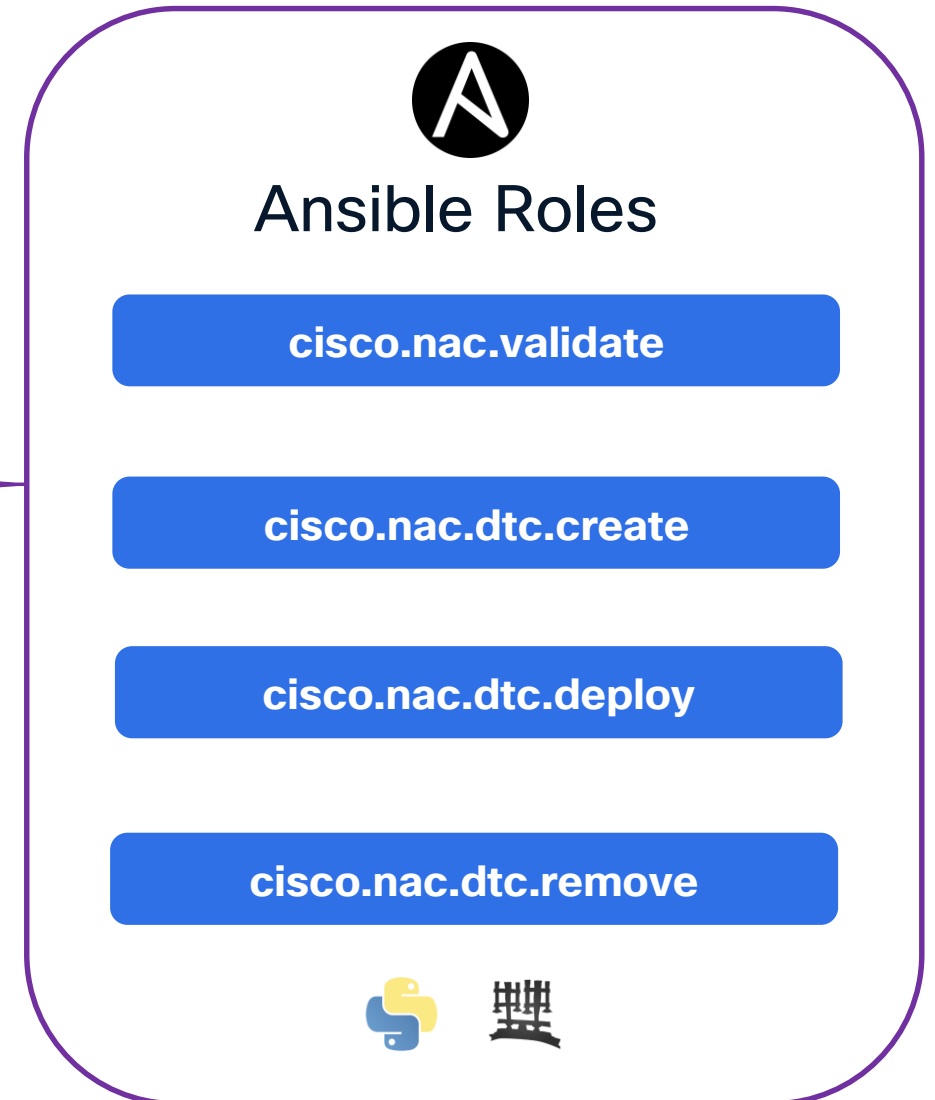
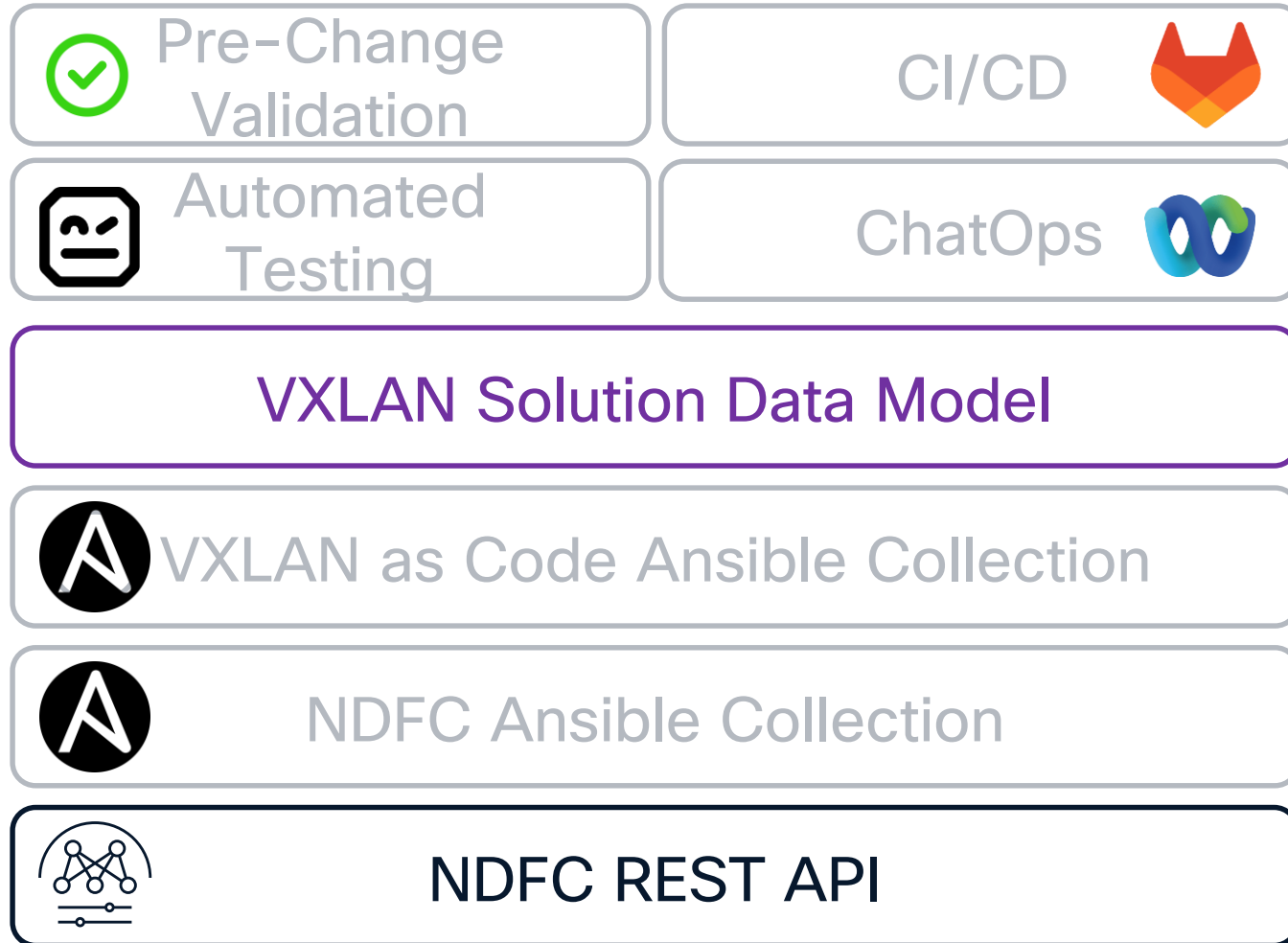
NDFC as Code Building Blocks



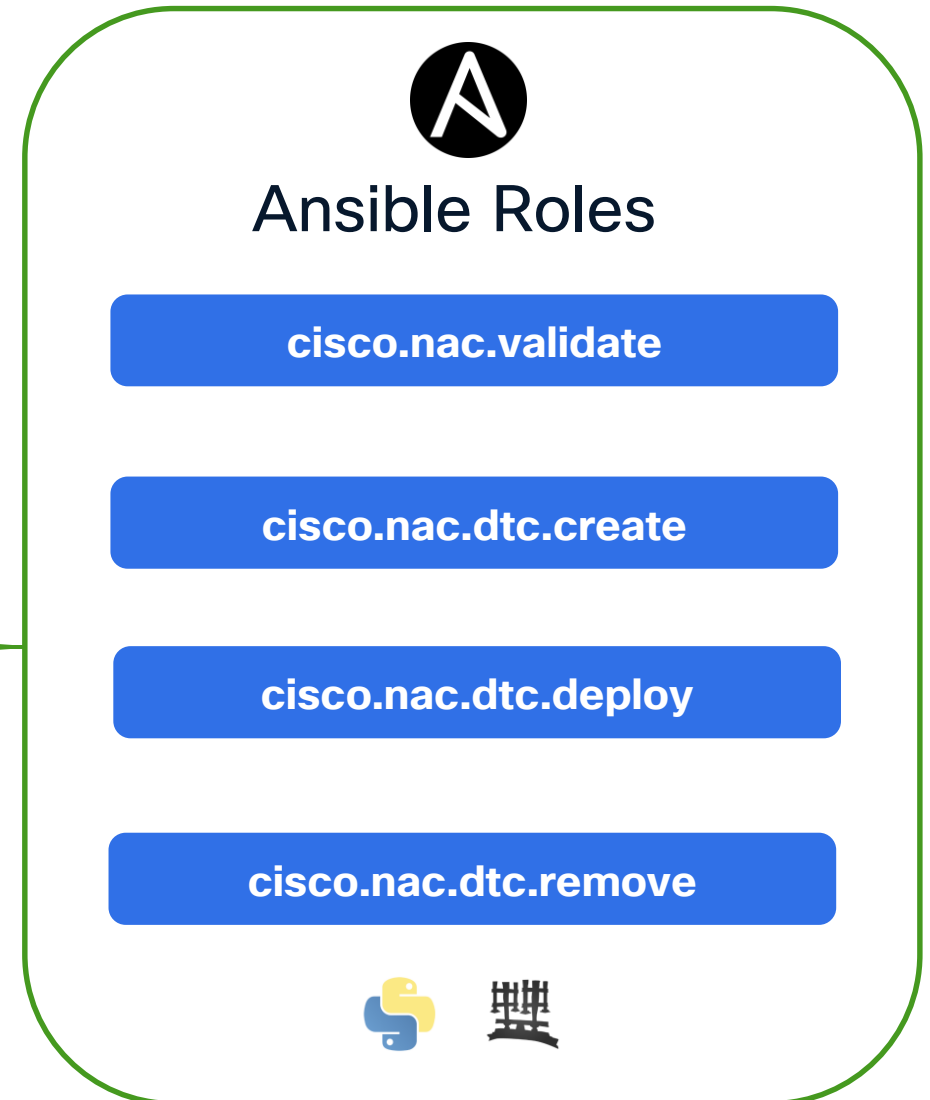
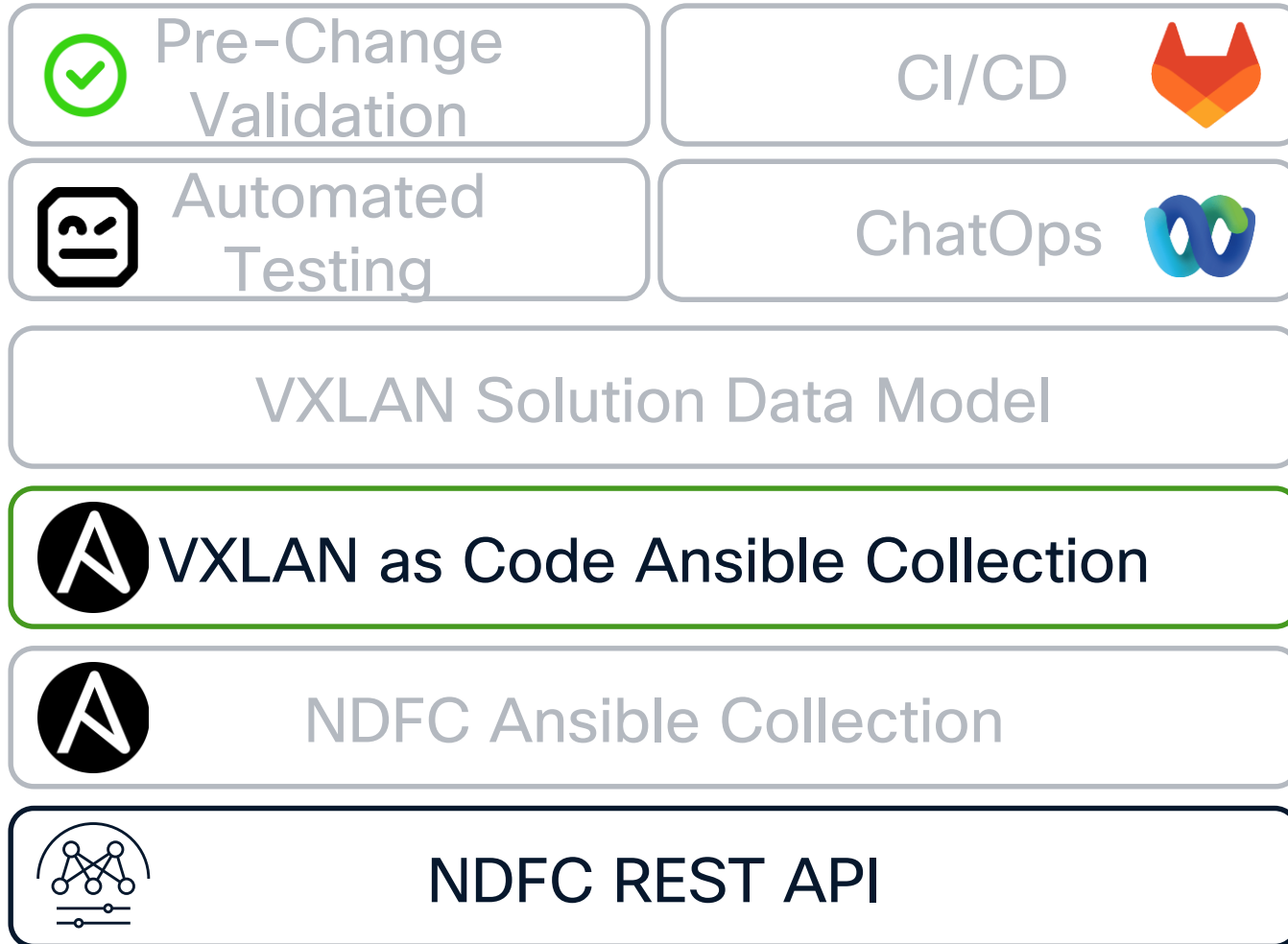
Automation Objectives

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

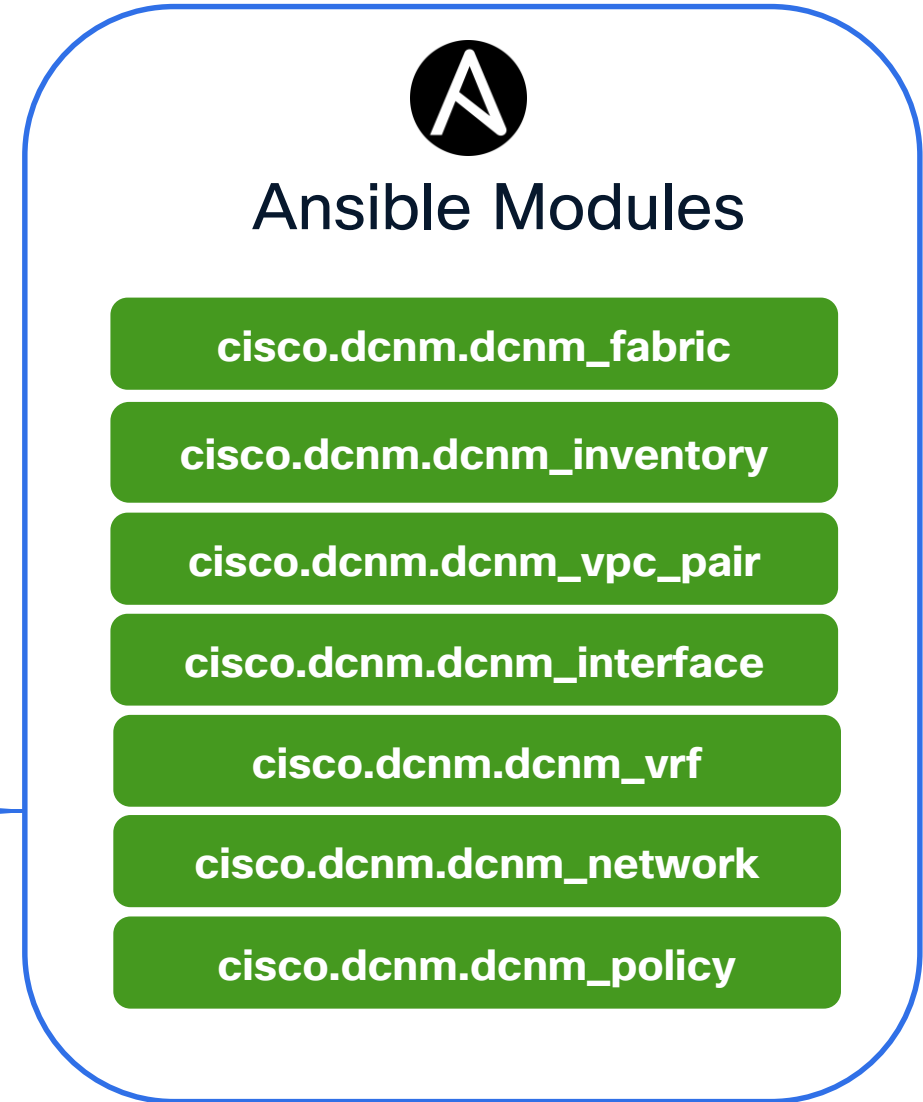
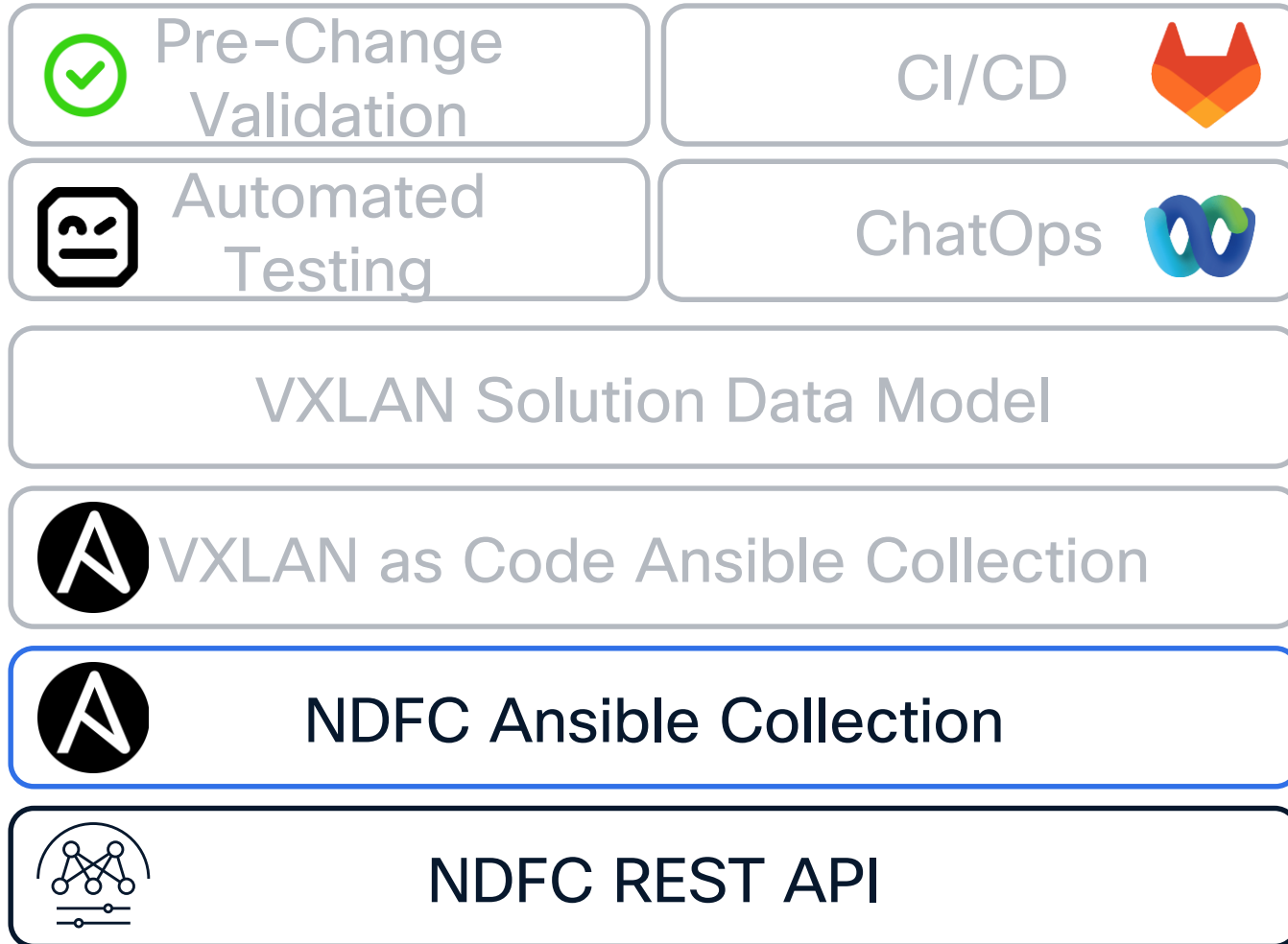
NDFC as Code Building Blocks



NDFC as Code Building Blocks

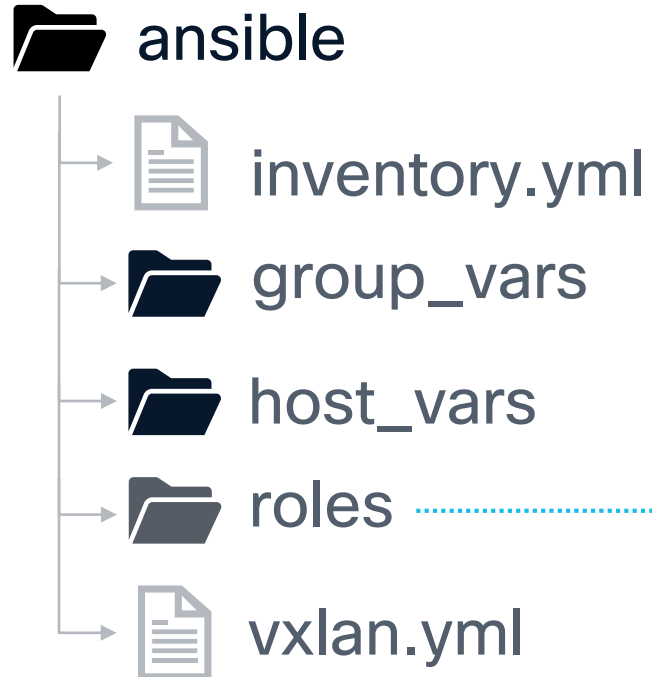


NDFC as Code Building Blocks



Ansible Playbook

What to Do



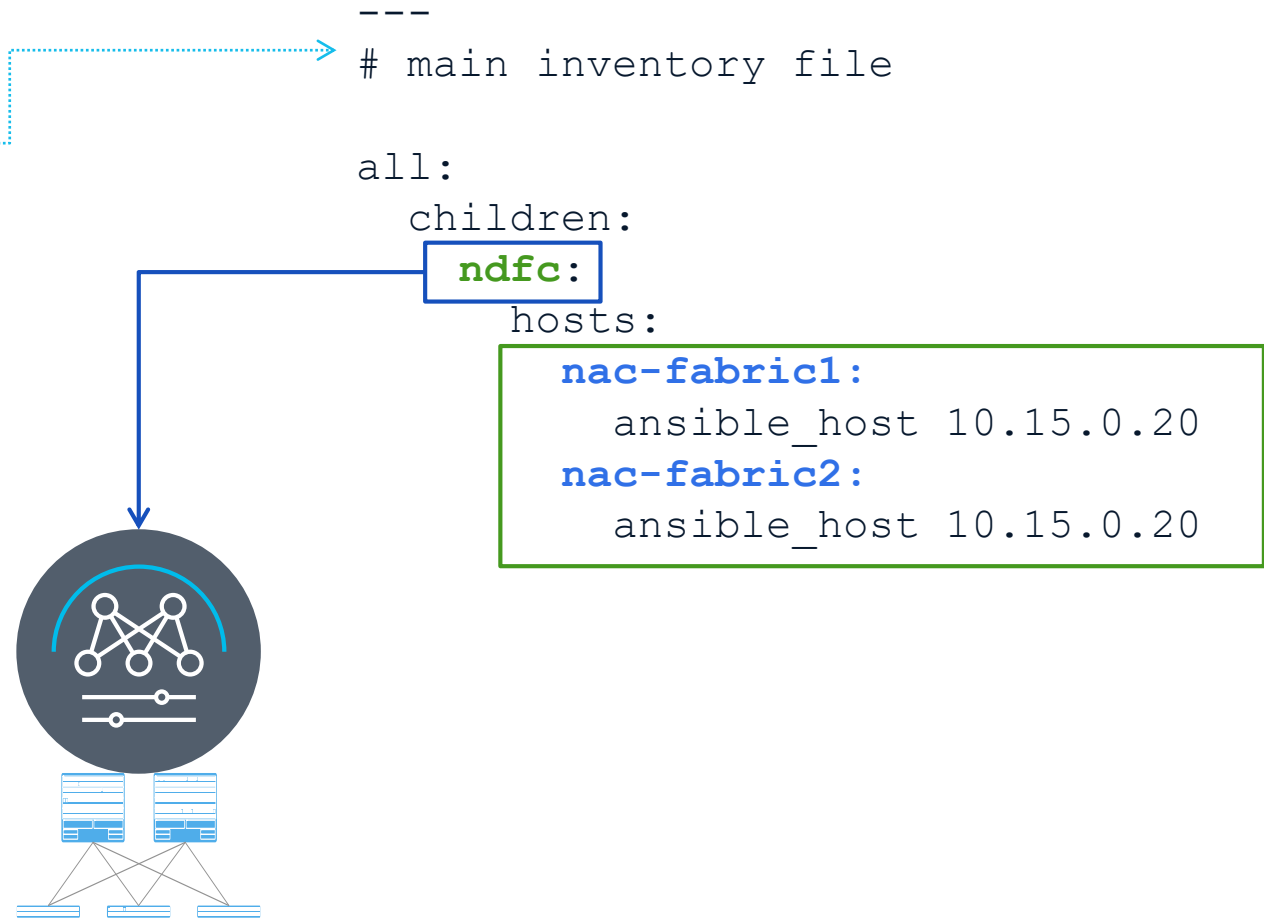
```
---
# main NaC playbook

- name: Build VXLAN EVPN Fabric on NDFC
  hosts: ndfc-fabric1, ndfc-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'
```

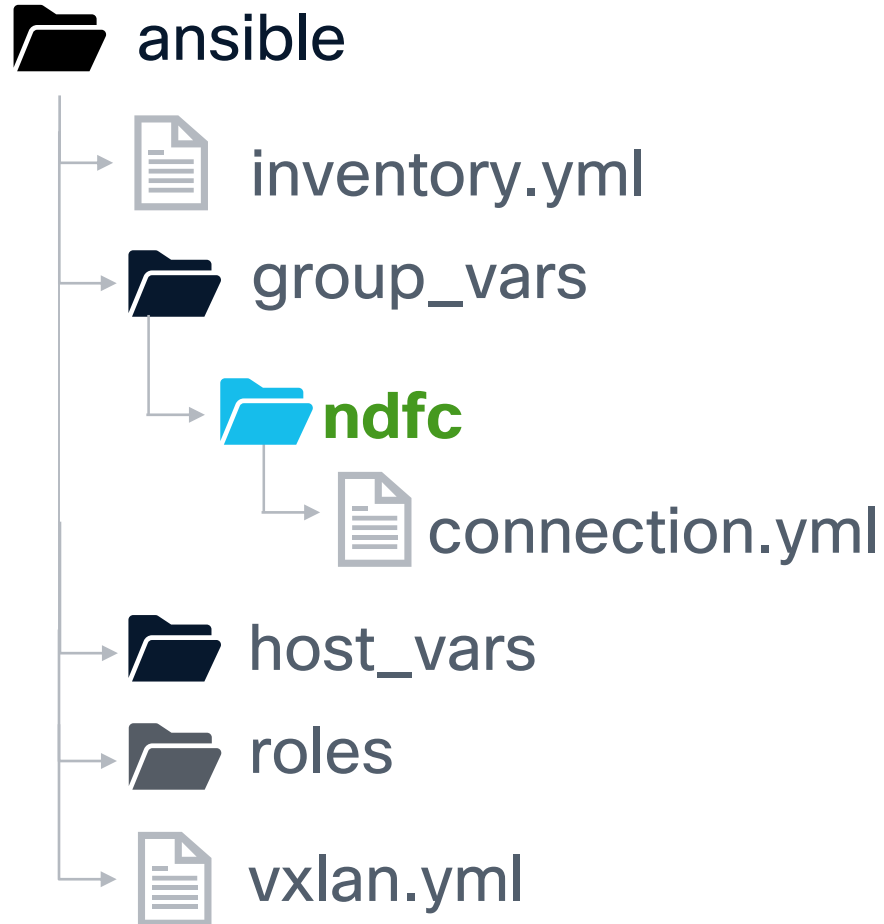
Ansible Inventory

Where to do it



Ansible Group Vars

Where to do it

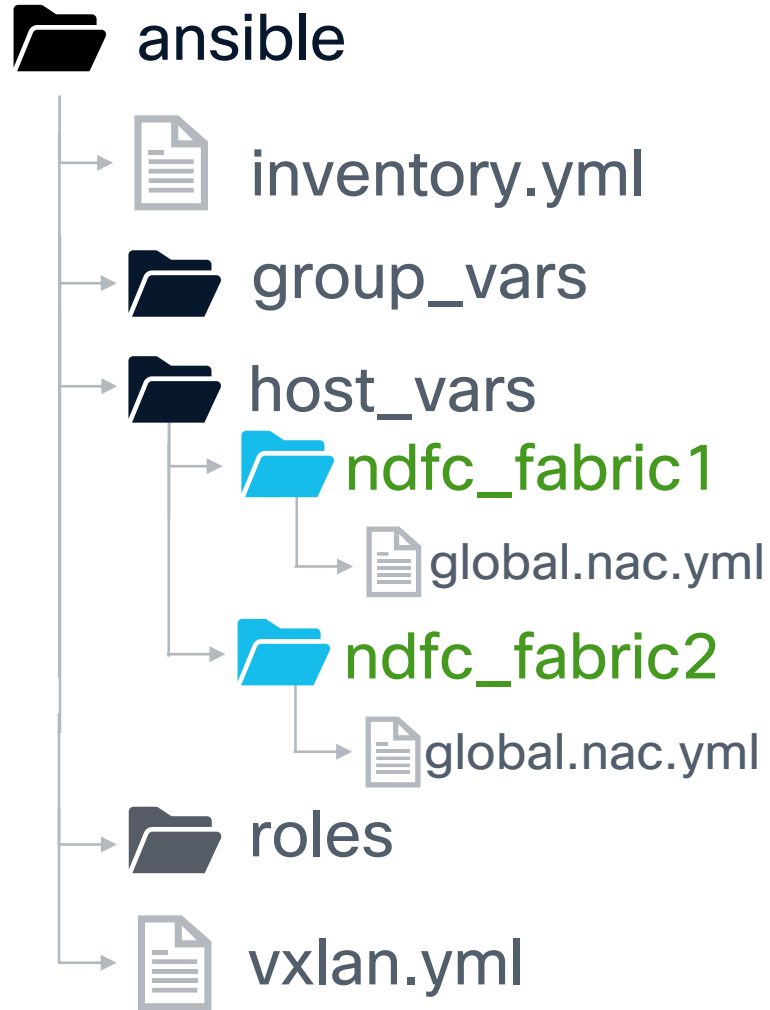


```
---  
# group_vars ndfc connection information  
  
ansible_connection: ansible.netcommon.httpapi  
ansible_user: ndfc_username  
ansible_password: !vault | (ndfc_password)  
ansible_network_os: cisco.dcnm.dcnm  
  
ansible_user: "{{ lookup('env', 'ND_USERNAME') }}"  
ansible_password: "{{ lookup('env', 'ND_PASSWORD') }}"  
  
ndfc_switch_username: "{{ lookup('env', 'NDFC_SW_USERNAME') }}"  
ndfc_switch_password: "{{ lookup('env', 'NDFC_SW_PASSWORD') }}"
```

Connection information for NDFC

Ansible Host Vars

Data to do It



Visual Studio Code `global.nac.yml`

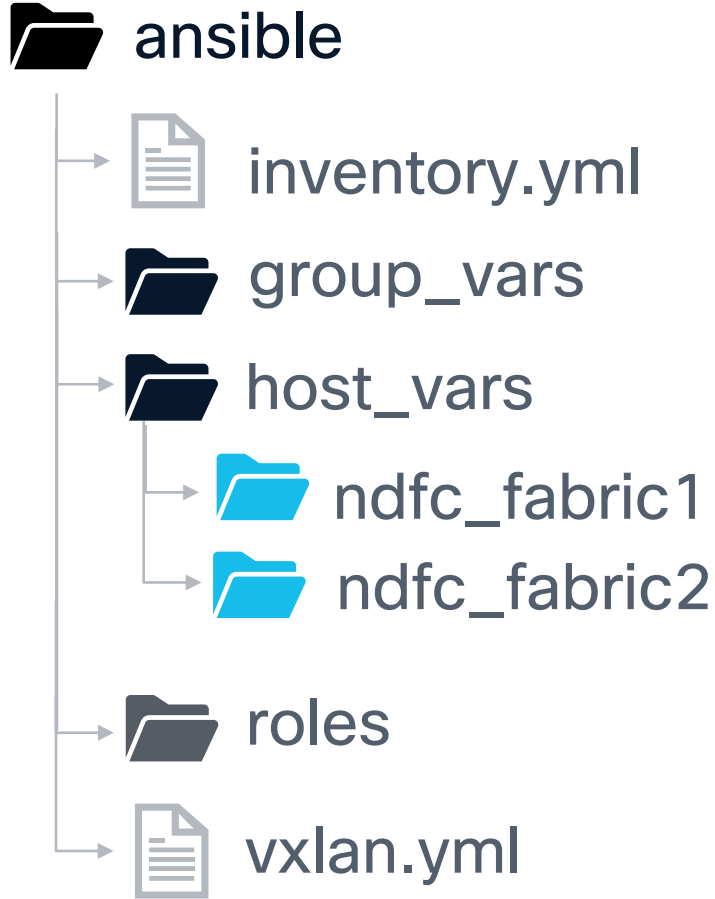
```
---
vxlan:
  global:
    name: ndfc_fabric1
    bgp_asn: 65055
    route_reflectors: 4
    anycast_gateway_mac: 20:20:00:00:00:aa
```

Visual Studio Code `global.nac.yml`

```
---
vxlan:
  global:
    name: ndfc_fabric2
    bgp_asn: 65088
    route_reflectors: 2
    anycast_gateway_mac: 20:20:00:00:00:bb
```

Ansible Host Vars

Data to do It



Visual Studio Code `global.nac.yml`

```
---
vxlan:
  global:
    name: ndfc_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`



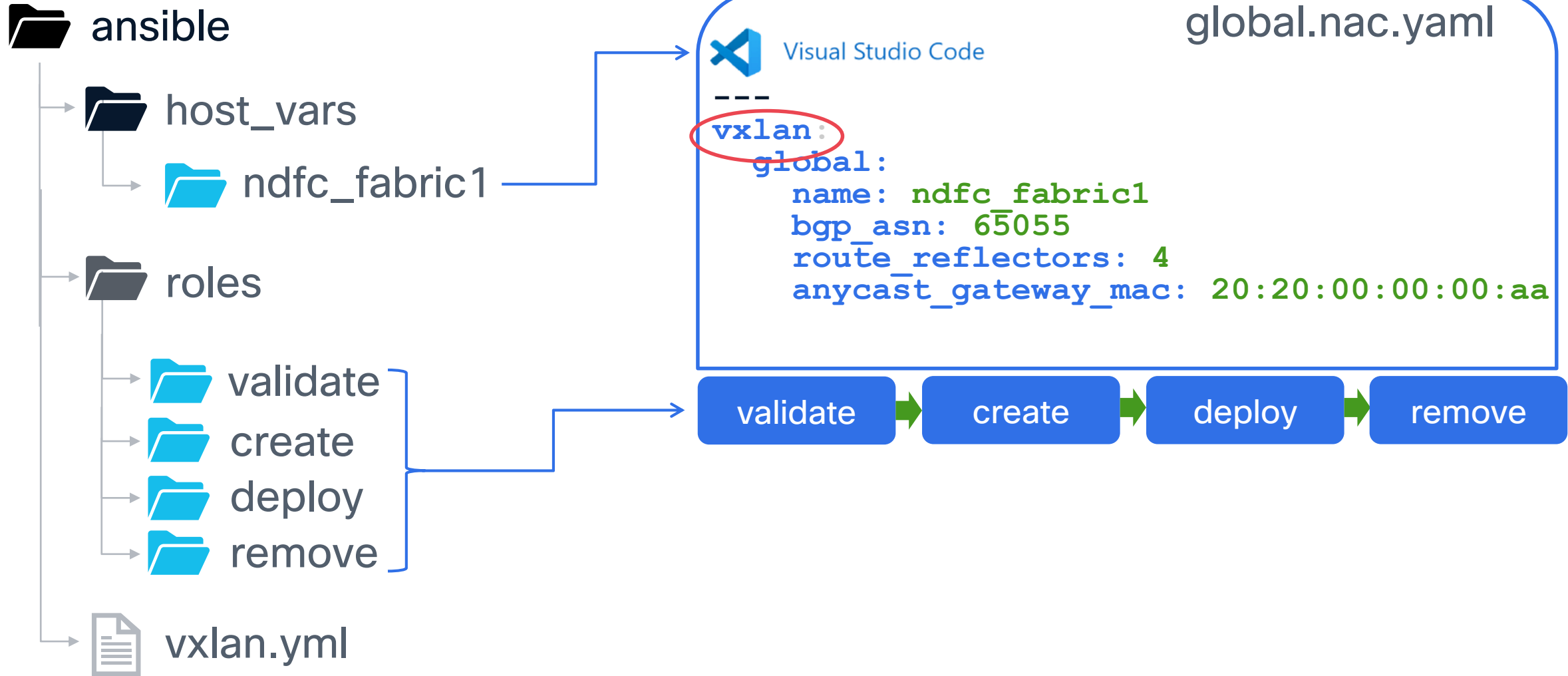
Workflow Step1: Fabric Management



Model - Fabric Management

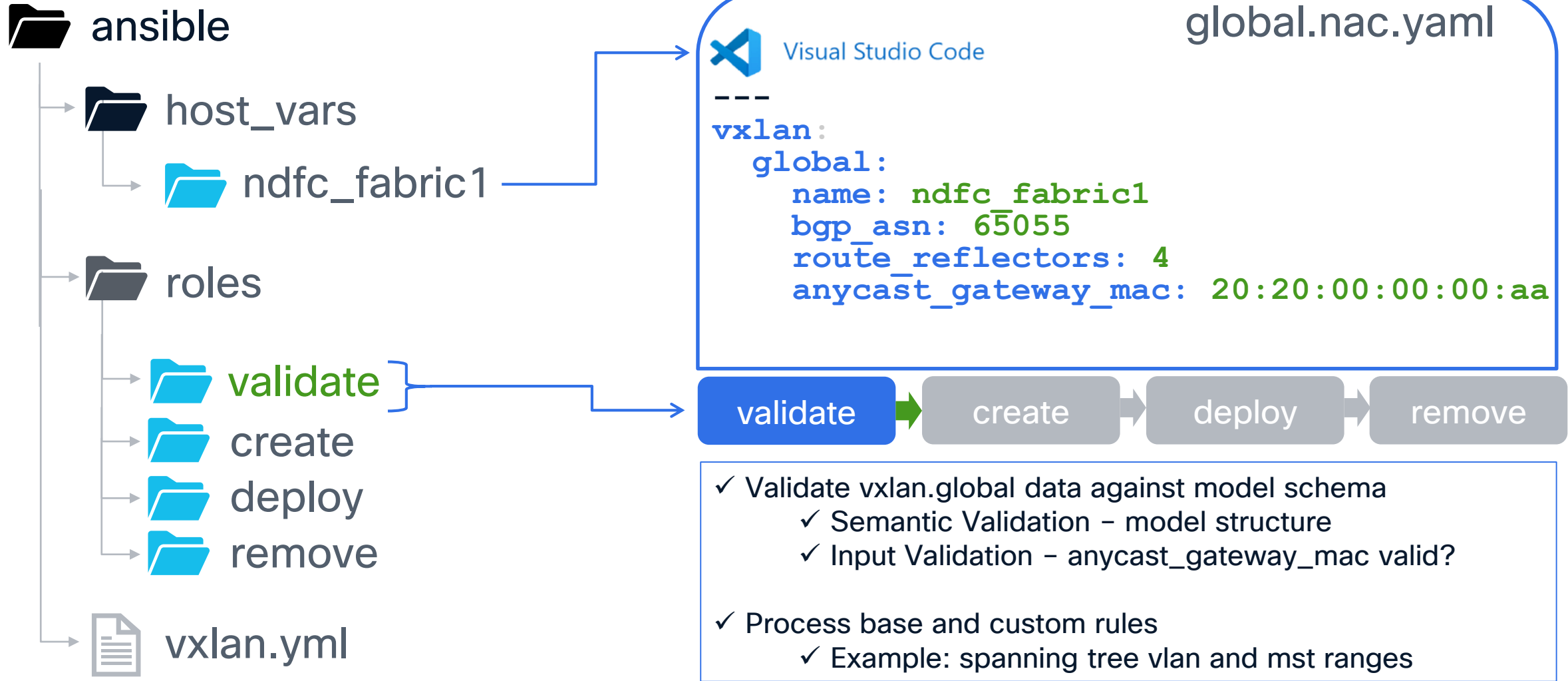
Data to do it

How to do it



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

ndfc_fabric1

roles

validate

create

deploy

remove

vxlan.yml



Visual Studio Code

global.nac.yaml

```
---  
vxlan:  
  global:  
    name: ndfc_fabric1  
    bgp_asn: 65055  
    route_reflectors: 4  
    anycast_gateway_mac: 20:20:00:00:00:aa
```

validate

create

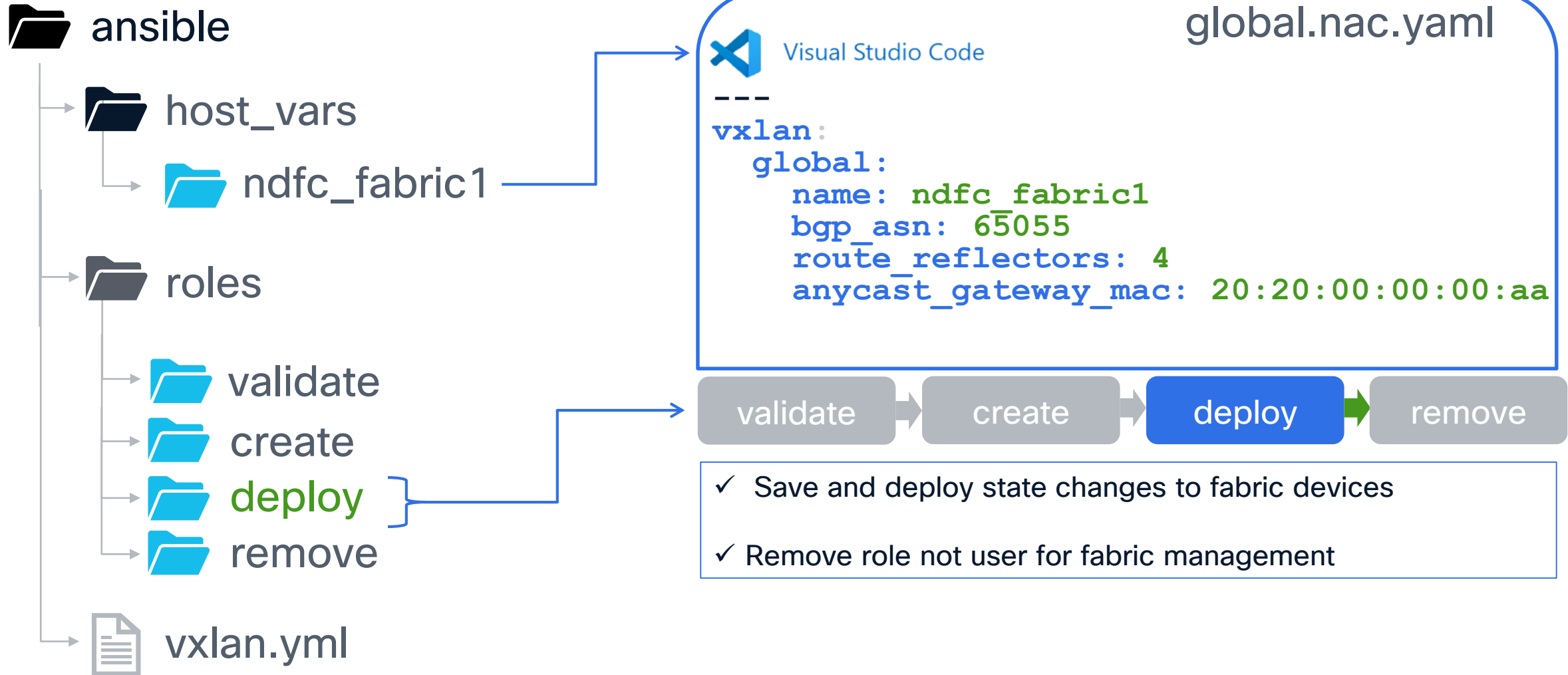
deploy

remove

- ✓ Transform model data for NDFC module processing
- ✓ Call cisco.dcnm.fabric module to create/manage fabric with model data settings.

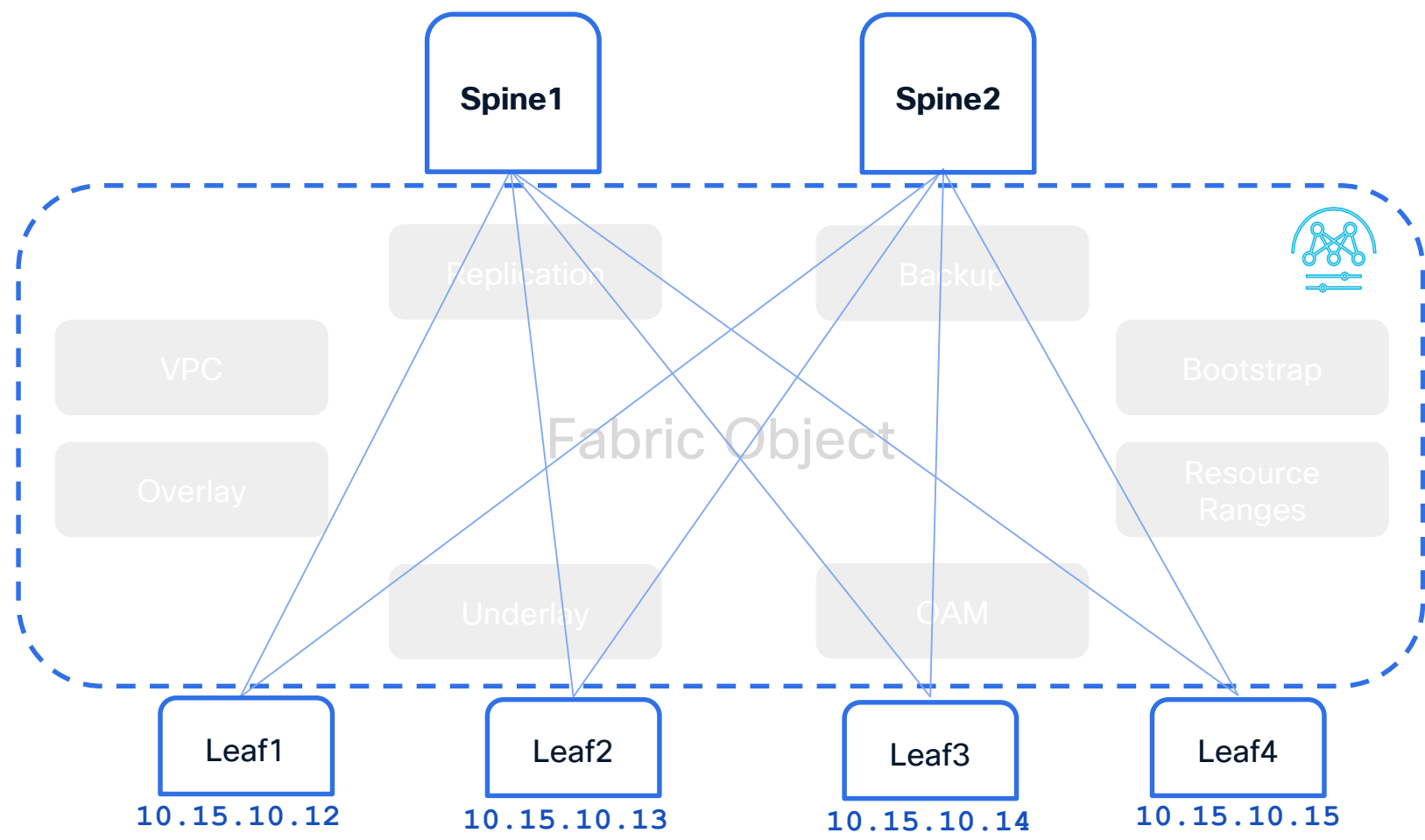
Model – Fabric Management

Data to do it
How to do it





Workflow Step2: Inventory Management





Workflow Step2: Inventory Management

Fabric Overview - fabric-stage Actions ⌵ 🔄 ? — ✕

Overview Switches Links Interfaces Interface Groups Policies Networks VRFs Services Event Analytics History Resources

Filter by attributes Actions ⌵

<input type="checkbox"/>	Switch	IP Address	Role	Serial Number	Mode	Config Status	Oper Status	Discovery Status	Model	VPC Role	VPC Peer
<input type="checkbox"/>	staging-leaf1	10.15.10.12	Leaf	9QRB2LQLD3H	● Normal	● In-Sync	⚠ Major	● OK	N9K-C9300v	Primary	staging-leaf2
<input type="checkbox"/>	staging-leaf2	10.15.10.13	Leaf	9CJC082GQEB	● Normal	● In-Sync	⚠ Major	● OK	N9K-C9300v	Secondary	staging-leaf3
<input type="checkbox"/>	staging-leaf3	10.15.10.14	Border	9OIZMJFK60Z	● Normal	● In-Sync	⚠ Minor	● OK	N9K-C9300v		
<input type="checkbox"/>	staging-spine1	10.15.10.11	Spine	9622UOESRG4	● Normal	● In-Sync	⚠ Minor	● OK	N9K-C9500v		

Add Switches

- Add Switches
- Preview
- Deploy
- Discovery >
- Set Role
- vPC Pairing

1

Add Switches - Fabric: fabric-stage

Switch Addition Mechanism*
 Discover Bootstrap Pre-provision

Seed Switch Details

Seed IP*

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

Authentication Protocol*

Username*

Max Hops*

Preserve Config

Unchecking this will clean up the configuration on switch(es)



Workflow Step2: Inventory Management

Overview **Switches** Links Interfaces

Filter by attributes

<input type="checkbox"/> Switch	IP Address
<input type="checkbox"/> staging-leaf2	10.15.7.13
<input type="checkbox"/> staging-spine1	10.15.7.11

1

Actions ^

- Add Switches
- Preview
- Deploy
- Discovery >
- Set Role
- vPC Pairing
- ToR/Access Pairing
- vPC Overview
- More >

2

Switch Addition Mechanism*
Discover

Seed Switch Details

Seed IP*
10.15.7.12

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

Authentication Protocol*
MD5

Username*
admin

Max Hops*
2

Preserve Config

Unchecking this will clean up the configuration on switch(es)

Password*

Set as individual device write credential

3

4

Close Add Switches

Fabric Overview - fabric-stage

Overview **Switches** Links Interfaces Interface Groups Policies Networks

Filter by attributes

<input type="checkbox"/> Switch	IP Address	Role	Serial Number	Mode
<input type="checkbox"/> staging-leaf1	10.15.7.12	Leaf	90VV2DNDLFJ	Migration

5

Select Role

Search Role

- Spine
- Leaf (current)
- Border
- Border Spine
- Border Gateway
- Border Gateway Spine
- Super Spine
- Border Super Spine
- Border Gateway Super Spine
- ToR

6

7

Actions ^

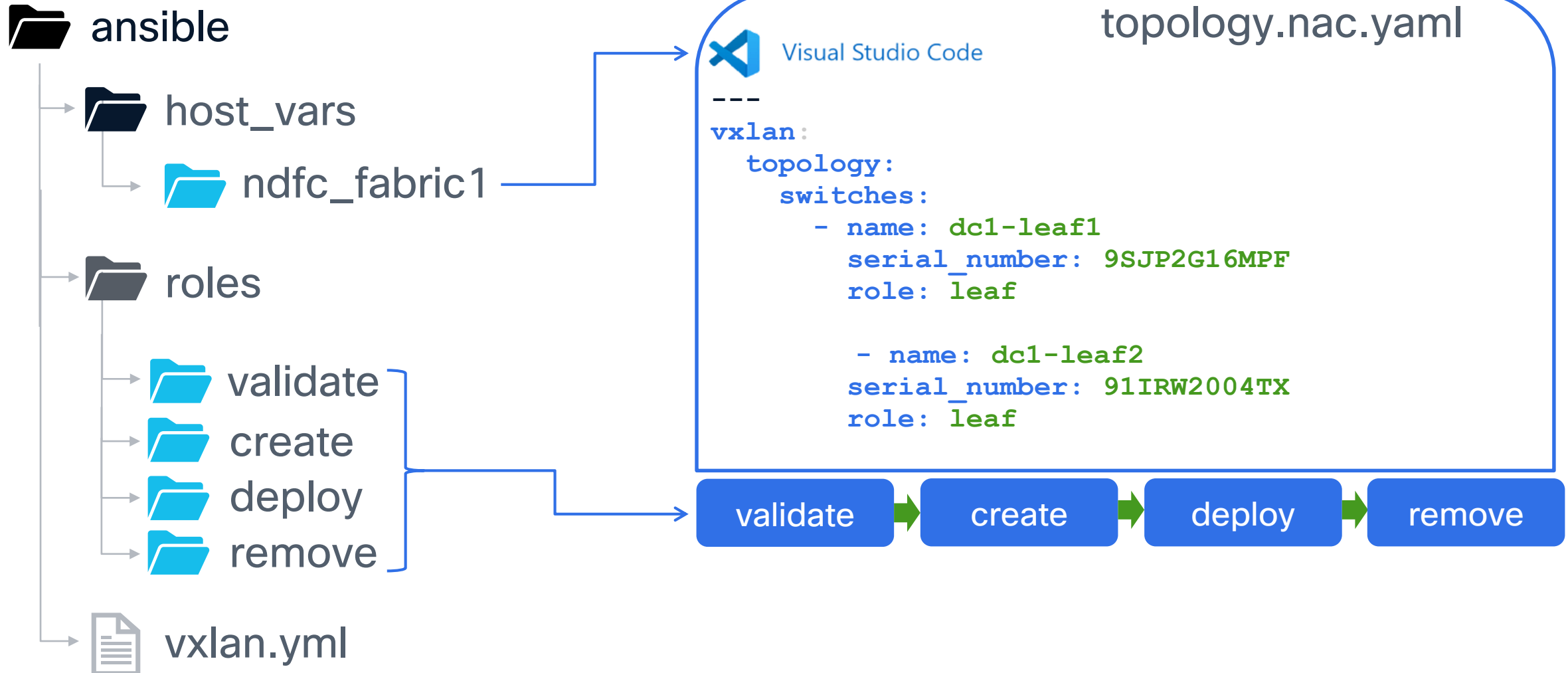
- Edit Fabric
- Add Switches
- Recalculate and Deploy
- More >

Actions v

Model – Inventory Management

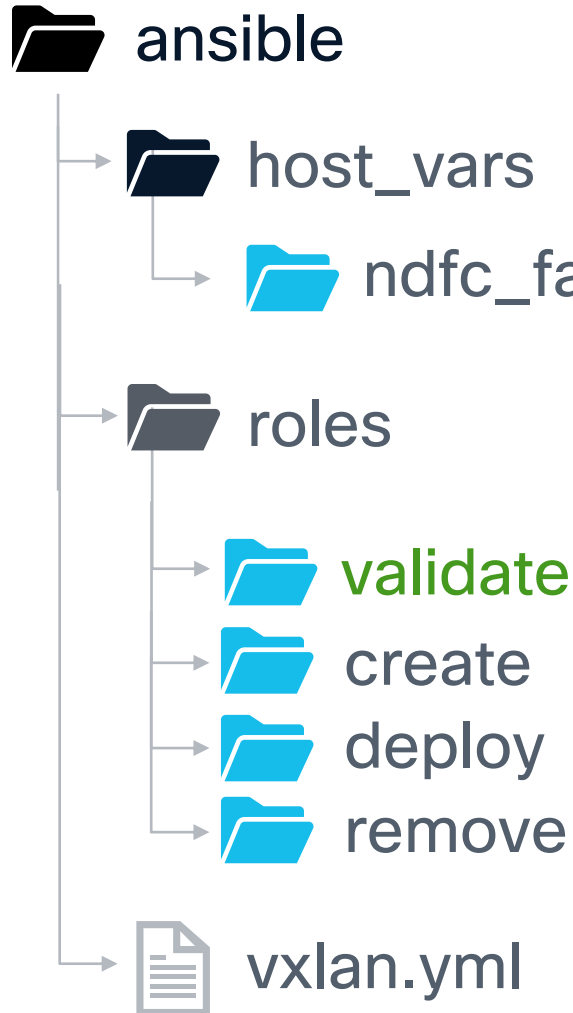
Data to do it

How to do it



Model - Inventory Management

Data to do it
How to do it



Visual Studio Code

topology.nac.yaml

```
---
vxlan:
  topology:
    switches:
      - name: dc1-leaf1
        serial_number: 9SJP2G16MPF
        role: leaf

      - name: dc1-leaf2
        serial_number: 91IRW2004TX
        role: leaf
```



- ✓ Validate vxlan.topology.switches data against model schema
- ✓ Model structure and input validation
- ✓ Process base and custom rules
 - ✓ Example: Serial Number must be included

Model – Inventory Management

Data to do it

How to do it

ansible

host_vars

ndfc_fabric1

roles

validate

create

deploy

remove

vxlan.yml

Visual Studio Code

topology.nac.yaml

```
---
vxlan:
  topology:
    switches:
      - name: dc1-leaf1
        serial_number: 9SJP2G16MPF
        role: leaf

      - name: dc1-leaf2
        serial_number: 91IRW2004TX
        role: leaf
```

validate

create

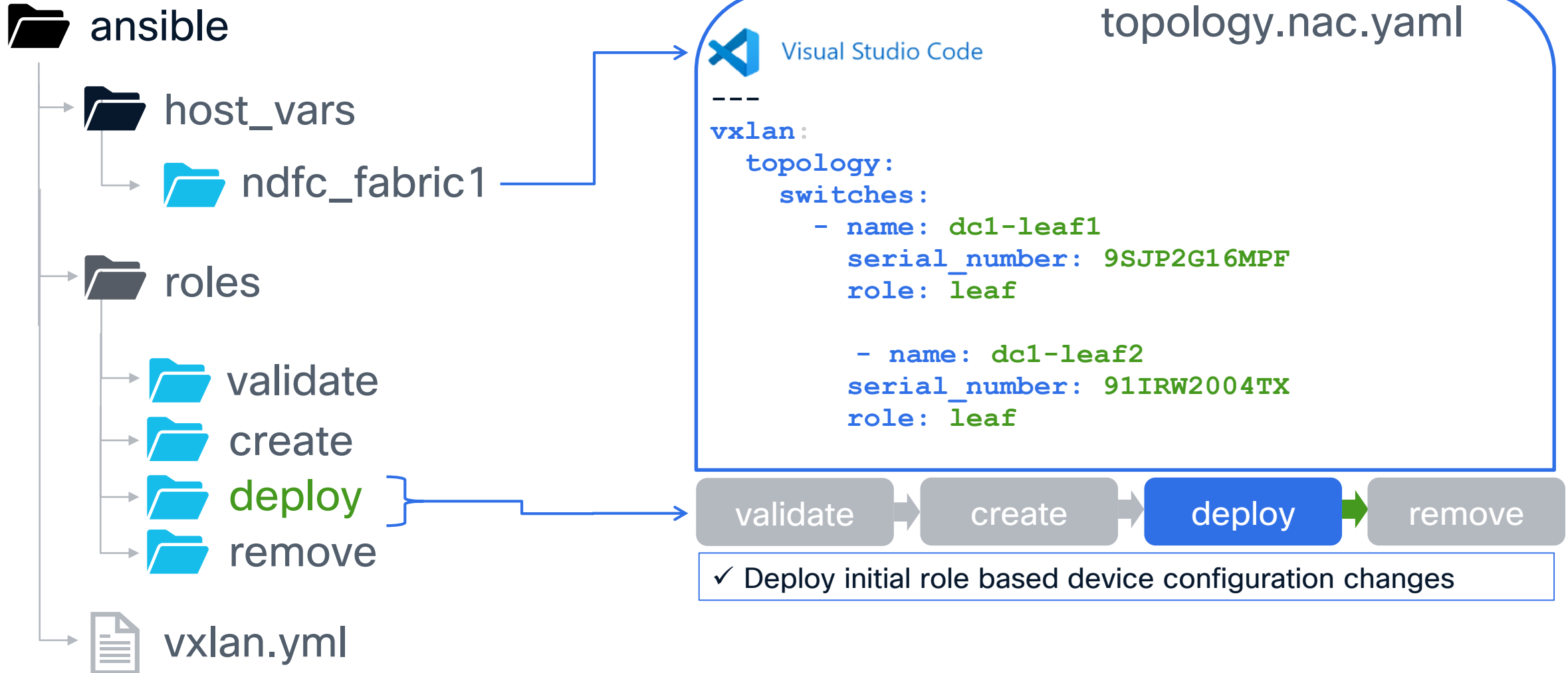
deploy

remove

- ✓ Add devices to the fabric but don't deploy changes
- ✓ Set Role and other management properties
- ✓ Devices remain in pending state until deploy role executed

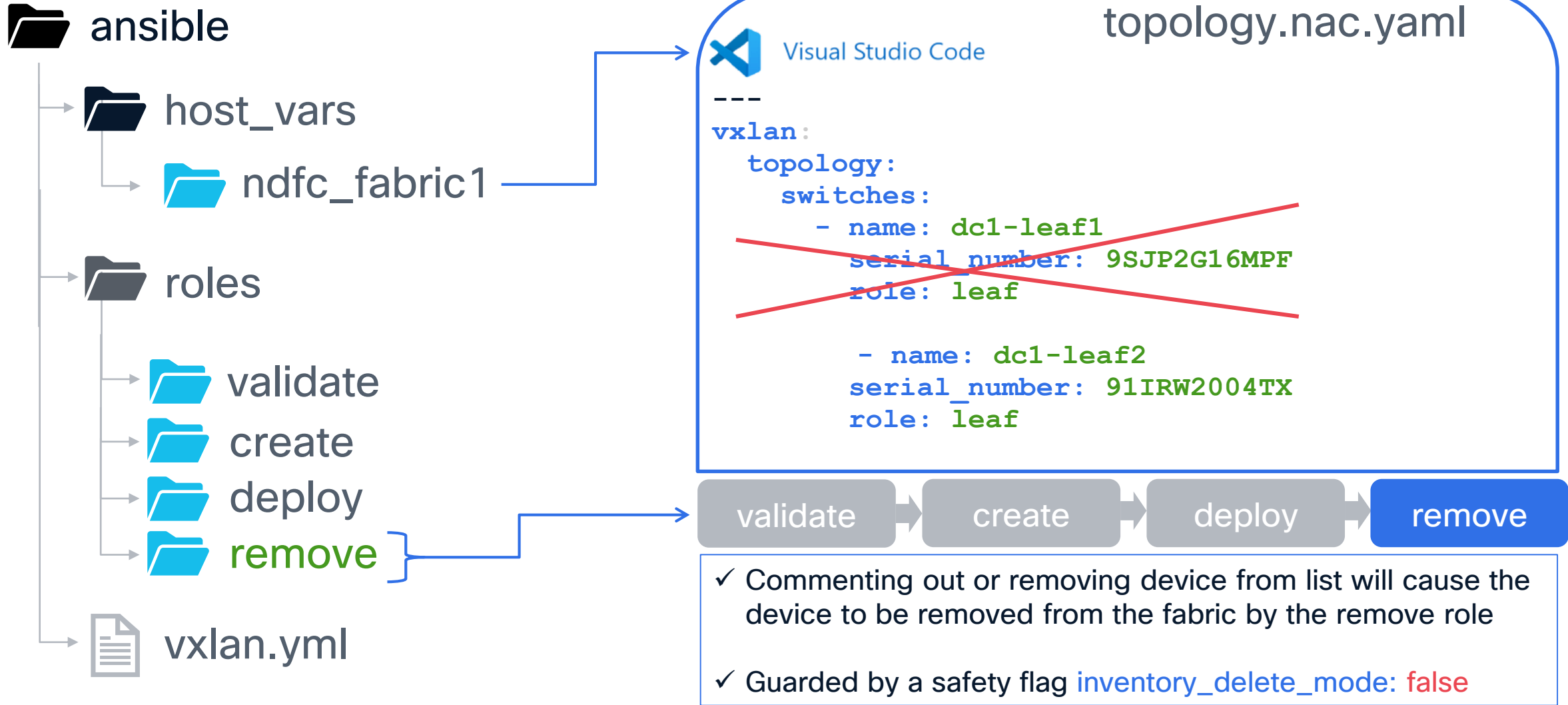
Model - Inventory Management

Data to do it
How to do it



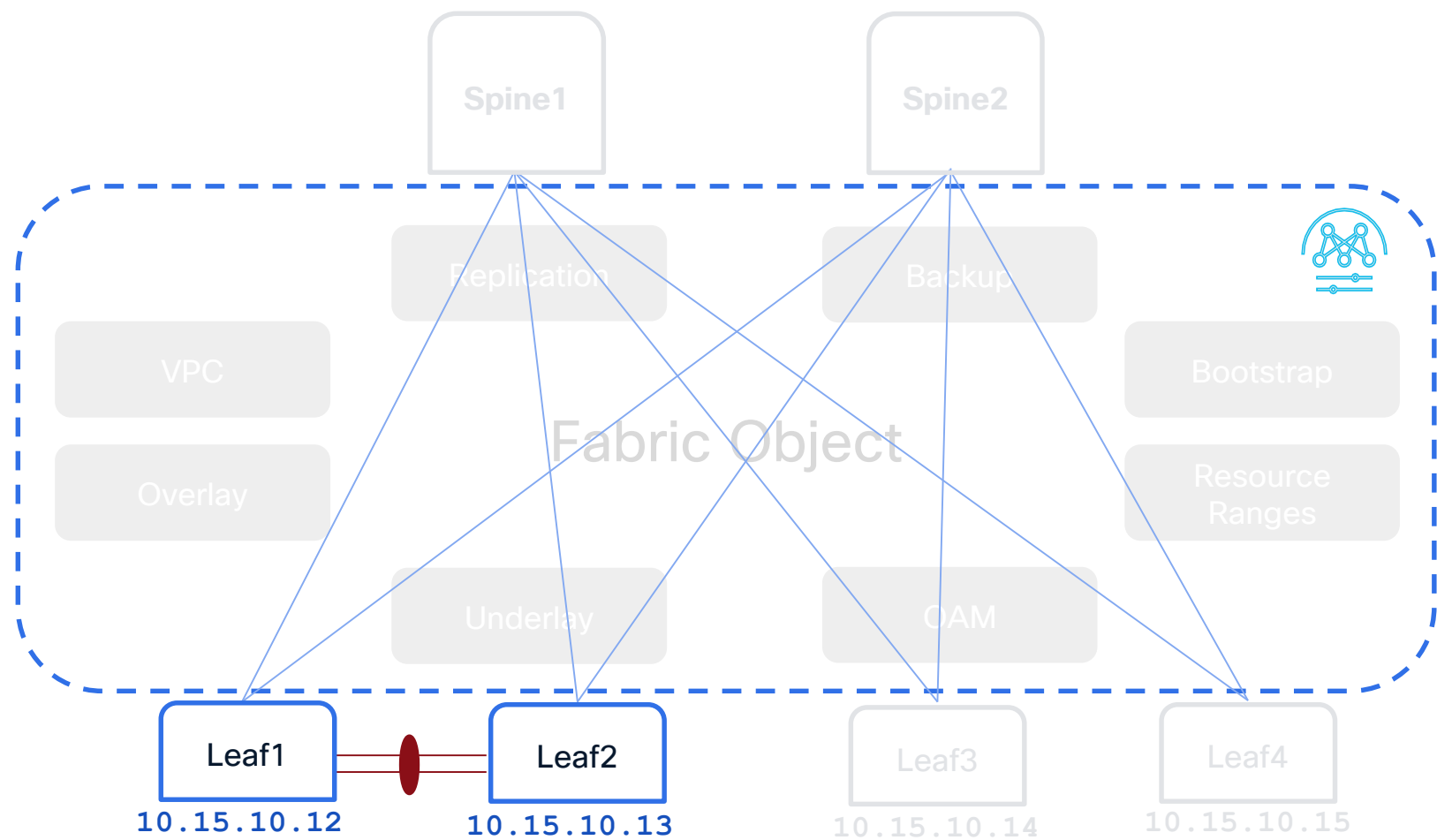
Model – Inventory Management

Data to do it
How to do it





Workflow Step3: vPC Pair Management





Workflow Step3: vPC Pair Management

Fabric Overview - fabric-stage

Actions

Overview Switches Links Interfaces Interface Groups Policies Networks VRFs Services Event Analytics History Resources

Filter by attributes

Actions ^

Switch	IP Address	Role	Serial Number	Mode	Config Status	Oper Status	Discovery Status	Model	VPC Role	VPC Peer
<input checked="" type="checkbox"/> staging-leaf1	10.15.7.12	Leaf	90VV2DNDLFJ	Normal	In-Sync	Minor	Ok	N9K-C9300v		
<input type="checkbox"/> staging-leaf2	10.15.7.13	Leaf	9VH4J9YPUMV	Normal	In-Sync	Minor	Ok	N9K-C9300v		
<input type="checkbox"/> staging-spine1	10.15.7.11	Spine	96IWSX5RHVM	Normal	In-Sync	Minor	Ok	N9K-C9500v		

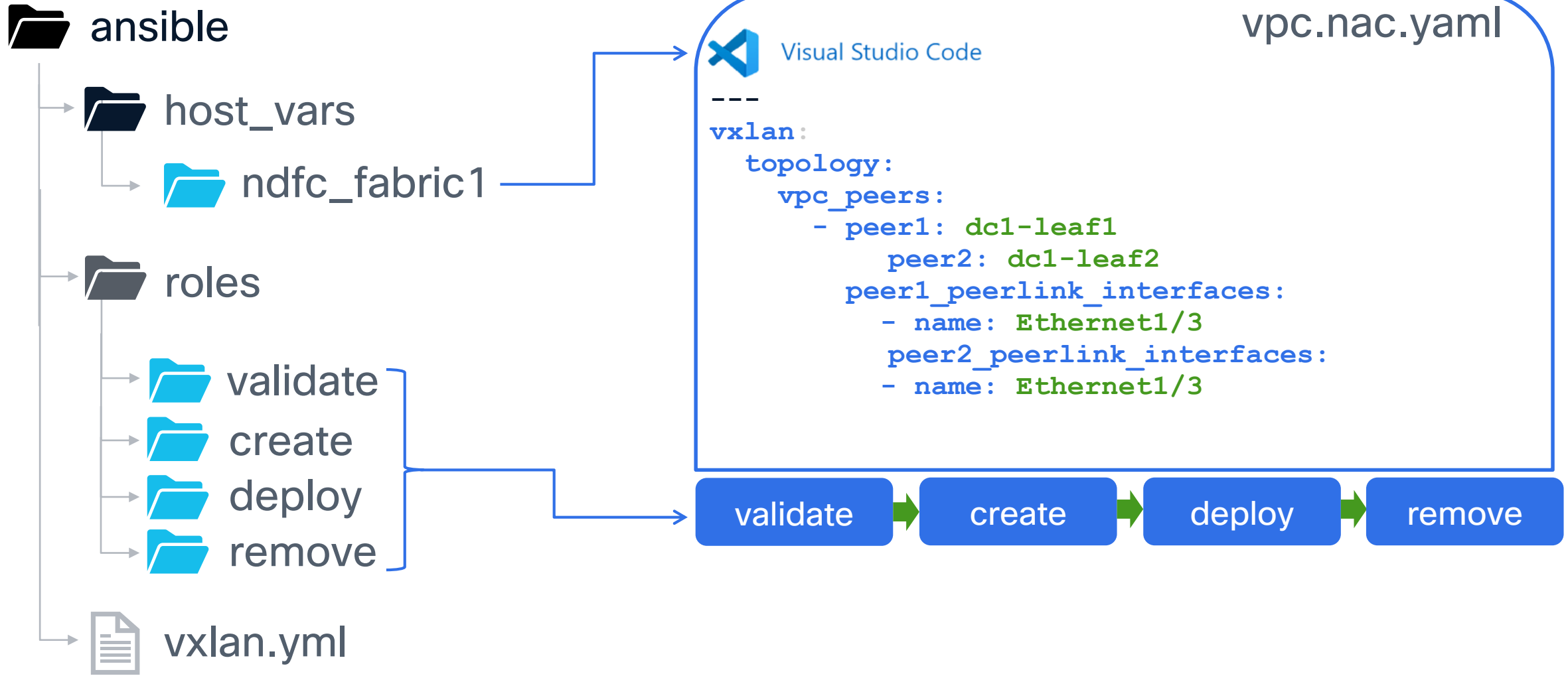
- Add Switches
- Preview
- Deploy
- Discovery >
- Set Role
- vPC Pairing**
- ToR/Access Pairing
- vPC Overview
- More >



Model - vPC Pair Management

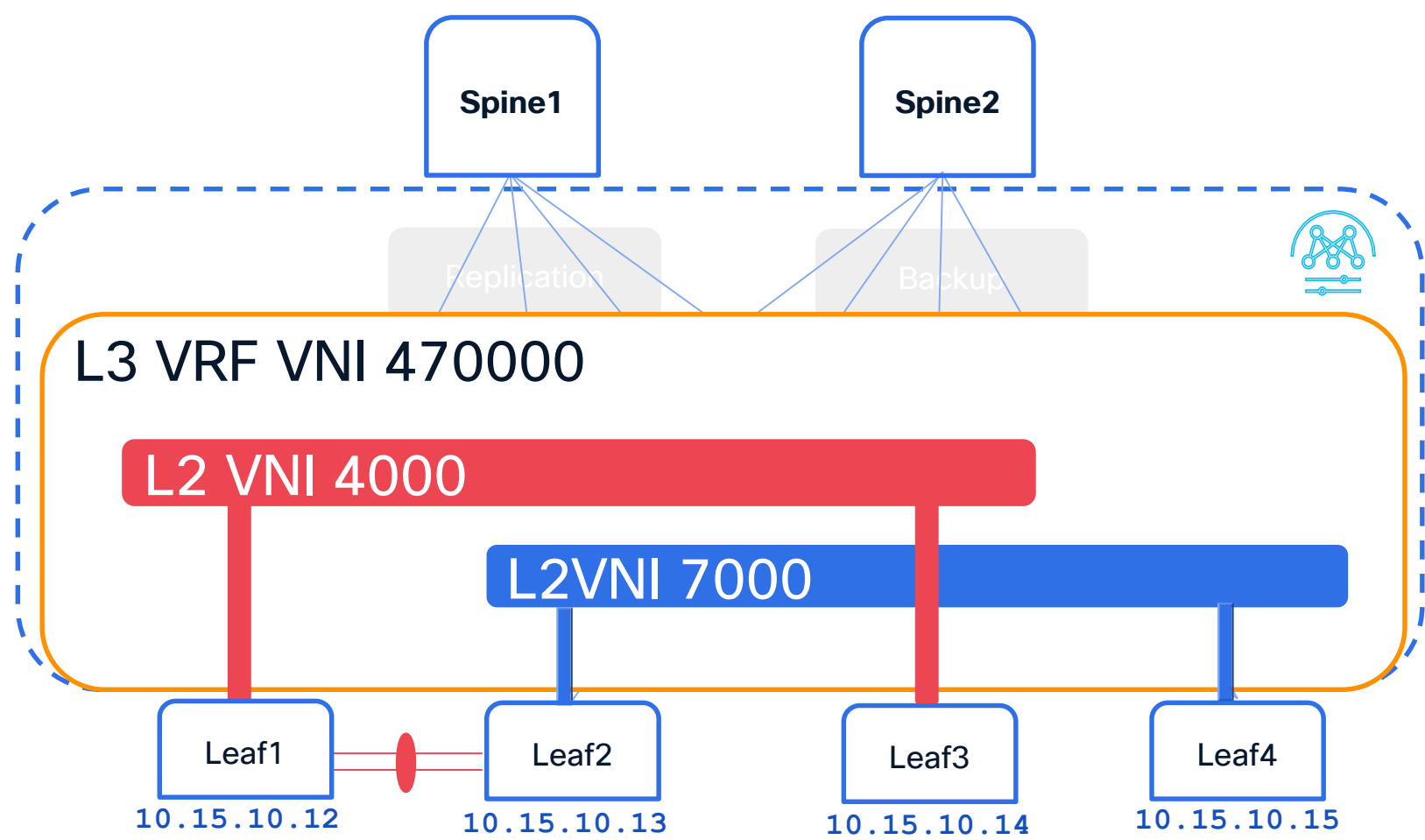
Data to do it

How to do it





Workflow Step4: Overlay Management





Workflow Step3: Overlay Management

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	VRF ID	VLAN ID	Status
CL-VRF1	470000	2055	DEPLOYED

Fabric Name	VRF Template	VRF Extension Template	VRF Description
fabric-stage	Default_VRF_Universal	Default_VRF_Extension_Universal	NA

L3VniMcastGroup
NA

VRF Status

3
Status

DEPLOYED 3

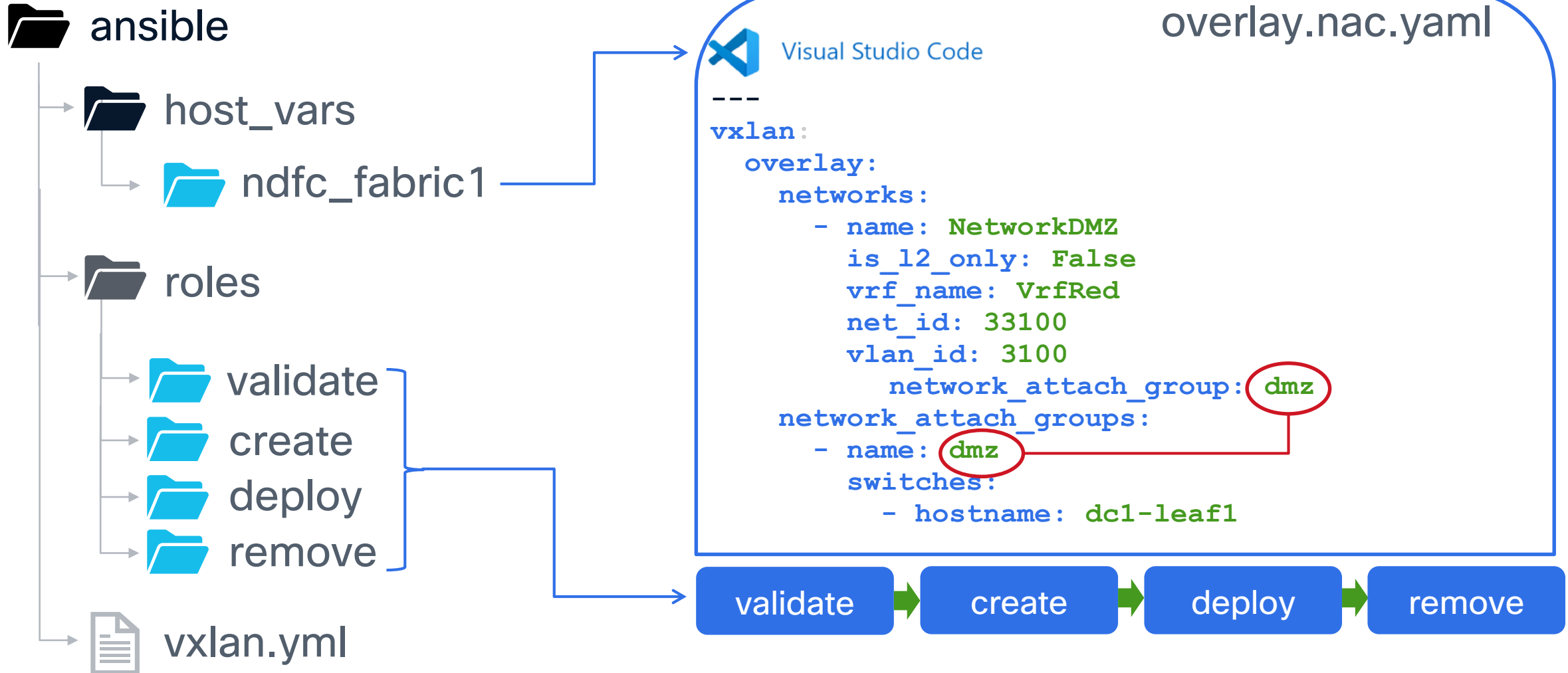
Attached Roles Association

3
Role

- border 1
- leaf 2

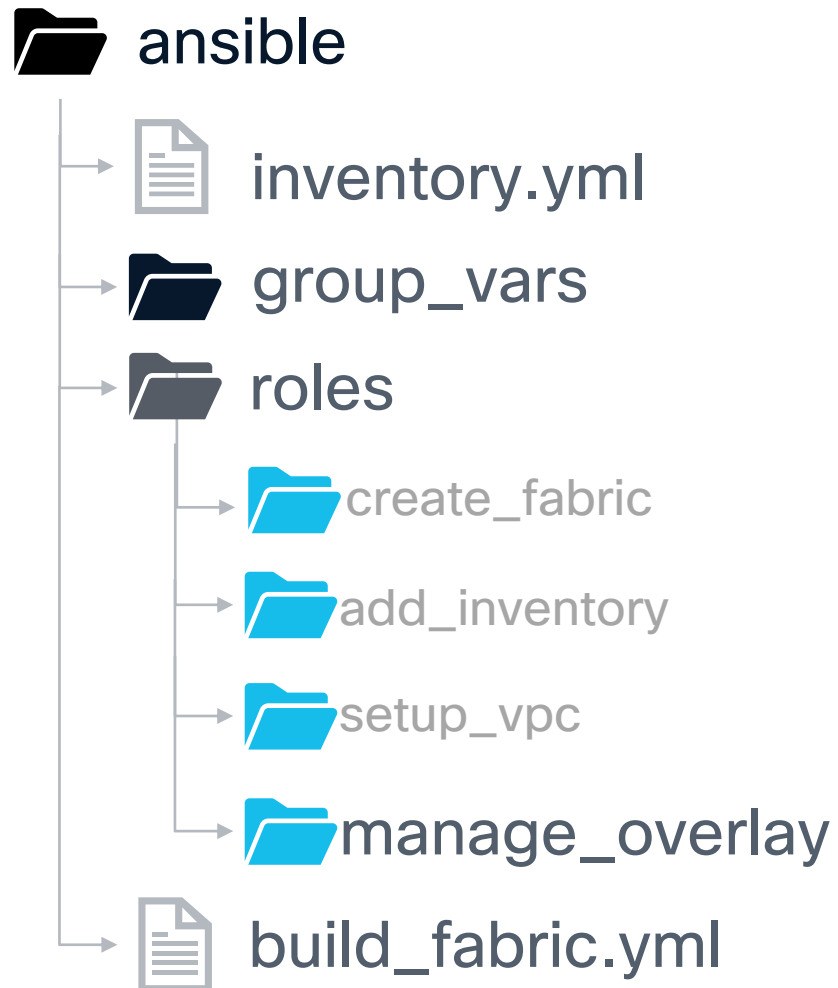
Model - Overlay Management

Data to do it
How to do it



Ansible Roles - VRF / Net Tasks

How to do it



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

- name: Add Overlay VRFs
  cisco.dcnm.dcnm_vrf:
    fabric: "{{ fabric.name }}"
    state: replaced
    config:
      - vrf_name: CL-VRF1
        vrf_id: 470000
        vlan_id: 2055
        attach:
          - 192.168.1.1
          - 192.168.1.2
          - 192.168.1.3
          - 192.168.1.4

- name: Add Overlay Networks
  cisco.dcnm.dcnm_network:
    fabric: "{{ fabric.name }}"
    state: overridden
    config:
      - net_name: CL-NET7000
        vrf_name: CL-VRF1
        net_id: 7000
        vlan_id: 88
        attach:
          - 192.168.1.2
          - 192.168.1.4
```

- Creates VRF and Network objects
- Attach and deploy VRF and Network config to fabric leaf devices
- Deploy Control

Validate Role – Deeper Dive

Example of YAMALE Schema Validation

- Invalid anycast_gateway_mac
- Early detection before sending any requests to the fabric

```
---  
vxlan:  
  global:  
    name: nac-ndfc1  
    fabric type: VXLAN_EVPN  
    bgp asn: 65001  
    route reflectors: 2  
    anycast_gateway_mac: 12:34:56:78:90:00E99E
```

Validate Role – Deeper Dive

Example of YAMALE Schema Validation

- Invalid anycast_gateway_mac
- Early detection before sending any requests to the fabric

```
---
vxlan:
  global:
    name: nac-ndfc1
    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"

Validate Role – Deeper Dive

- 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
```

Validate Role – Deeper Dive

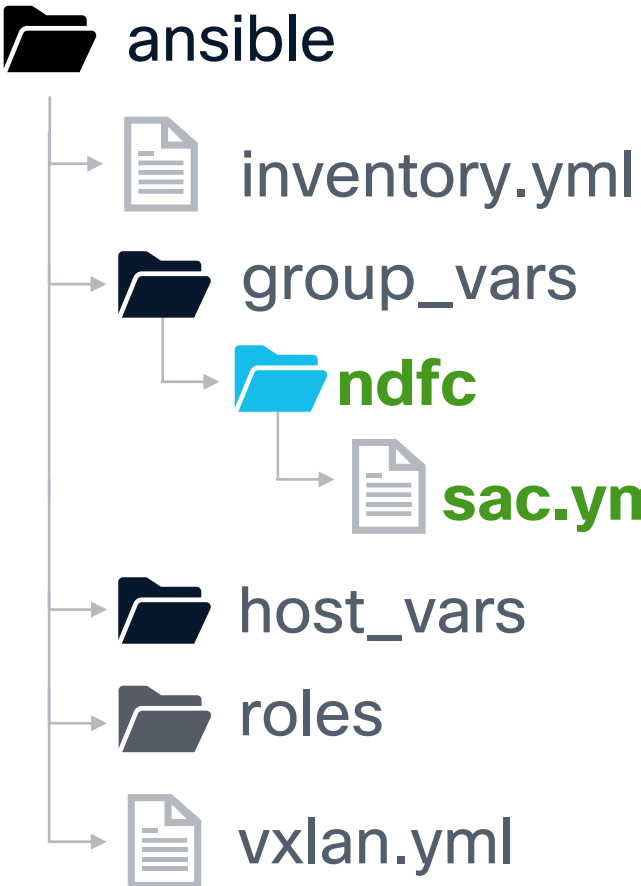
- Network referencing an undefined VRF
- Early detection before sending any requests to the fabric

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

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

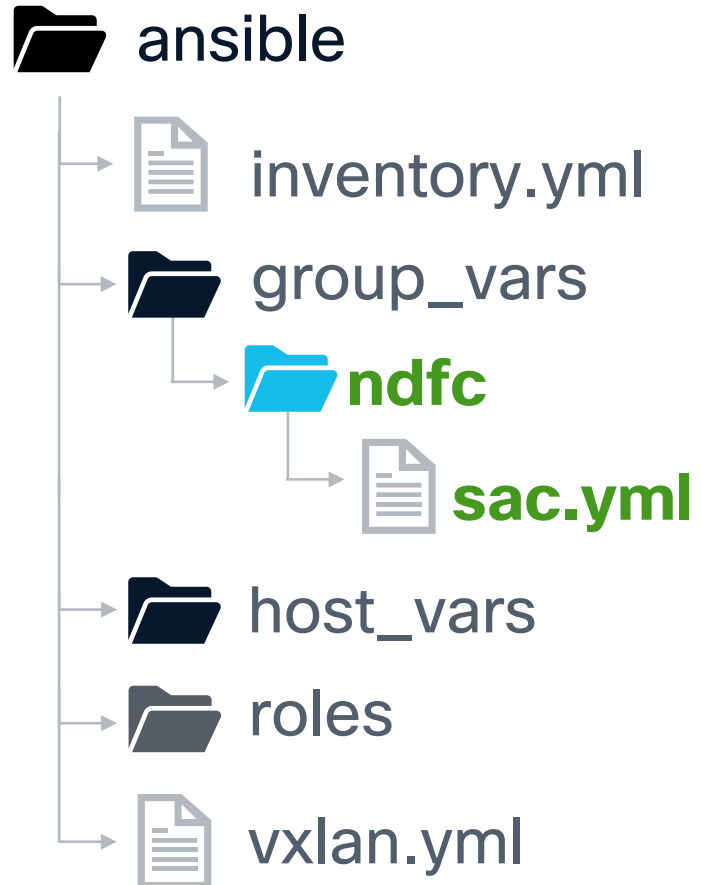
"msg": "Semantic error, rule 401: Cross Reference VRFs and Networks items in the Service Model (['Network (NaC-Net01) is referencing VRF (NaC-VRF01) which is not defined in the service model. Add the VRF to the service model or remove the network from the service model and re-run the playbook.]"

Ansible Group Vars - Schema and Custom Rules



```
enhanced_rules_path: "{{ lookup('ansible.builtin.env', 'DC_VXLAN_RULES') }}"
```

Ansible Group Vars - Schema and Custom Rules



```
# 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

NDFC as Code – Ansible Tags

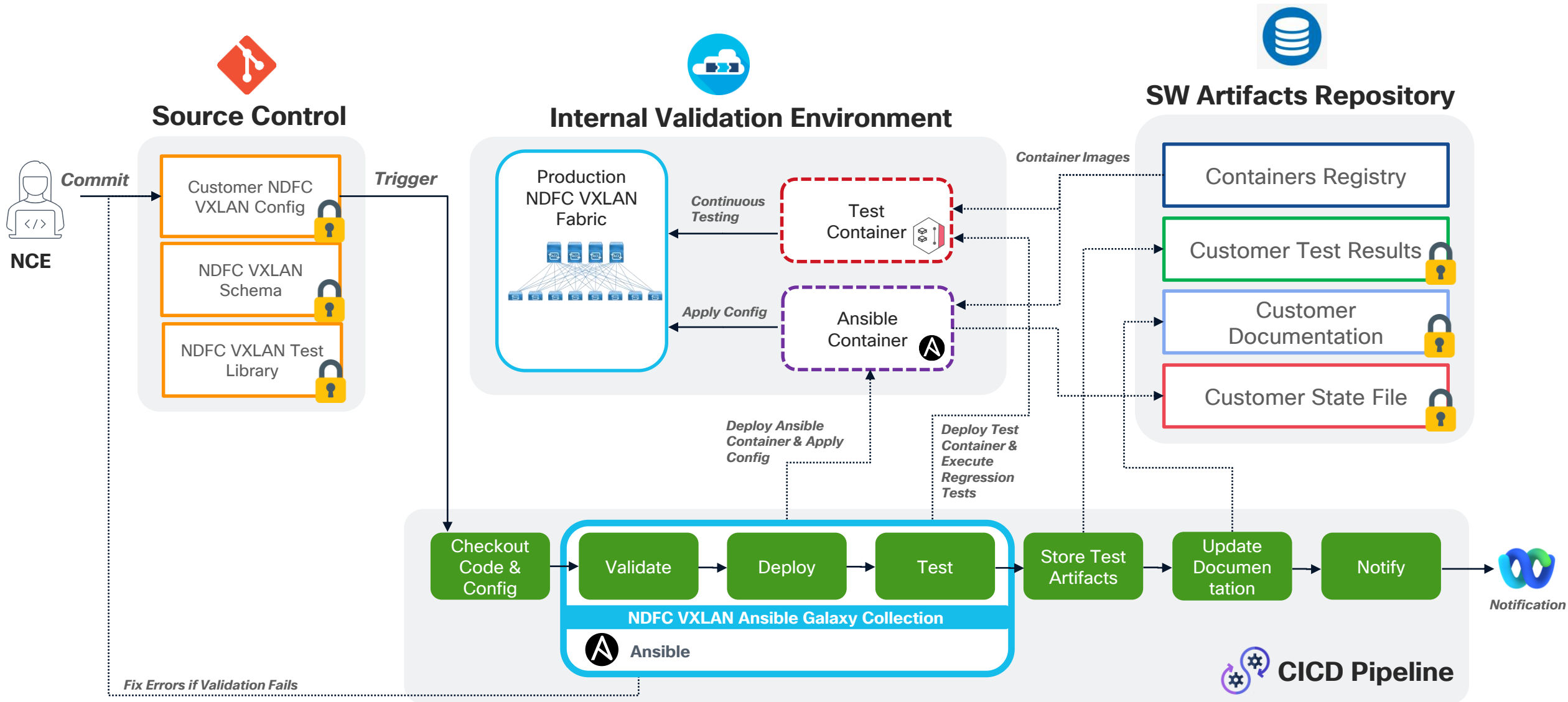
Tags for selective execution control

```
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

Standard NDFC new customer configuration pipeline



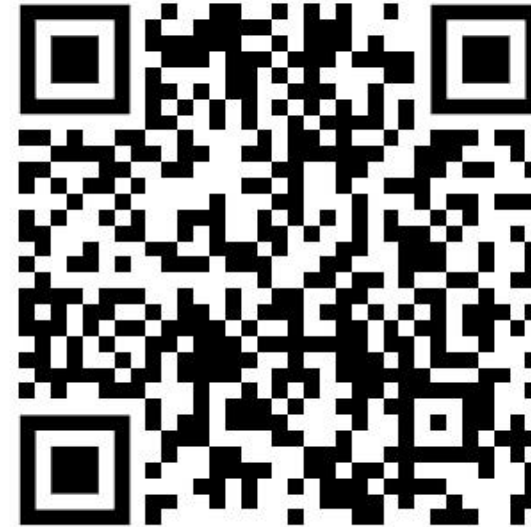
References to Start Your Journey

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 NDFC Repos



BRKDCN-2946
Session Repo

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

Complete your session evaluations



Complete a minimum of 4 session surveys and the Overall Event Survey to be entered in a drawing to win 1 of 5 full conference passes to Cisco Live 2026.



Earn 100 points per survey completed and compete on the Cisco Live Challenge leaderboard.



Level up and earn exclusive prizes!



Complete your surveys in the Cisco Live mobile app.

Continue your education



Visit the Cisco Showcase for related demos



Book your one-on-one Meet the Engineer meeting



Attend the interactive education with DevNet, 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 !

