
cisco Live!
Let's go

The bridge to possible

Infrastructure as Code for NXOS and NDFC with Ansible

Matt Tarkington, Technical Leader
Mike Wiebe, Technical Leader

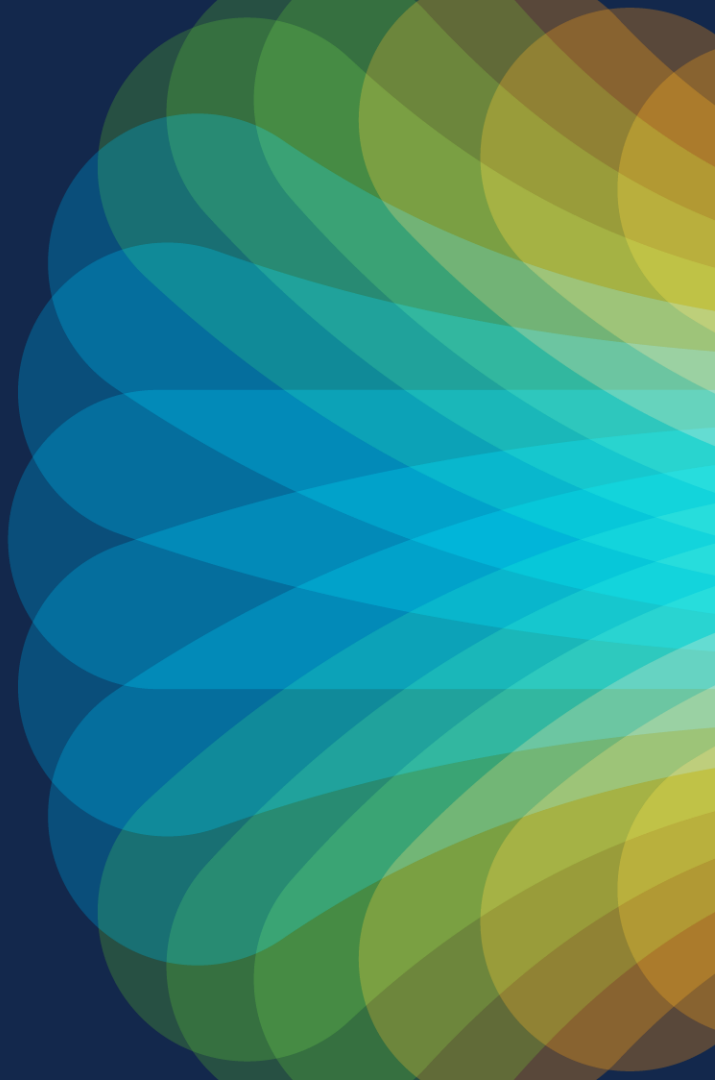
cisco Live!

BRKDCN-2946

Agenda

- What is Infrastructure as Code?
- Infrastructure as Code with NXOS and Ansible
- Infrastructure as Code with NDFC and Ansible
- 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 programing language
- The intended configuration state of network devices are sourced from source code management (git) instead of the devices themselves

Infrastructure as Code Tools

SCM

- Git
- GitHub
- **GitLab**
- Bitbucket

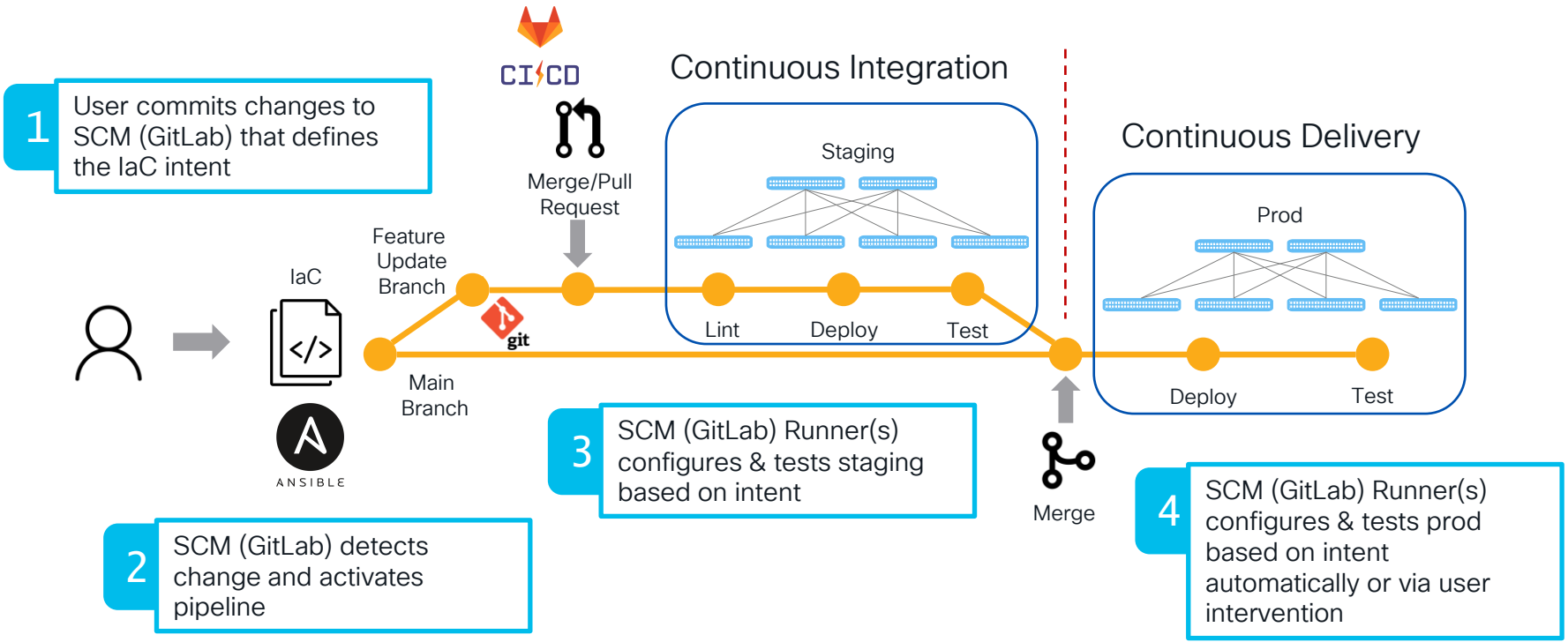
Pipeline

- Jenkins
- **GitLab CI**
- CircleCI
- Travis CI
- GitHub Actions

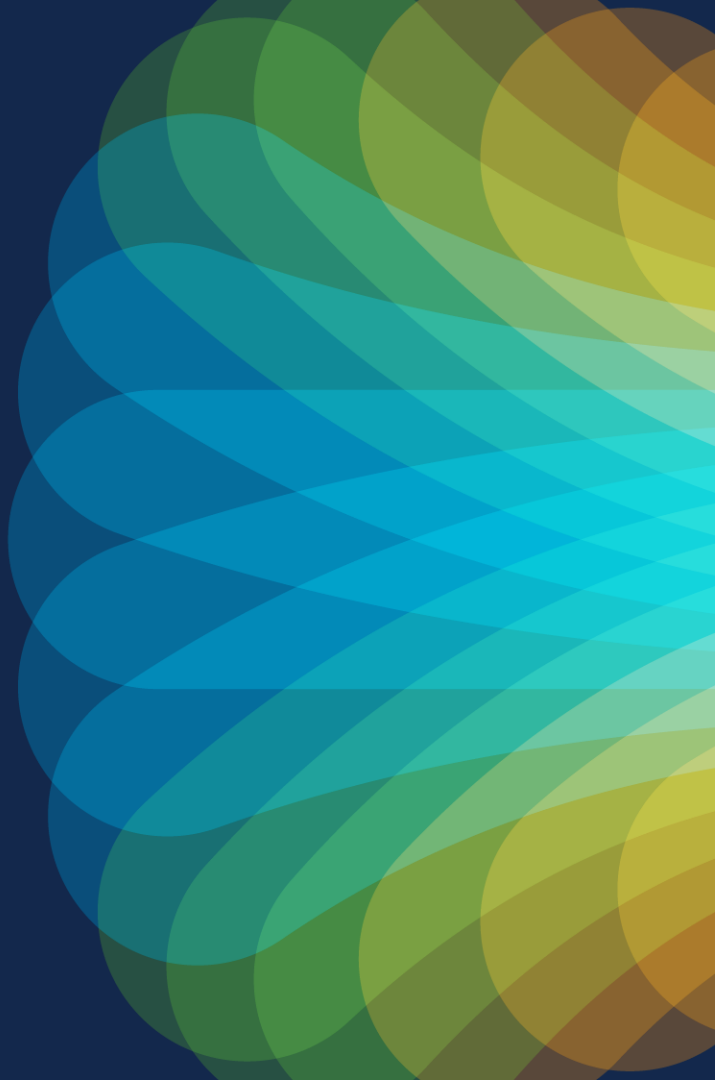
Execution Software

- **Ansible**
- Terraform
- Chef
- Puppet
- AWS CloudFormation
- Google Cloud Deployment Manager
- Vagrant
- Saltstack

Infrastructure as Code for Network Ops



Infrastructure as Code with NXOS and Ansible



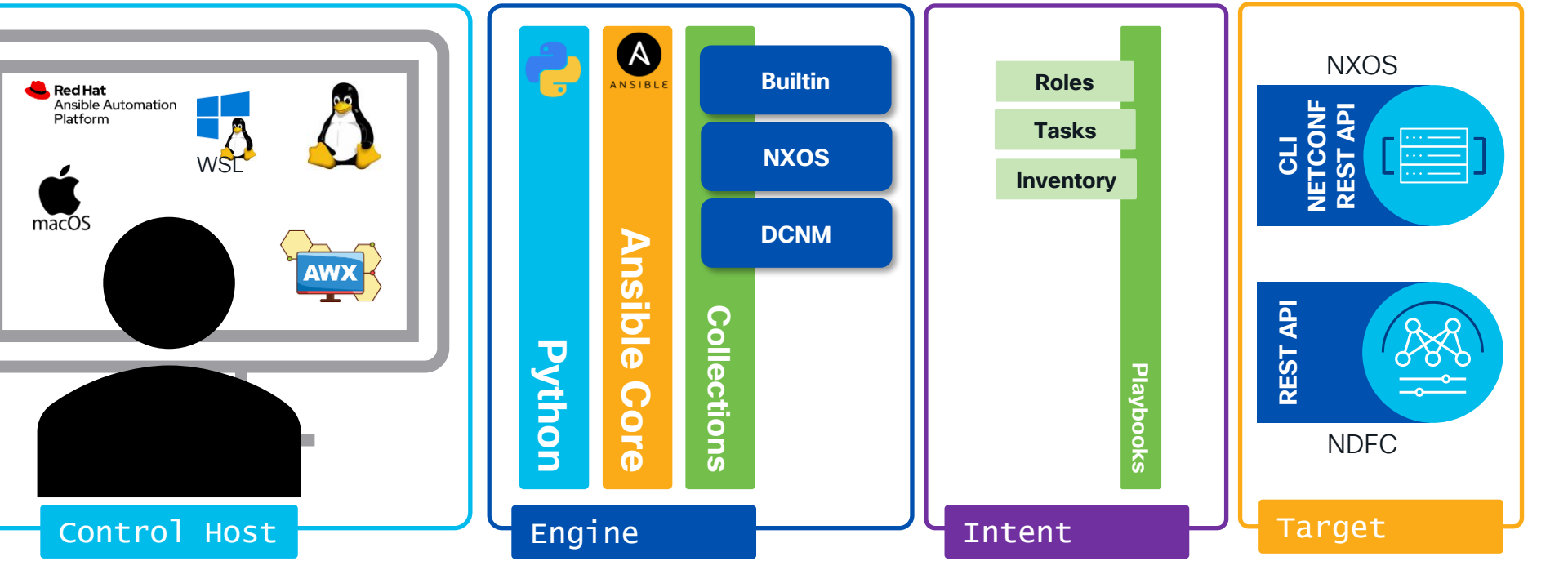
cisco Live!

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?



Python Virtual Environments

- You should use a **virtual environment**
- Allows for installing Ansible inside a contained area with specific version of Python
- Makes it possible to run different Python scripts that require different versions of Python and libraries
- Detailed steps beyond scope of this session

pyenv

- pyenv is the best mechanism to control python virtual environments
- Allows control of python version to execute independent of system version
- pyenv virtualenv also needed

Install instructions:

https://github.com/pyenv/pyenv/wiki
https://github.com/pyenv/pyenv-virtualenv

1 install a version of python

```
% pyenv install 3.9.11
```

2 create virtual environment

```
% pyenv virtualenv 3.9.11 ansible
```

3 create directory for ansible development

```
% mkdir my_ansible_dir
```

4 Set pyenv virtual environment

```
% pyenv local 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 cisco.dcnm
```

NXOS - https://galaxy.ansible.com/cisco/nxos NDFC - https://galaxy.ansible.com/cisco/dcnm

Ansible.Builtin

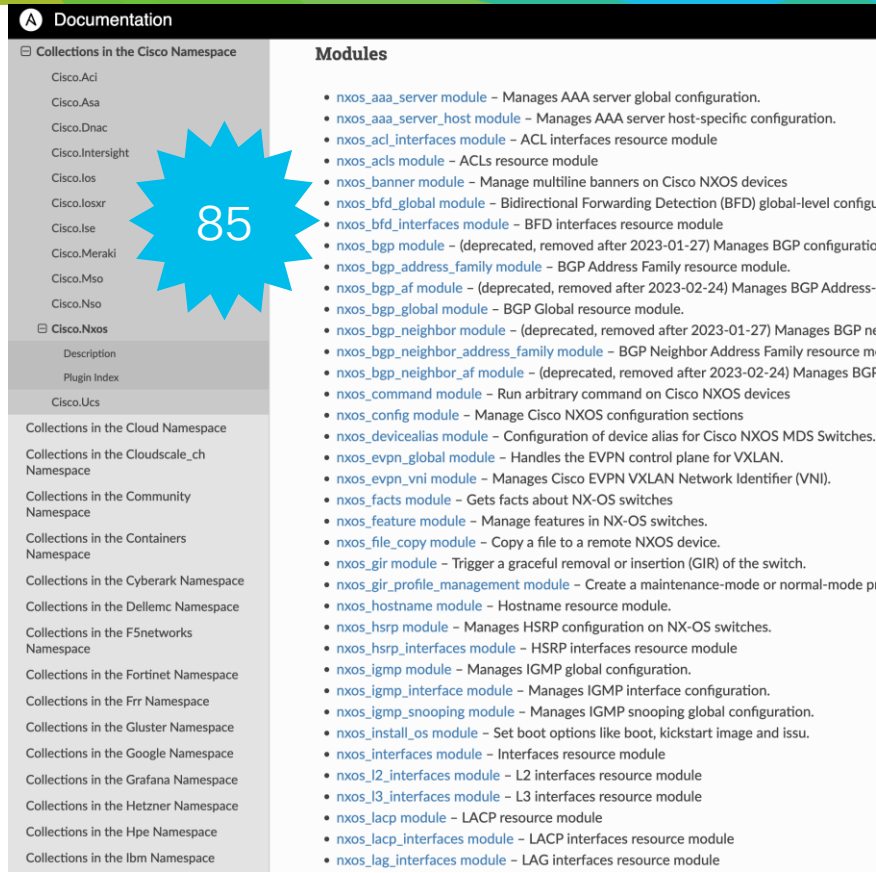
- Modules perform specific task like set facts (variables), import roles, tasks, vars, and more
- Includes many filters for working with data sets
- Actively maintained by RedHat

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/index.html

Ansible Cisco.Nxos

Collection Modules

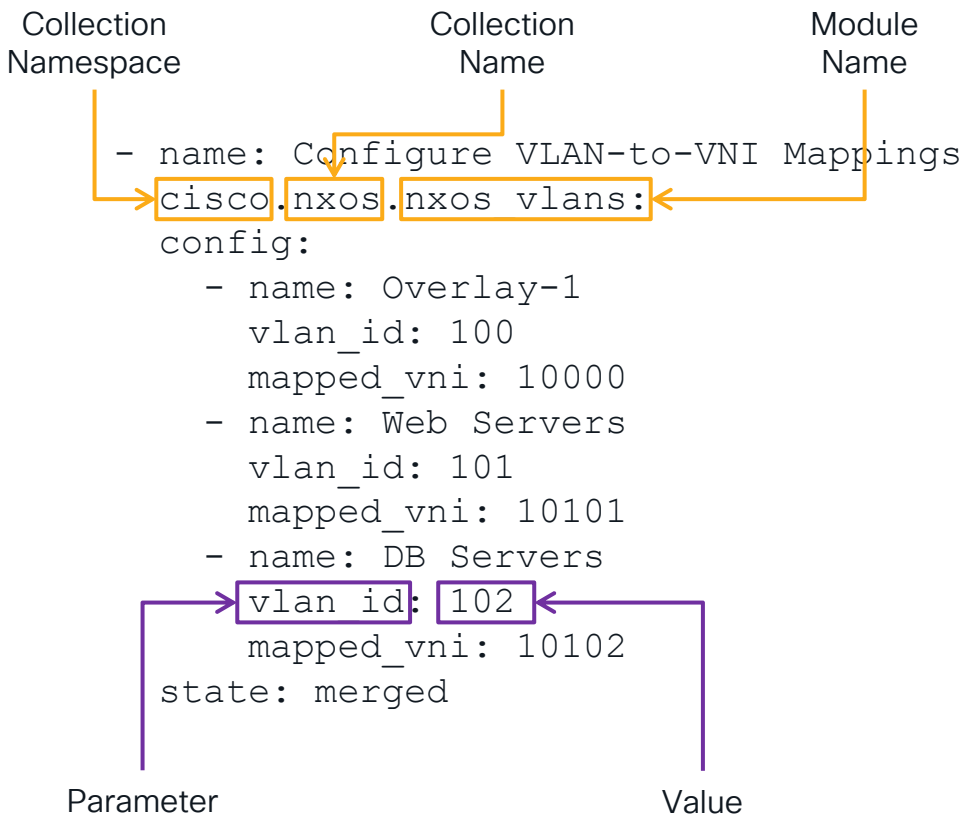
- Modules perform specific task like configure vlans, interfaces, OSPF, BGP, and more
- Documentation provides usage details, required variables, default variables, etc
- Actively maintained by RedHat with Cisco support



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



A word about YAML Syntax

YAML Ain't Markup Language

- Human Readable Data Structures
  - Lists, Dictionaries, etc
- Used in inventory, playbooks, & variable files
- Best practice is to use:
  - Text editor (e.g. Notepad++)
  - IDE (e.g. VSCode) with language assistant support for YAML
  - Indentation is very important, and the proper editor will help you


Microsoft VSCode

ATOM



PyCharm

Eclipse

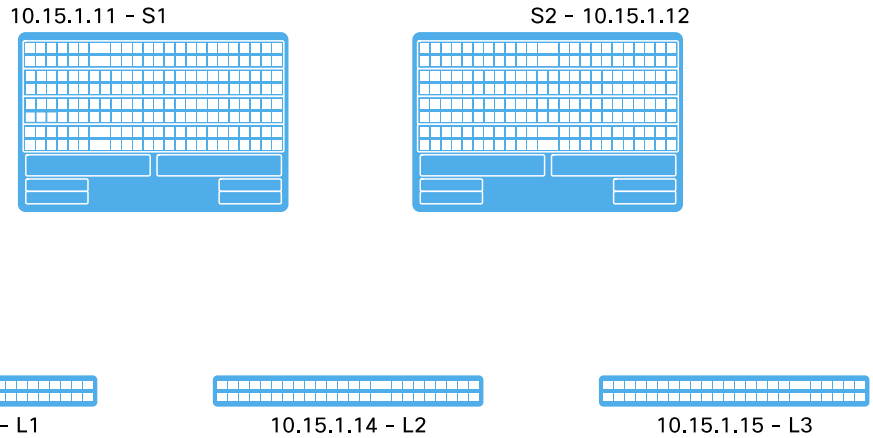

Notepad++

Ansible Concepts

VXLAN EVPN Fabric

IaC – Nexus as Code

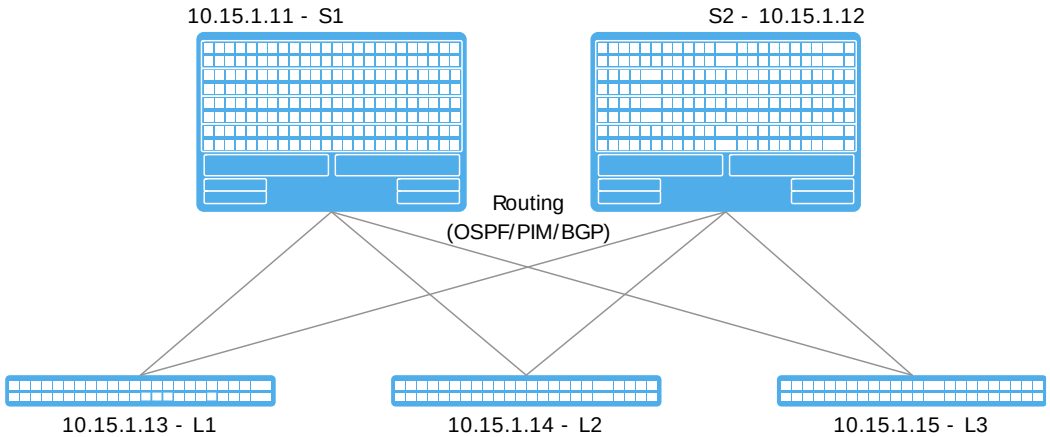
- Configure Hostnames, Features, etc (**Common** configuration)



VXLAN EVPN Fabric

IaC – Nexus as Code

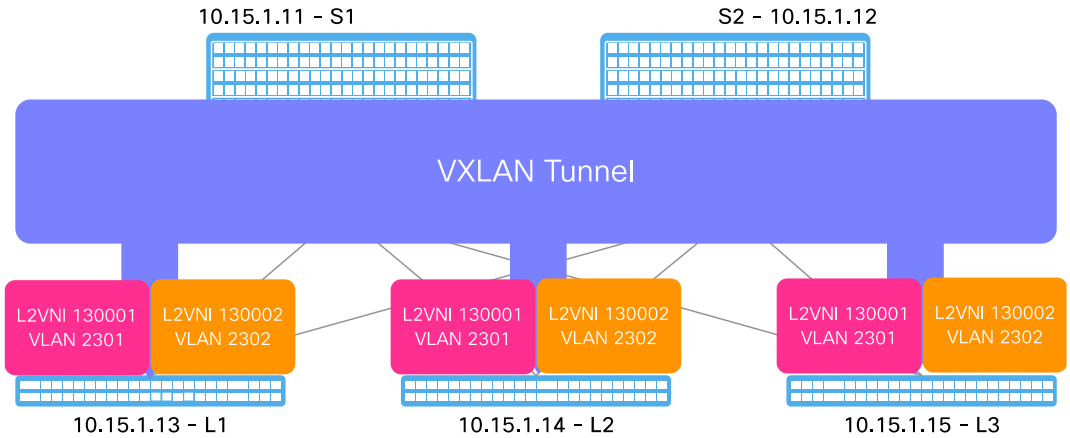
- Configure **Underlay** (Interfaces, Routing protocols, etc)



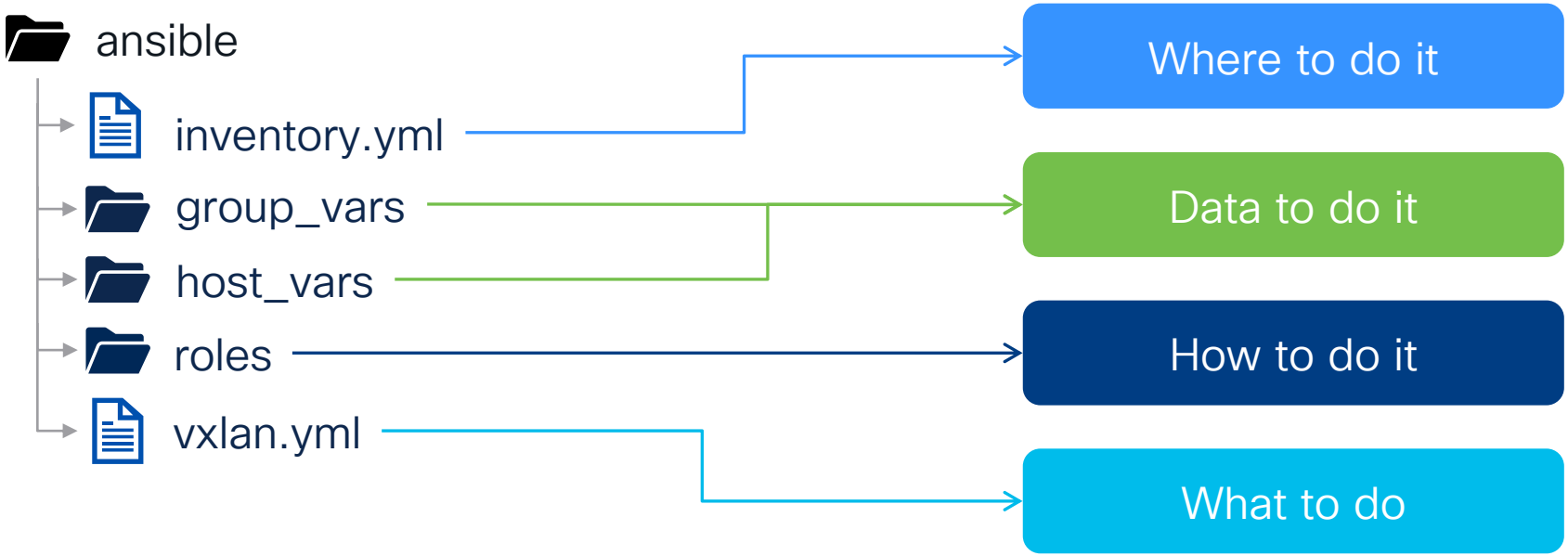
VXLAN EVPN Fabric

IaC – Nexus as Code

- Configure **Overlay** (VRFs, VLANs, SVIs, etc)

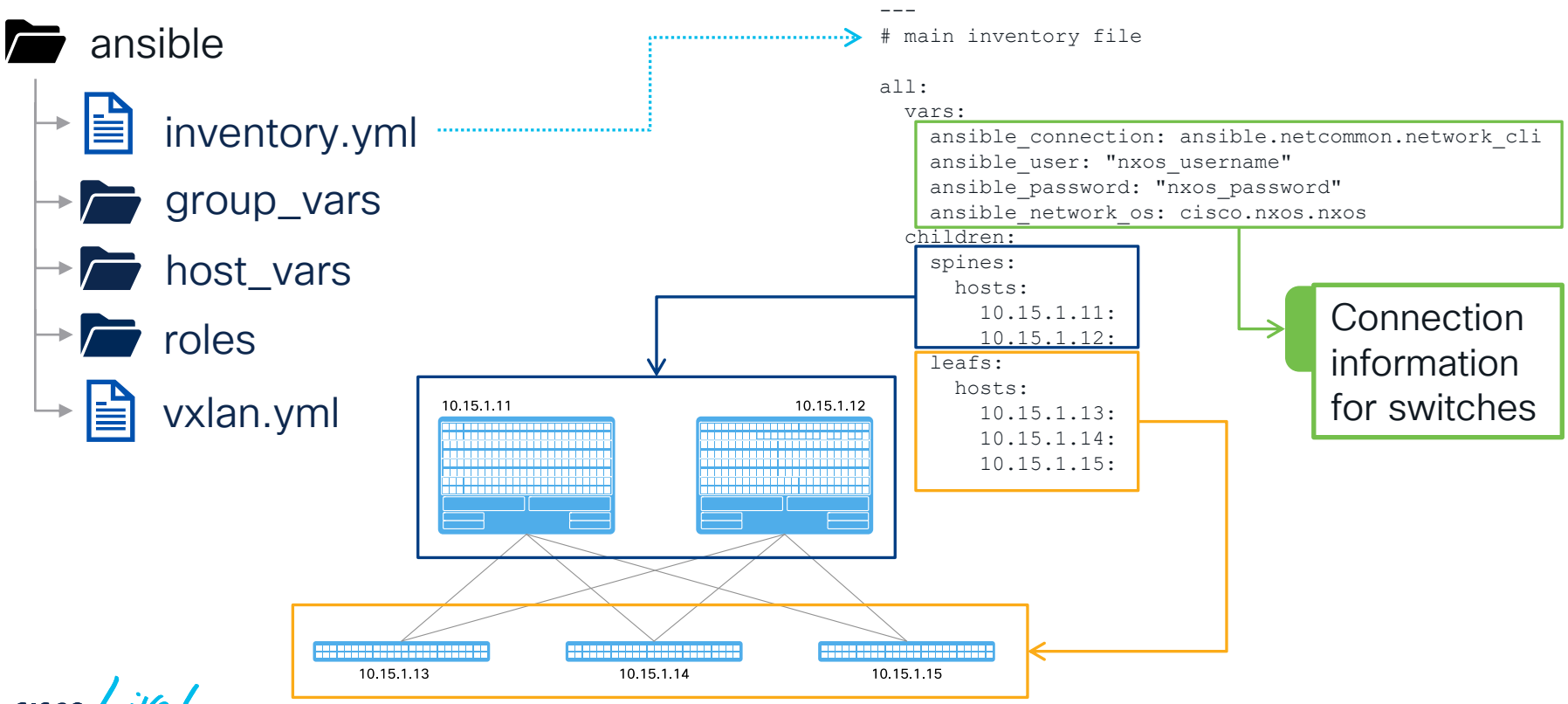


Ansible Directory Structure



Ansible Inventory

Where to do it

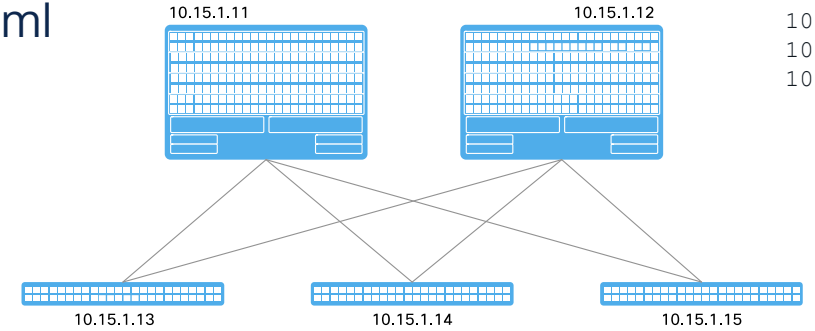


Ansible Inventory

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:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

This will be annotated shorthand in subsequent slides as `network_cli`



Ansible Playbook

What to do

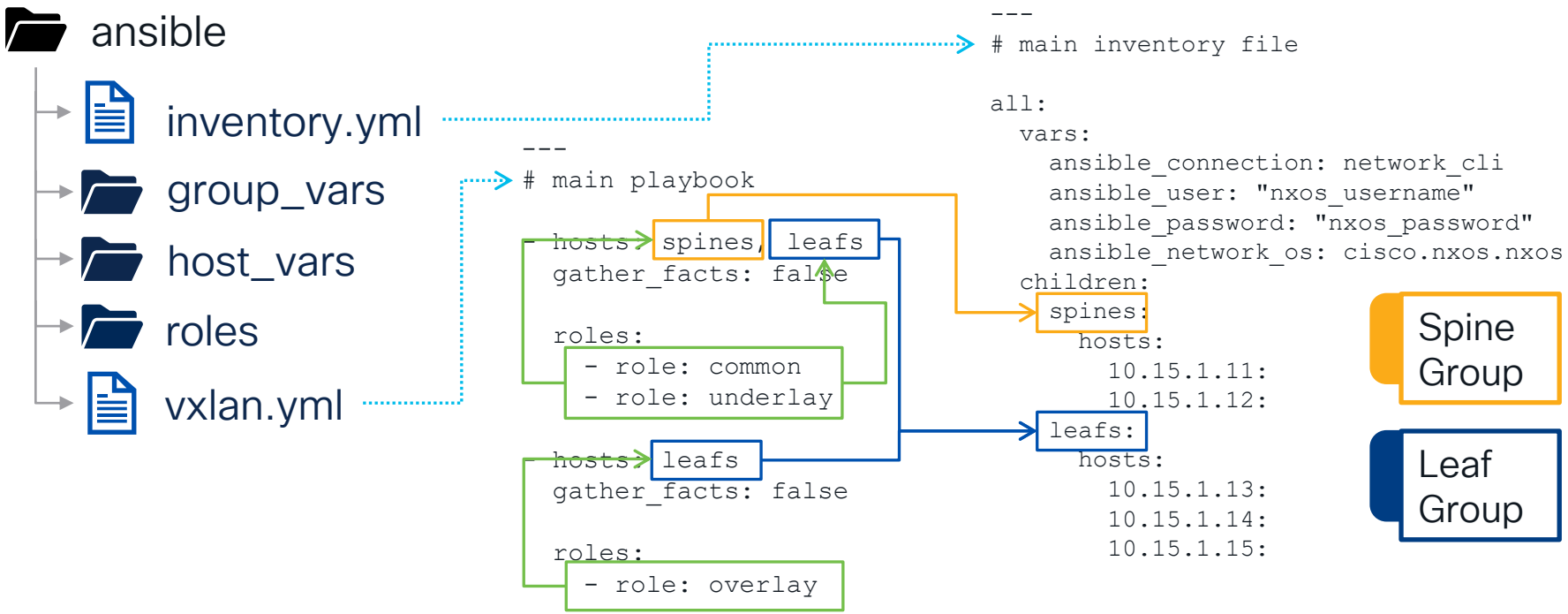

```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false

  roles:
    - role: overlay
```

Ansible Networking

- Network modules execute from control node (Ansible host)
- Collections organized by network platform/OS
- Offers multiple connection protocols

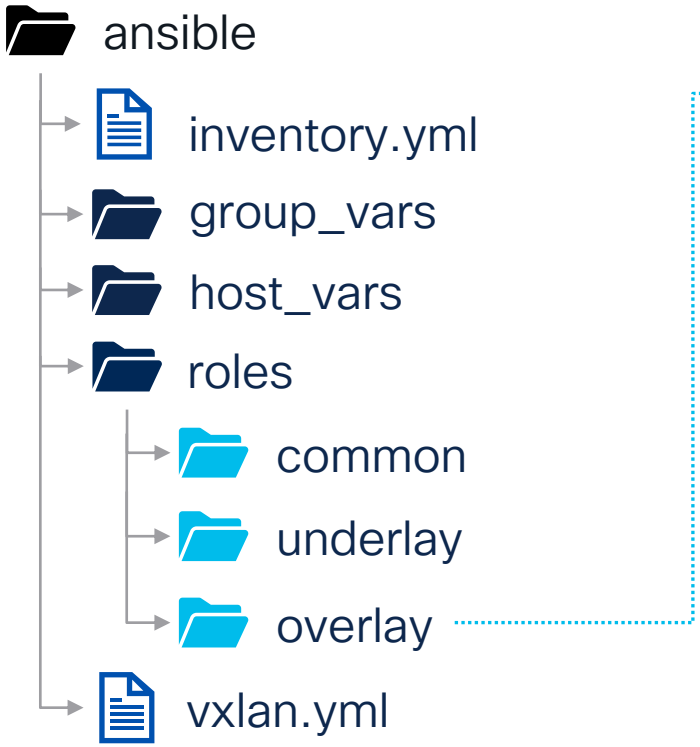
Value of ansible_connection	Protocol	Requires	Persistent
ansible.netcommon.network_cli	CLI over SSH	ansible_network_os	Yes
ansible.netcommon.httpapi	API over HTTP/HTTPS	ansible_network_os	Yes

Ansible Playbook Relationships



Ansible Roles

How to do it

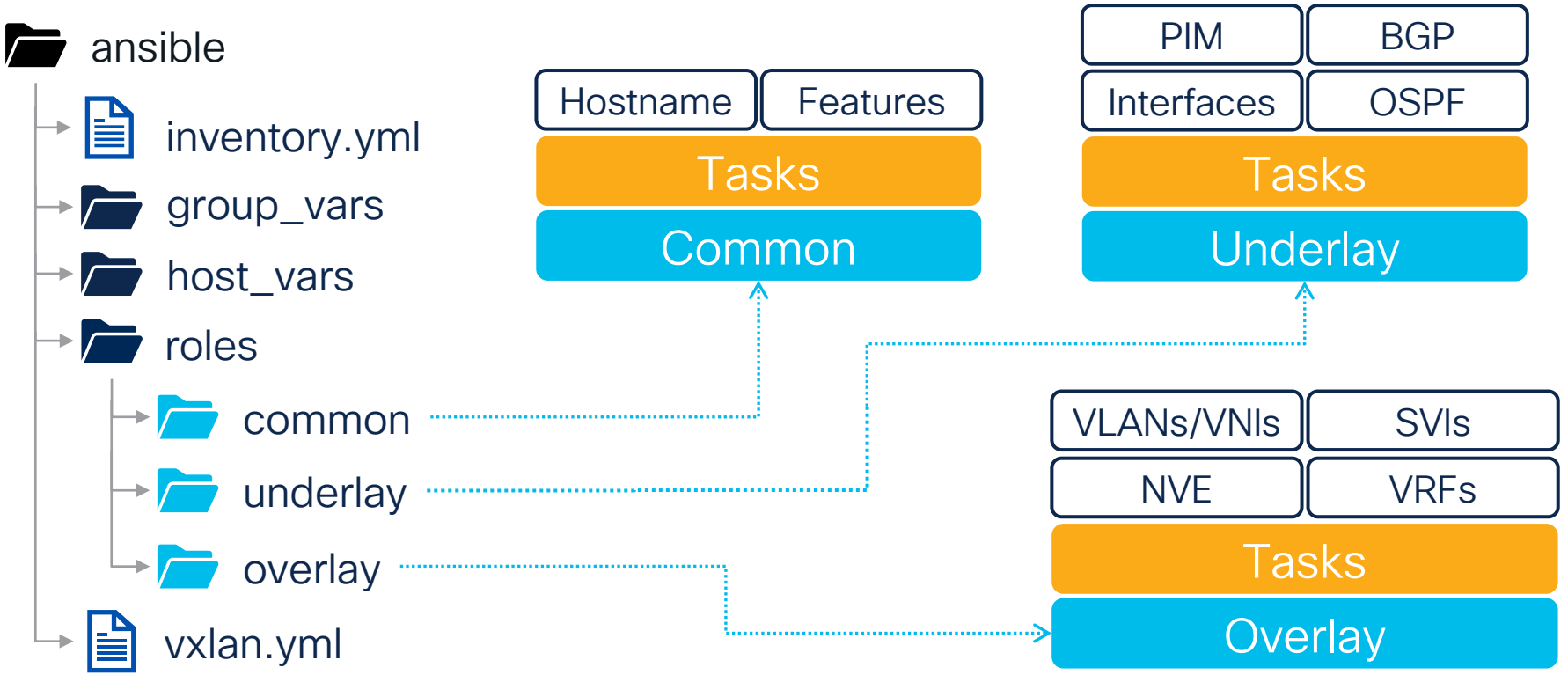



```
❯ ansible-galaxy init overlay
- Role overlay was created successfully
❯ tree overlay
overlay
├── README.md
├── tasks
│   └── main.yml
├── templates
└── vars
    └── main.yml
```

This is how we do it

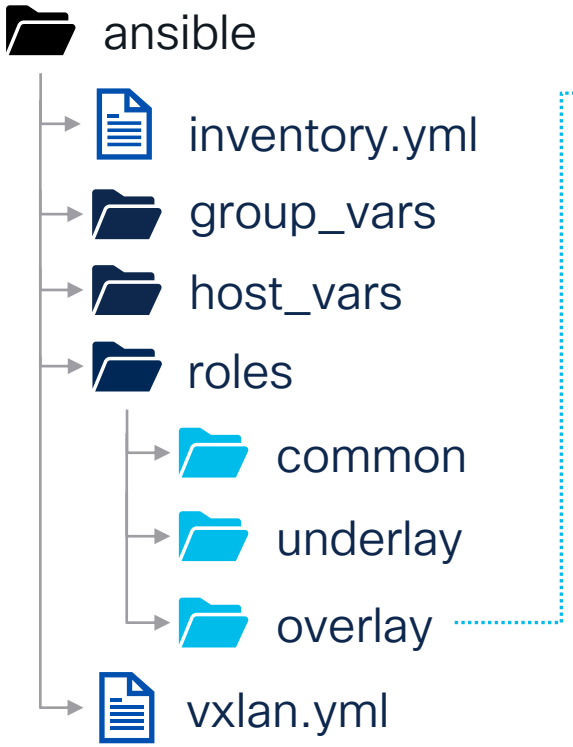
Ansible Roles – Tasks

How to do it



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
        vlan_id: 101
        mapped_vni: 10101
      - name: DB Servers
        vlan_id: 102
        mapped_vni: 10102
      - name: vMotion
        vlan_id: 103
        mapped_vni: 10103
    state: merged
<snip>
```

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

Putting It All Together

Where to do it

inventory.yml

```
---
# main inventory file

all:
  vars:
    ansible_connection: 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:
    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: 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: merged
<snip>
```

Putting It All Together

Where to do it	What to do	How to do it

inventory.yml

```
---
# main inventory file

all:
  vars:
    ansible_connection: 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:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

vxlan.yml

```
---
# main playbook

- hosts: spines, leafs
  gather_facts: false

  roles:
    - role: common
    - role: underlay

- hosts: leafs
  gather_facts: false

  roles:
    - role: overlay
```

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: merged
<snip>
```

Three VLAN-to-VNI mappings configured on all leaf switches!

Putting It All Together

Where to do it

inventory.yml

```
---
# main inventory file

all:
  vars:
    ansible_connection: 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:
    leafs:
      hosts:
        10.15.1.13:
        10.15.1.14:
        10.15.1.15:
```

What to do

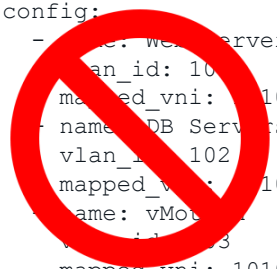
vxlan.yml

```
---
# 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: Configure VLAN-to-VNI Mappings
  cisco.nxos.nxos_vlans:
    config:
      -
        mapped_vni: 10103
    state: merged
<snip>
```



Do not do Ansible variables this way!

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

Data to do it

- 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

Data to do it

Go from this... → To this...

```
# vars defined somewhere in Ansible

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

```
# 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
```

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

Ansible Loop with Jinja2 Syntax

Data to do it

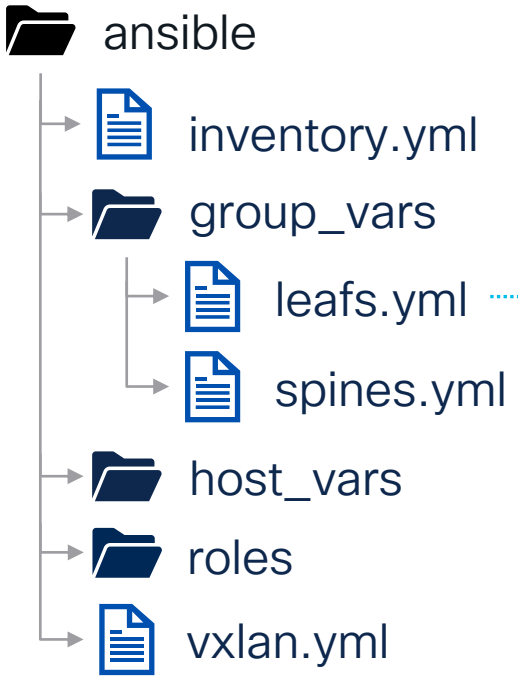
- Use Ansible loop to iterate data for a tasks

```
# 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
```

Ansible Group Vars

Data to do it

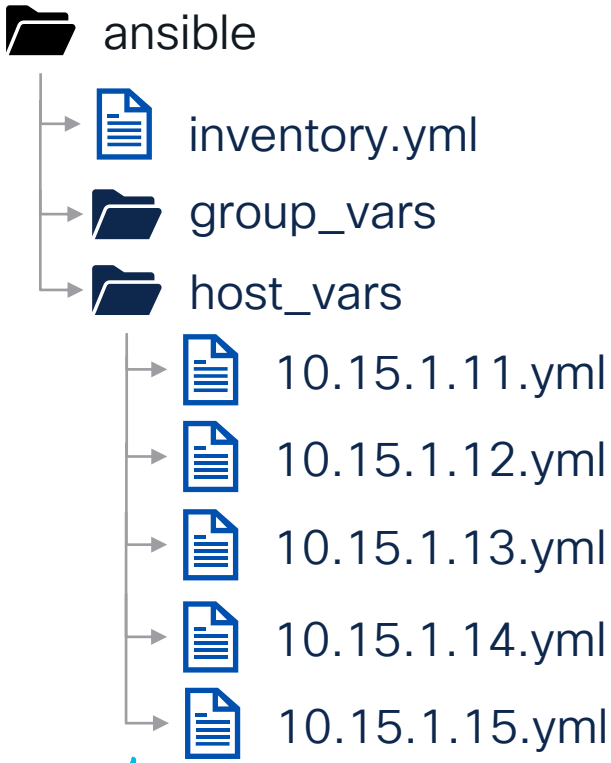



```
---
# var file for leafs group

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

networks:
  - 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
```

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

Ansible Host Vars

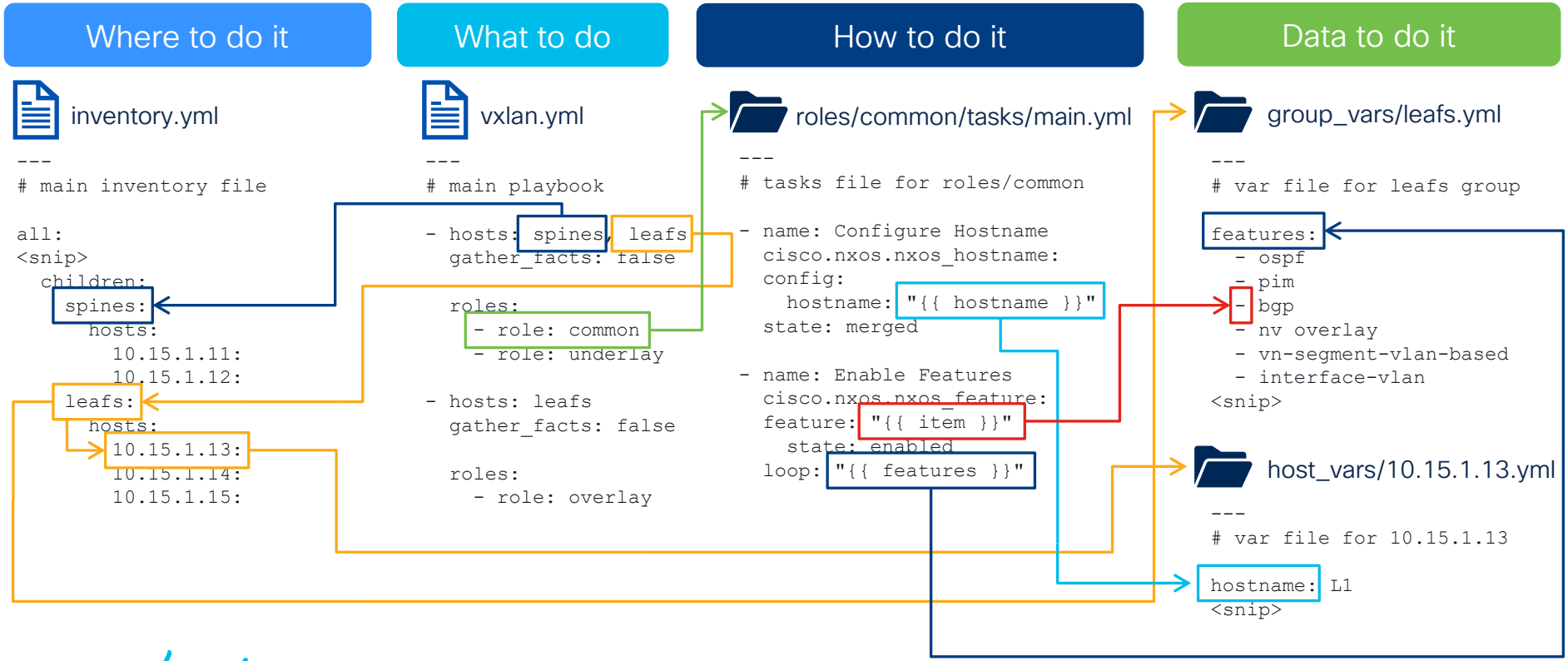
Data to do it




```
---
# 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
```

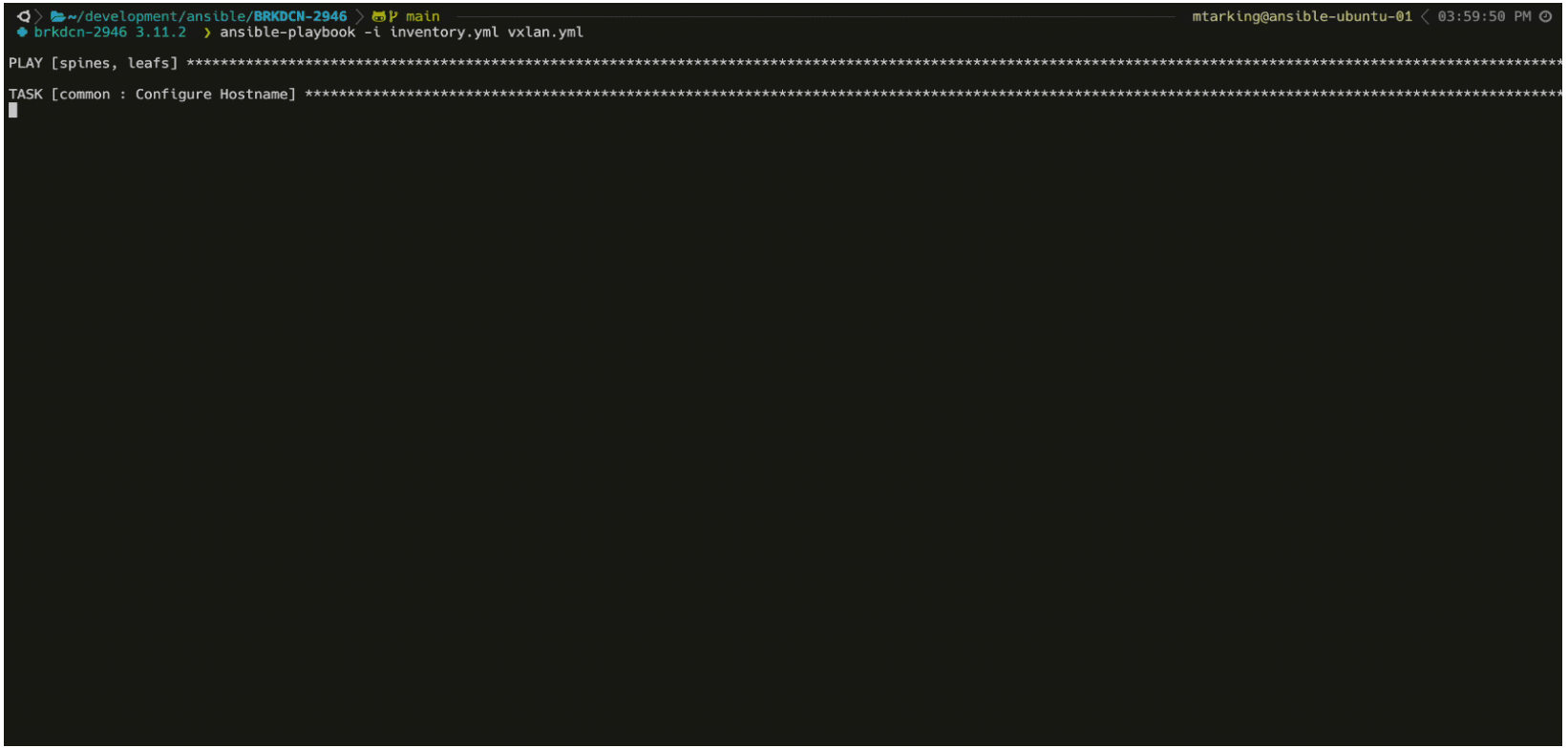
- host_vars files are named and referenced after device IP address or FQDN
- match inventory name!
- contains data specific to that device

Putting It All Together



Executing Ansible Playbooks

```
ansible-playbook -i <inventory file> <playbook file>
```



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.

A word about Jinja Templating

Data to do it

- 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 %}
```

NXOS Config Fallback Module

How to configure NXOS when a module is missing

- 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

How to send commands to NXOS when a module is missing

- 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

An Example – Add VXLAN EVPN Overlay

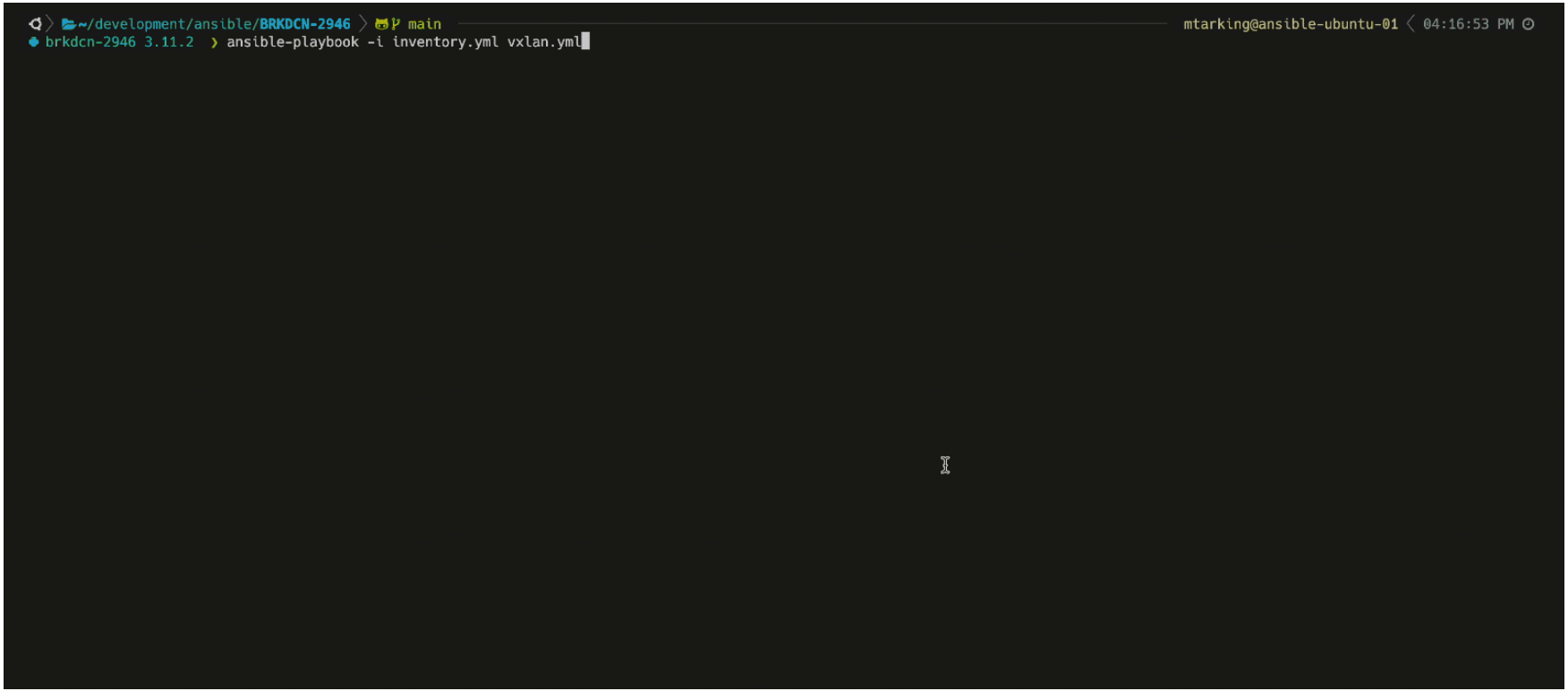
```
vlan 500
  vn-segment 50000
vlan 101
  vn-segment 10101
vrf context AnsibleVRF
  vni 50000
  rd auto
  address-family ipv4 unicast
    route-target both auto evpn
evpn
  vni 10101 l2
    rd auto
    route-target import auto
    route-target export auto
```

Modules Required

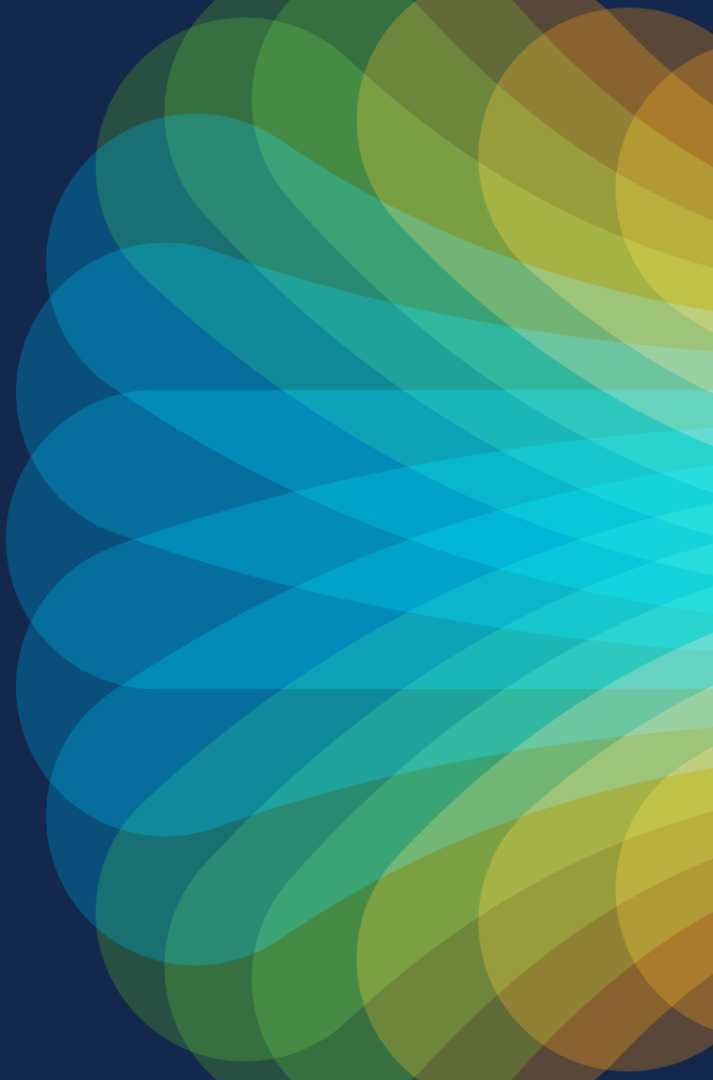
ansible.builtin.set_fact
cisco.nxos.nxos_vlans
cisco.nxos.nxos_overlay_global
cisco.nxos.nxos_vxlan_vtep
cisco.nxos.nxos_vxlan_vtep_vni
cisco.nxos.nxos_evpn_vni
cisco.nxos.nxos_vrf
cisco.nxos.nxos_vrf_af
cisco.nxos.nxos_vrf_interface
cisco.nxos.nxos_interfaces
cisco.nxos.nxos_l3_interfaces

```
interface Vlan500
  no shutdown
  vrf member AnsibleVRF
  ip forward
interface Vlan101
  no shutdown
  vrf member AnsibleVRF
  ip address 10.0.11.1/24
  fabric forwarding mode anycast-gateway
interface nve1
  no shutdown
  source-interface loopback1
  host-reachability protocol bgp
  member vni 50000 associate-vrf
  member vni 10101
    mcast-group 239.0.0.1
```

An Example – Add VXLAN EVPN Overlay



Infrastructure as Code with NDFC and Ansible



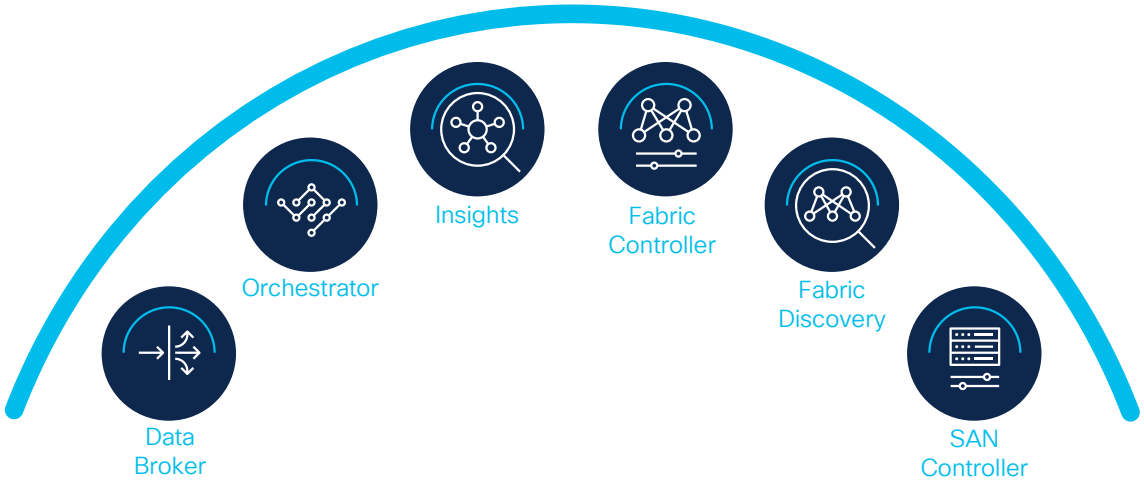
cisco Live!

Cisco Nexus Dashboard Fabric Controller (NDFC)

Formerly Called (DCNM)

Nexus Dashboard

Simple to Automate, Simple to Consume



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

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

20/20 Hindsight Tech “Specs”

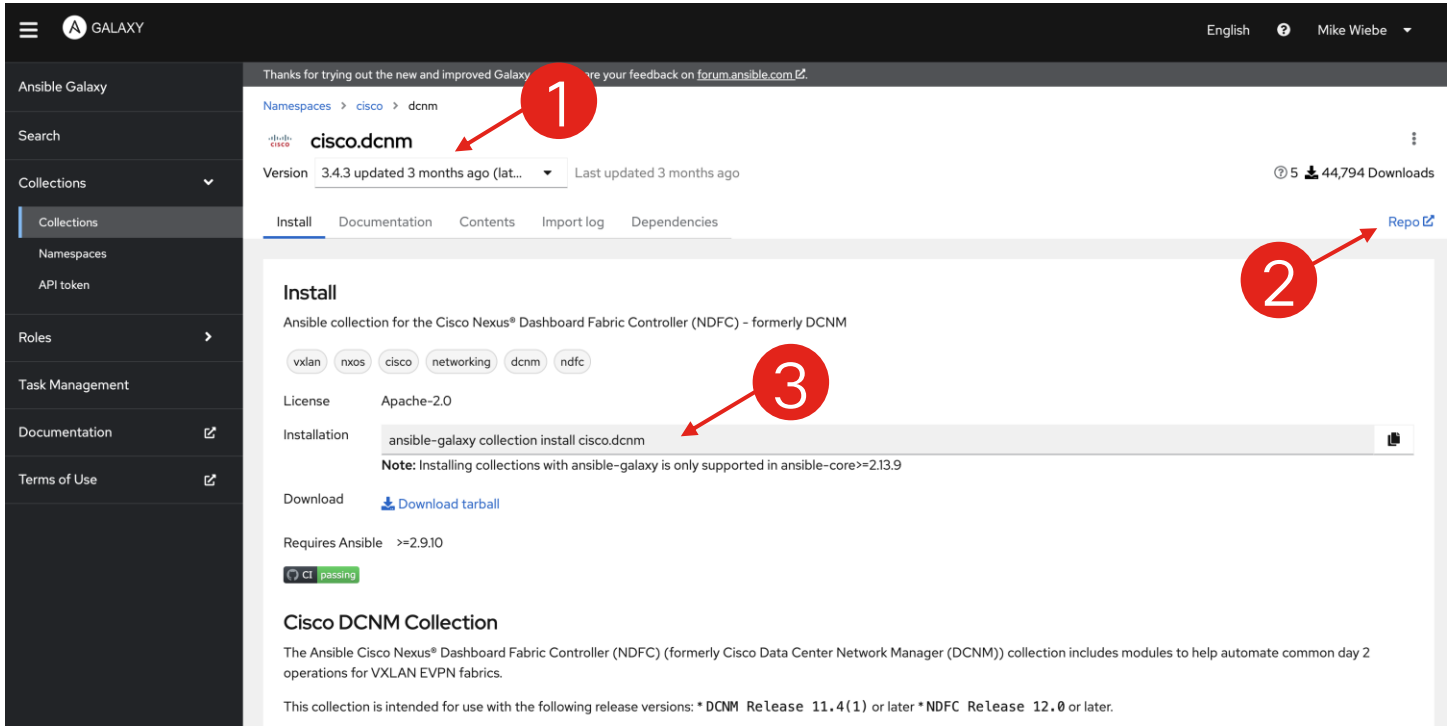
NDFC - https://galaxy.ansible.com/cisco/dcnm

Hindsight “Specs” Model

I See Ansible Collection Names

Reference Slide

NDFC Ansible Galaxy Collection Site



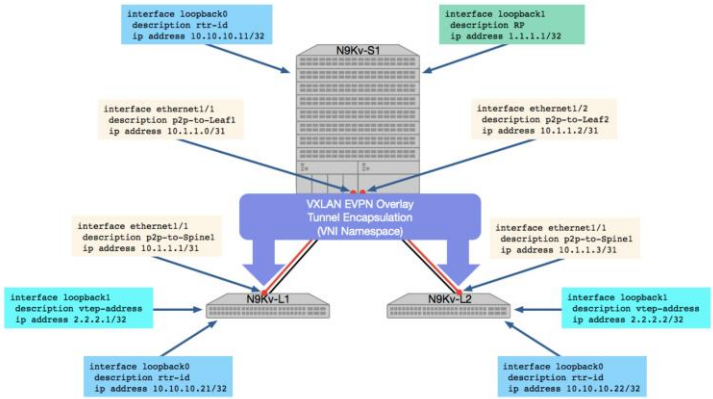
1. Collection Version
2. GitHub Repo Link
3. Install Command

NDFC - https://galaxy.ansible.com/cisco/dcnm

Let’s Build Something With NDFC and Ansible Together!

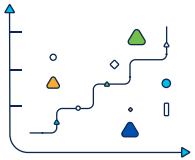
Levels of Complexity – CLI

```
feature bgp
feature interface-vlan
feature vn-segment-vlan-based
feature nv overlay
nv overlay evpn
vlan 10
  vn-segment 10000
vlan 11
  vn-segment 10011
interface loopback0
    ip address 10.10.10.21/32
    ip pim sparse-mode
    ip router ospf UNDERLAY area 0
interface loopback1
    ip address 2.2.2.1/32
    ip pim sparse-mode
    ip router ospf UNDERLAY area 0
vrf context Tenant-1
  vni 10000
  rd auto
  address-family ipv4 unicast
    route-target both auto evpn
```




```
router bgp 65001
  router-id 10.10.10.21
  neighbor 10.10.10.11
    remote-as 65001
    update-source loopback0
    address-family l2vpn evpn
      send-community
      send-community extended
  vrf Tenant-1
    address-family ipv4 unicast
      advertise l2vpn evpn
evpn
  vni 10011 l2
    rd auto
    route-target import auto
    route-target export auto
interface Vlan10
  no shutdown
  vrf member Tenant-1
  ip forward
interface Vlan11
  no shutdown
  vrf member Tenant-1
  ip address 10.0.11.1/24
  fabric forwarding mode anycast-gateway
interface nve1
  no shutdown
  source-interface loopback1
  host-reachability protocol bgp
  member vni 10000 associate-vrf
  member vni 10011
    mcast-group 239.0.0.11
```

Levels of Complexity – NXOS Modules



```
feature bgp
feature interface-vlan
feature vn-segment-vlan-based
feature nv overlay
nv overlay evpn
vlan 10
  vn-segment 10000
vlan 11
  vn-segment 10011
interface loopback0
    ip address 10.10.10.21/32
    ip pim sparse-mode
    ip router ospf UNDERLAY area 0
interface loopback1
    ip address 2.2.2.1/32
    ip pim sparse-mode
    ip router ospf UNDERLAY area 0
vrf context Tenant-1
  vni 10000
  rd auto
  address-family ipv4 unicast
    route-target both auto evpn
```

- > nxos_feature
- > nxos_interface
- > nxos_l3_interface
- > nxos_interface_ospf
- > nxos_pim_rp_address
- > nxos_pim_interface
- > nxos_evpn_global
- > nxos_bgp
- > nxos_bgp_af
- > nxos_bgp_neighbor
- > nxos_bgp_neighbor_af
- > nxos_vlan
- > nxos_vrf
- > nxos_vrf_af
- > nxos_vrf_interface
- > nxos_vxlan_vtep
- > nxos_vxlan_vtep_vni
- > nxos_evpn_vni
- > nxos_config

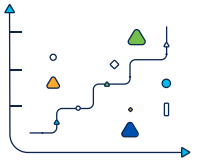
ANSIBLE

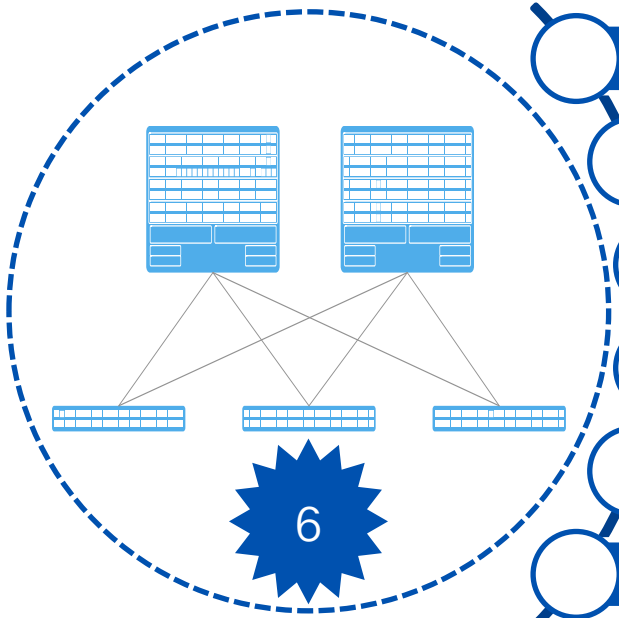
19

```
router bgp 65001
  router-id 10.10.10.21
  neighbor 10.10.10.11
    remote-as 65001
    update-source loopback0
    address-family l2vpn evpn
      send-community
      send-community extended
  vrf Tenant-1
    address-family ipv4 unicast
      advertise l2vpn evpn
evpn
  vni 10011 l2
    rd auto
    route-target import auto
    route-target export auto
interface Vlan10
  no shutdown
  vrf member Tenant-1
  ip forward
interface Vlan11
  no shutdown
  vrf member Tenant-1
  ip address 10.0.11.1/24
  fabric forwarding mode anycast-gateway
interface nve1
  no shutdown
  source-interface loopback1
  host-reachability protocol bgp
  member vni 10000 associate-vrf
  member vni 10011
    mcast-group 239.0.0.11
```

Levels of Complexity – NDFC Controller

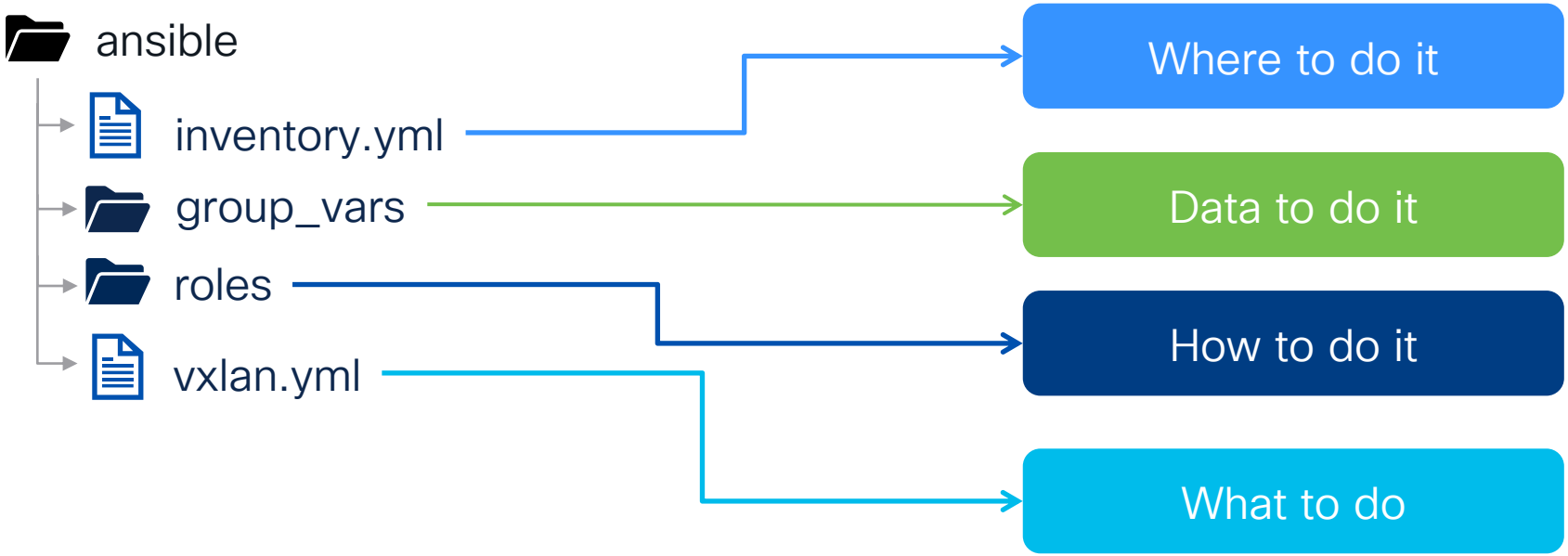
Primary VXLAN EVPN Fabric Modules





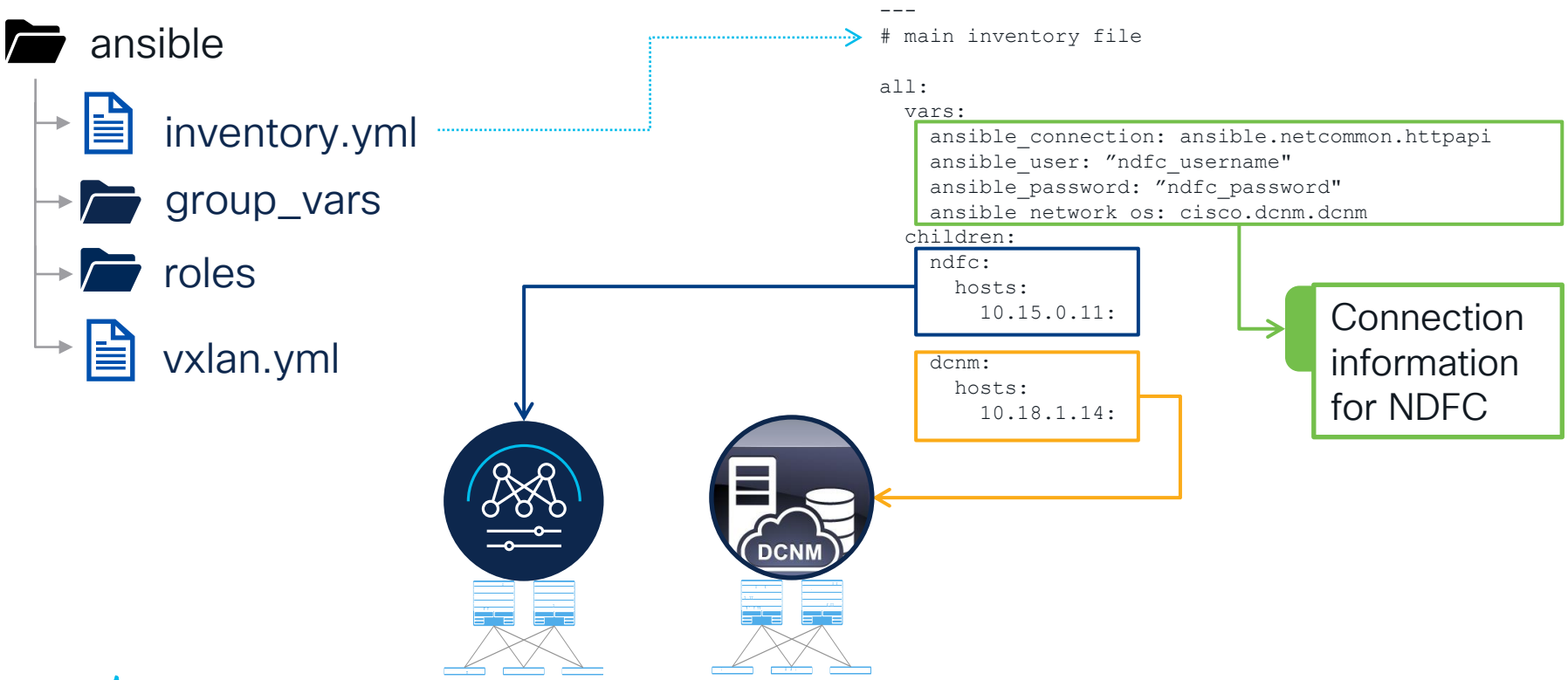
- cisco.dcnm.dcnm_rest
- cisco.dcnm.inventory
- cisco.dcnm.vrf
- cisco.dcnm.network
- cisco.dcnm.interface
- HTTPAPI Plugin

Ansible Directory Structure



Ansible Inventory

Where to do it



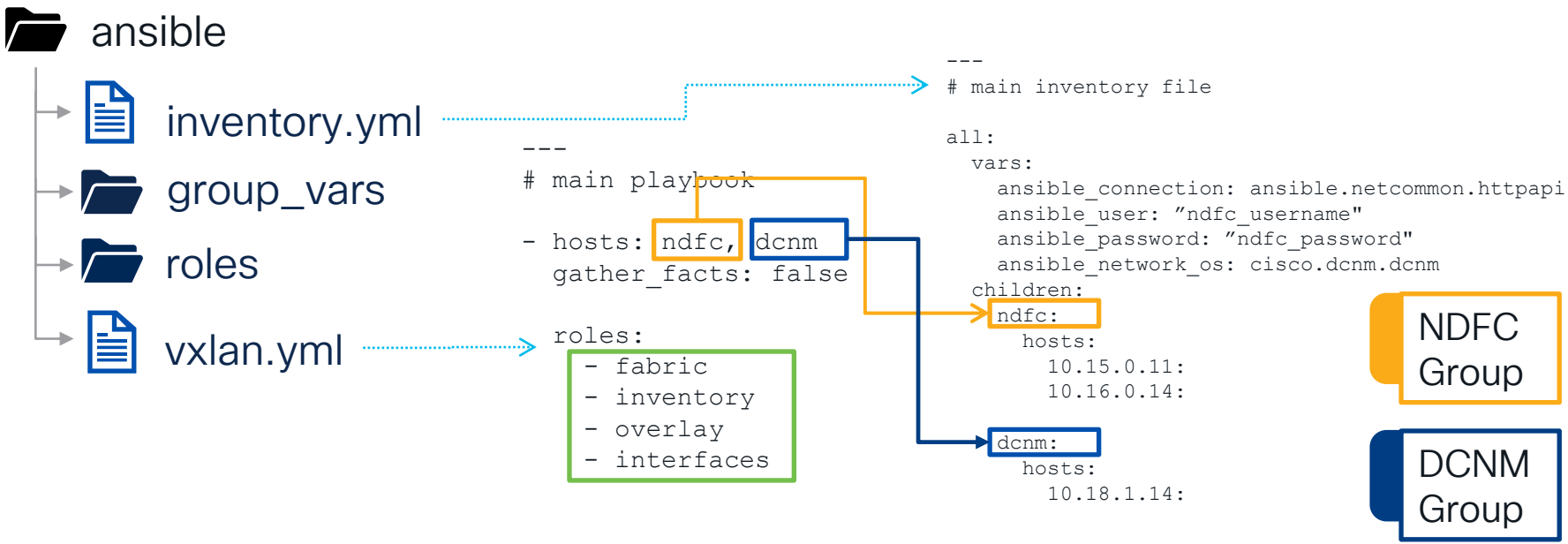
Ansible Playbook

What to do


```
---
# main NDFC playbook

- name: Build VXLAN EVPN Fabric on NDFC
  hosts: ndfc
  gather_facts: false

  roles:
    - fabric
    - inventory
    - overlay
    - interfaces
```

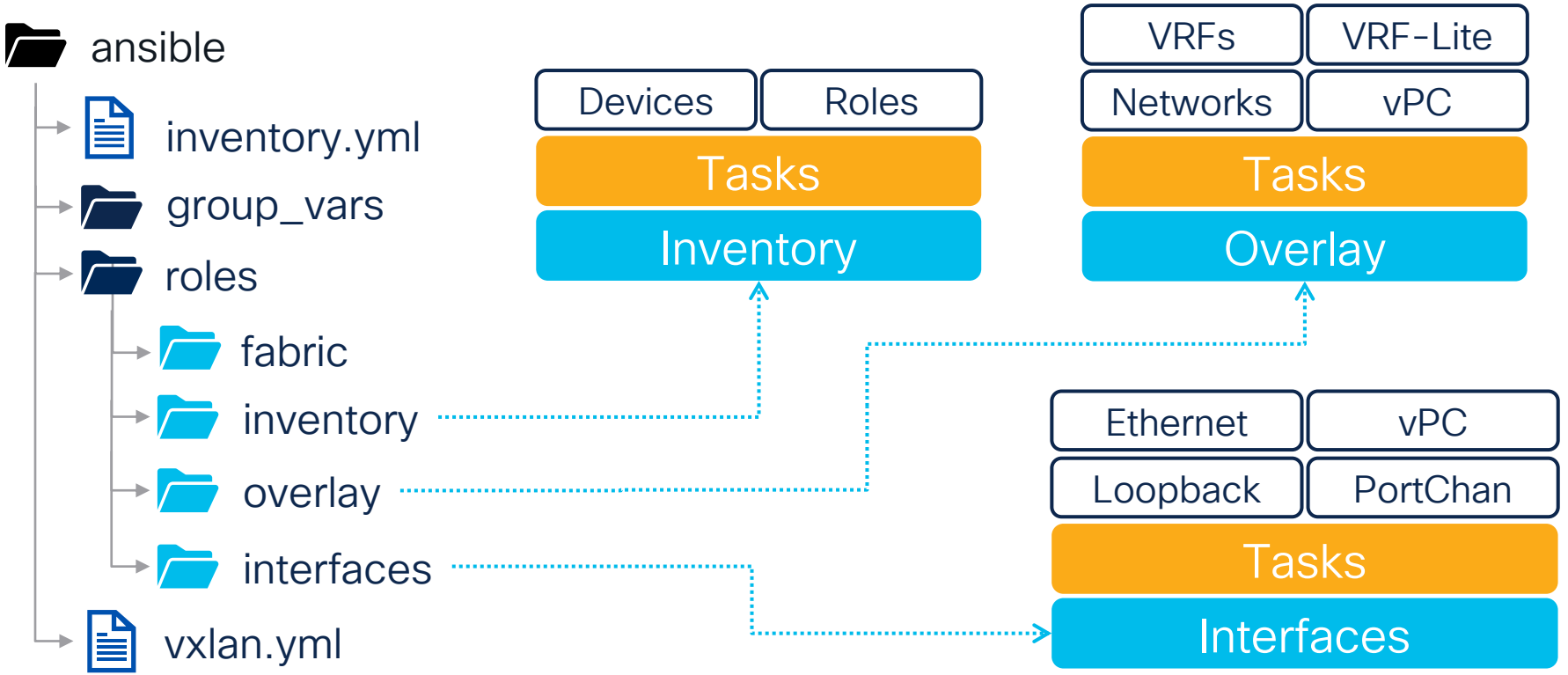
Ansible Playbook Relationships

Where to do it

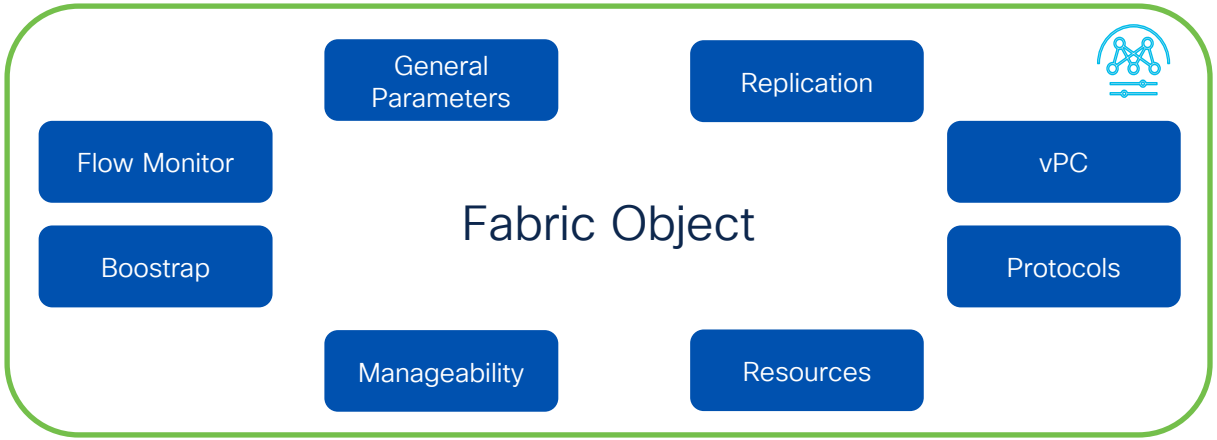


Ansible Roles – Tasks

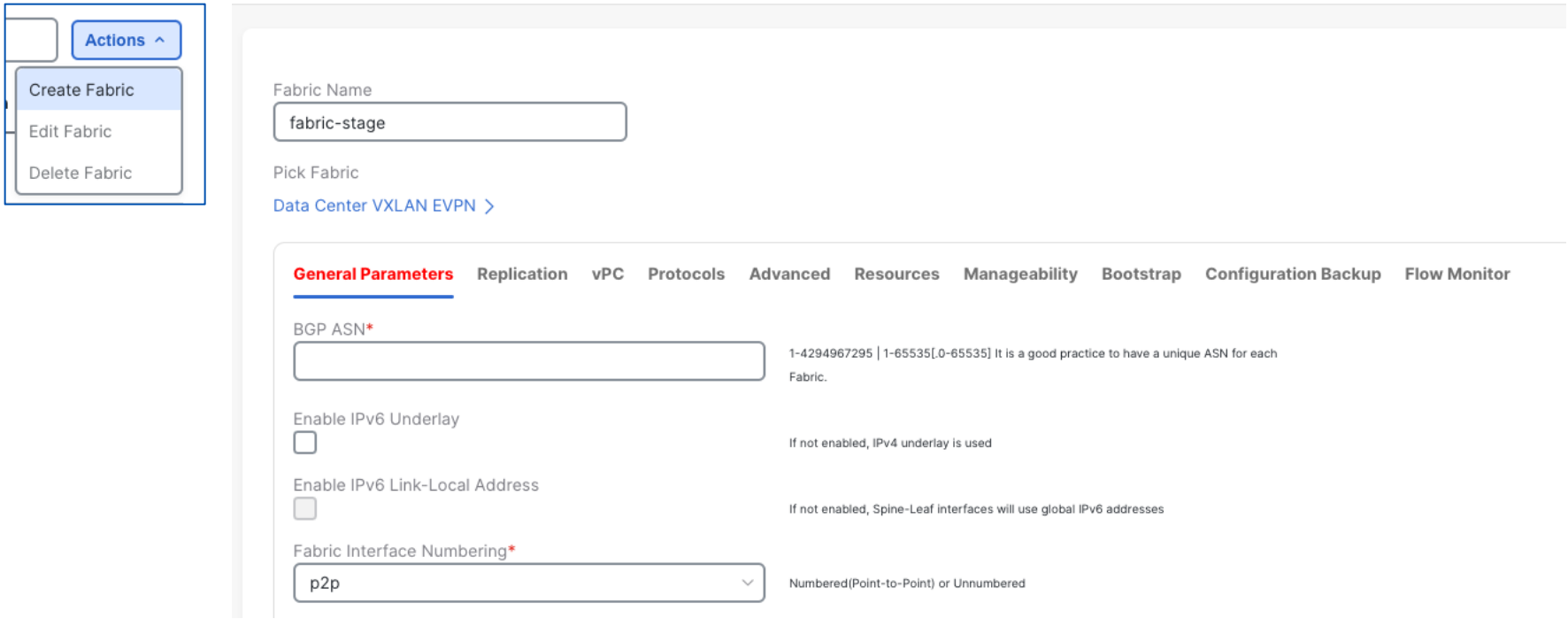
How to do it



Workflow Step1: Create VXLAN Fabric

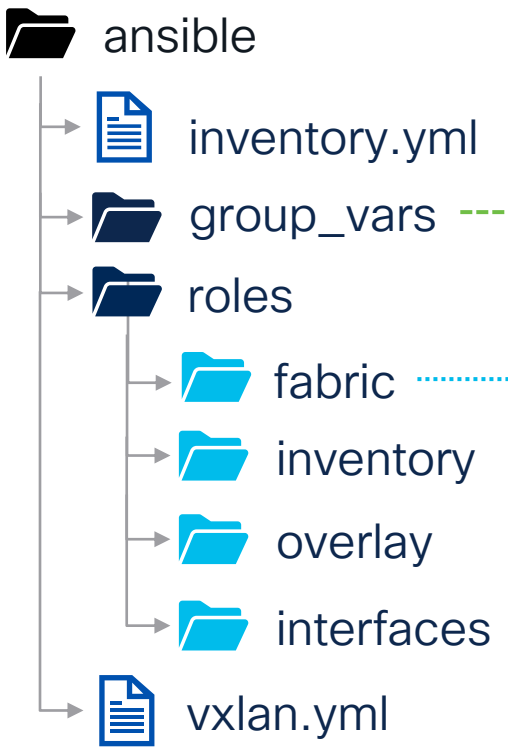


Workflow Step1: Create VXLAN Fabric



Ansible Roles – Tasks

How to do it



roles/fabric/tasks/main.yml

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

- name: Get Fabric List
  cisco.dcnm.dcnm_rest:
    method: GET
    path: "{{ fabric.gpath }}"
  register: create_fabric

 <snip>

- name: Create Fabric
  vars:
    payload:
      BGP_AS: "{{ fabric.asn }}"
  cisco.dcnm.dcnm_rest:
    method: POST
    path: "{ fabric.cpath }}"
    json_data: "{{ payload | to_json }}"
  when: create_fabric_flag
```

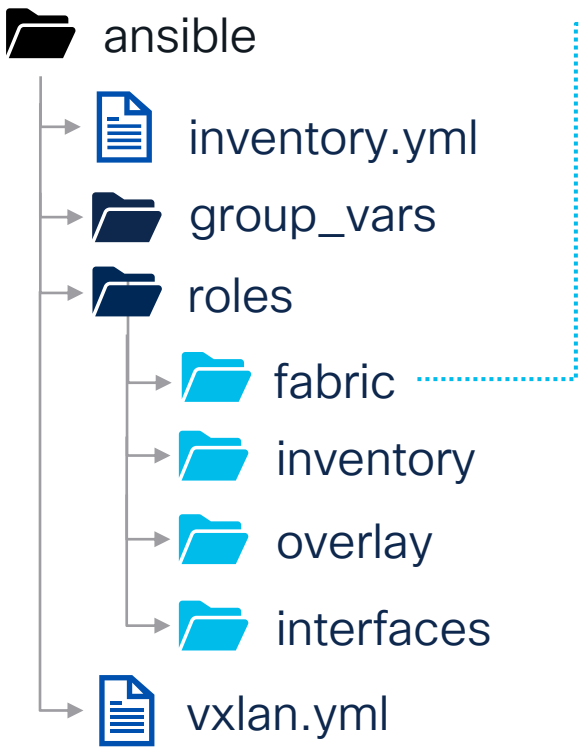
- Creates Fabric
- Uses json_data to pass in payload key/value pairs

/appcenter/cisco/ndfc/api/v1/lan-fabric/rest/control/fabrics

/appcenter/cisco/ndfc/api/v1/lan-fabric/rest/control/fabrics/{{ fabric.name }}/Easy_Fabric

Ansible Roles – Tasks

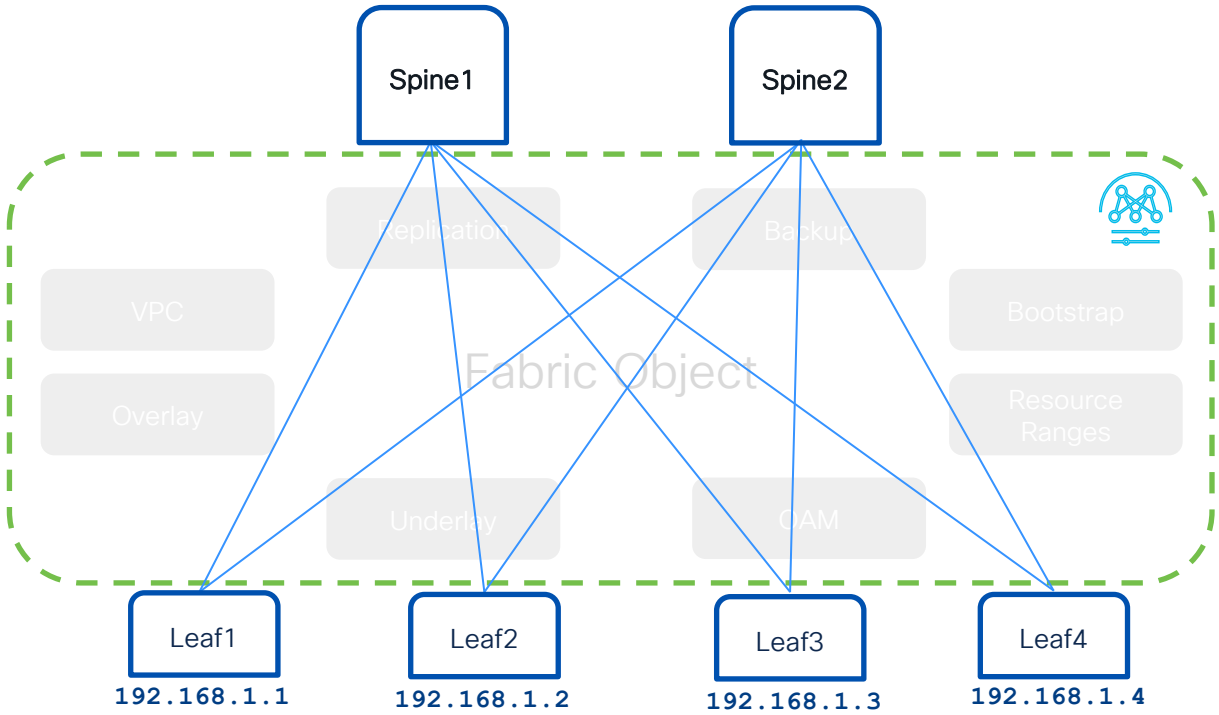
How to do it



roles/fabric/tasks/main.yml

```
- name: Intialize create_fabric_flag     <snip>
  ansible.builtin.set_fact:
    create_fabric_flag: true

- name: Check If Fabric Exists
  ansible.builtin.set_fact:
    create_fabric_flag: false
  when: item.fabricName == fabric.name
  loop: "{{ create_fabric_result.response.DATA }}"
  loop_control:
    label: "{{ item.fabricName }}"

- name: Check If Fabric Exists Log
  ansible.builtin.debug:
    msg: "Fabric {{ fabric.name }} Already Exists"
  when: not create_fabric_flag
```

Workflow Step2: Add Inventory



Workflow Step2: Add Inventory

Fabric Overview - fabric-stage

Actions

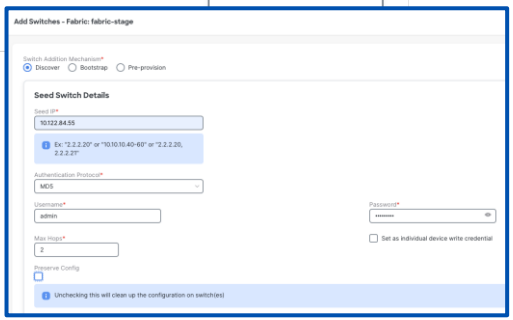
Overview | Switches | Links | Interfaces | Interface Groups | Policies | Networks | VRFs | Services | Event Analytics | History | Resources

Filter by attributes

Actions

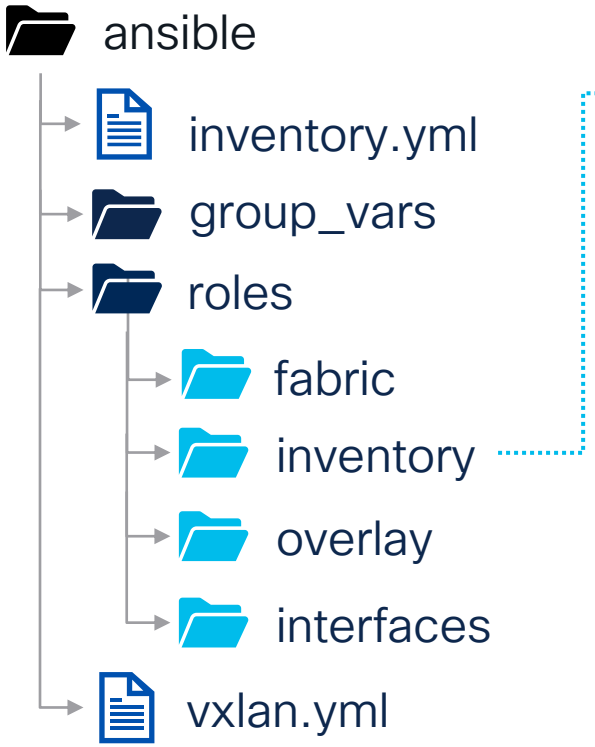
- Add Switches
- Preview
- Deploy
- Discovery
- Set Role
- vPC Pairing
- ToR/Access Pairing

Switch	IP Address	Role	Serial Number	Mode	Config Status	Oper Status	Discovery Status	Model	VPC Role	VPC Peer
staging-leaf1	10.15.10.12	Leaf	9QRB2LQLD3H	Normal	In-Sync	Major	Ok	N9K-C9300v	Primary	staging-leaf2
staging-leaf2	10.15.10.13	Leaf	9CJCO82GQEB	Normal	In-Sync	Major	Ok	N9K-C9300v	Secondary	staging-leaf1
staging-leaf3	10.15.10.14	Border	9OIZMJFK60Z	Normal	In-Sync	Minor	Ok	N9K-C9300v		
staging-spine1	10.15.10.11	Spine	9622UOESRG4	Normal	In-Sync	Minor	Ok	N9K-C9500v		



Ansible Roles – Tasks

How to do it

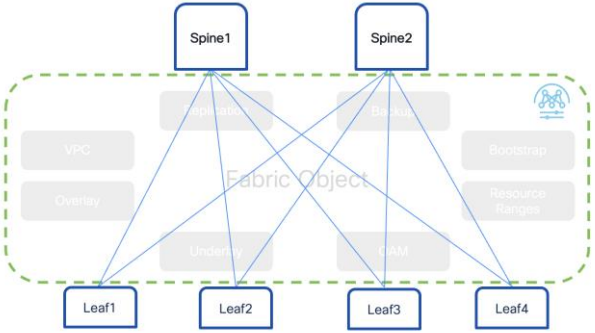


roles/inventory/tasks/main.yml

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

Add Devices To Fabric

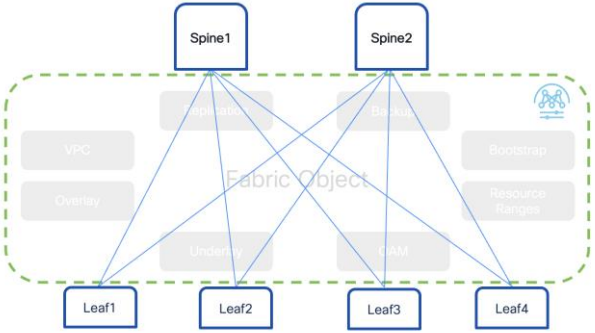


- Invoke module call to cisco.dcnm.dcnm_inventory
  - Specify target Fabric Name
- Config parameters
  - Credentials to access each switch device and set role
  - Wipe or preserve exiting device config

cisco.dcnm.dcnm_inventory

```
- name: Add switches to fabric {{ fabric.name }}
  cisco.dcnm.dcnm_inventory:
    fabric: "{{ fabric.name }}"
    config:
      - seed_ip: 192.168.1.1
        auth_proto: MD5 # choose from [MD5, SHA, MD5_DES, MD5_AES, SHA_DES, SHA_AES]
        user_name: "{{ switch_username }}"
        password: "{{ switch_password }}"
        max_hops: 0
        role: leaf # choose from [spine, leaf]
        preserve_config: False
```

Add Devices To Fabric – POAP

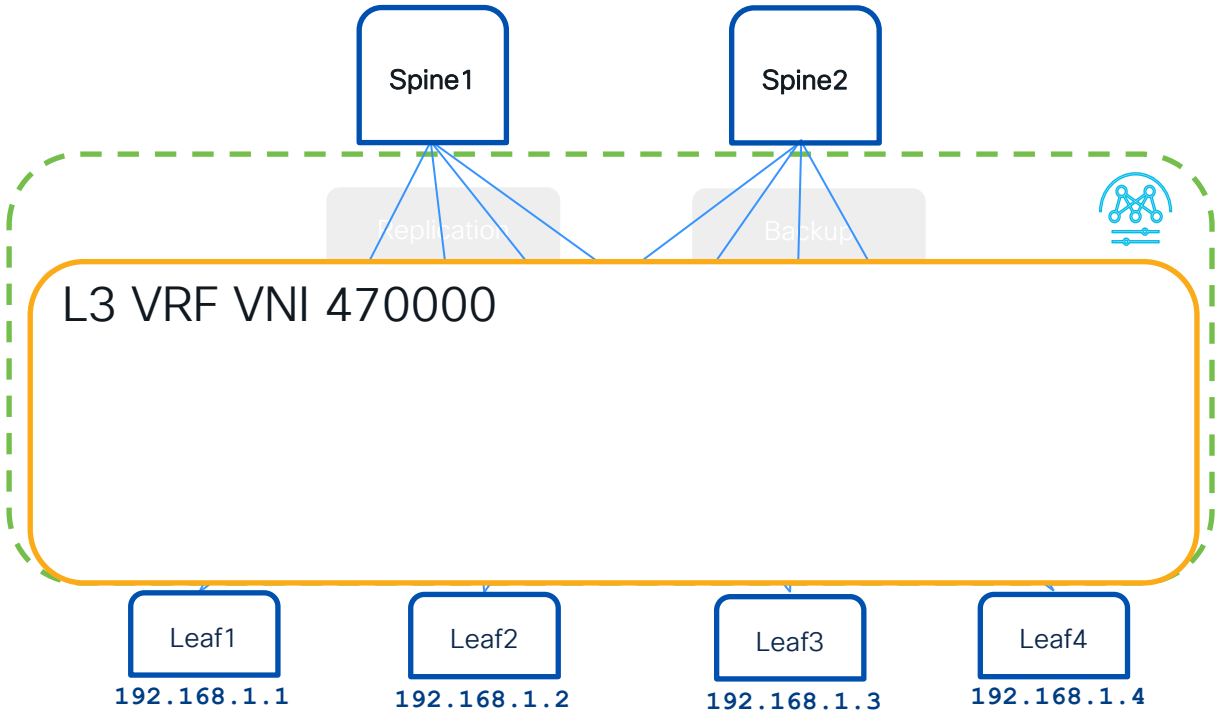


- Power On Auto Provisioning Support
  - Same module with poap: config block

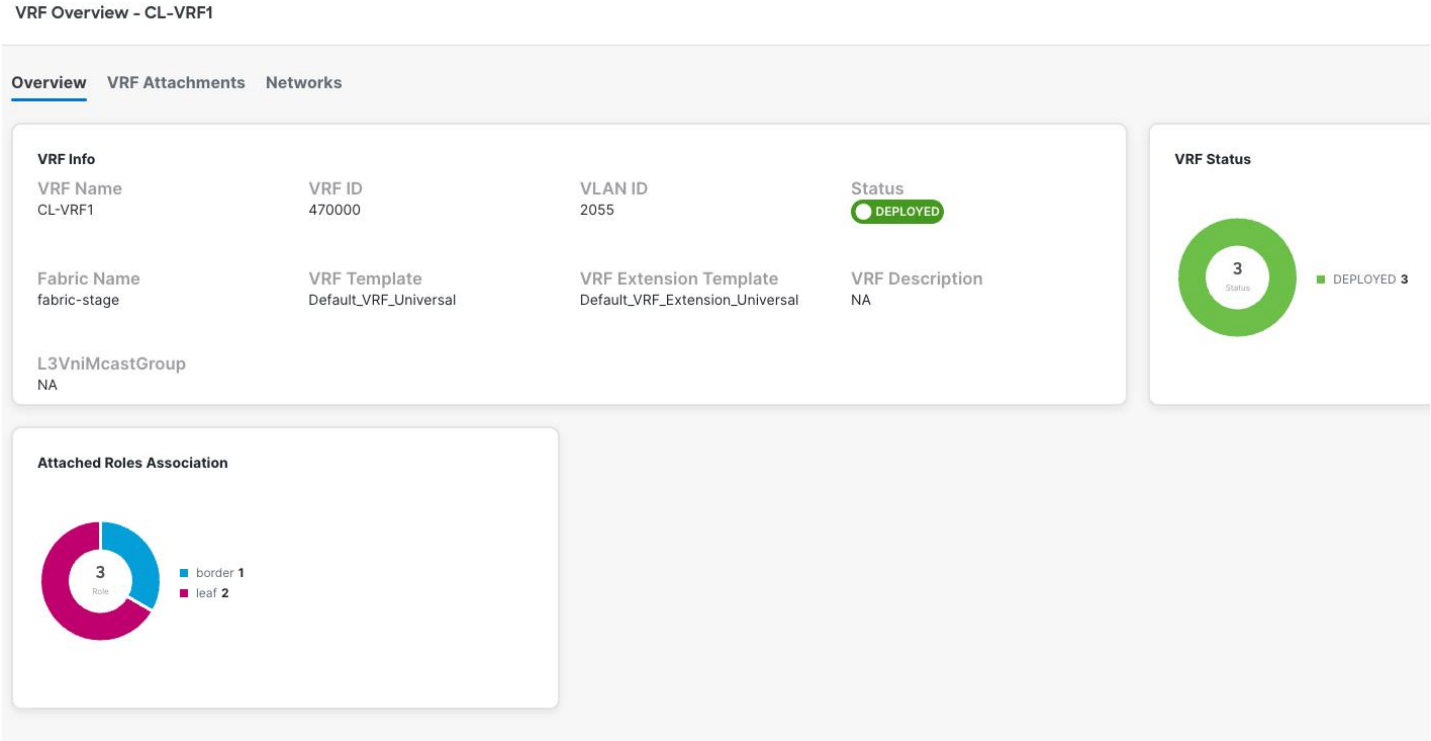
cisco.dcnm.dcnm_inventory

```
- name: Add Switch Fabric Using POAP
  cisco.dcnm.dcnm_inventory:
    fabric: "{{ fabric.name }}"
    state: merged # Only 2 options supported [merged, query] for poap config
    config:
      - seed_ip: 192.168.1.4
      {…}
        poap:
          - serial_number: 2A3BCDEFJKL
            model: 'N9K-C9300v'
            version: '9.3(7)'
            hostname: 'POAP_SWITCH'
            image_policy: "poap_image_policy"
            config_data:
              modulesModel: [N9K-X9364v, N9K-vSUP]
              gateway: 192.168.0.1/24
```

Workflow Step3: Add Overlay – VRFs

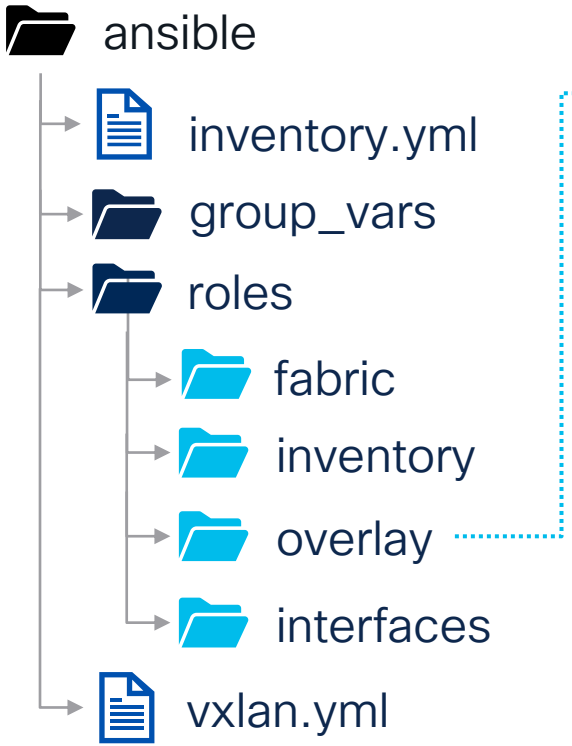


Workflow Step3: Add Overlay – VRFs



Ansible Roles – Tasks

How to do it

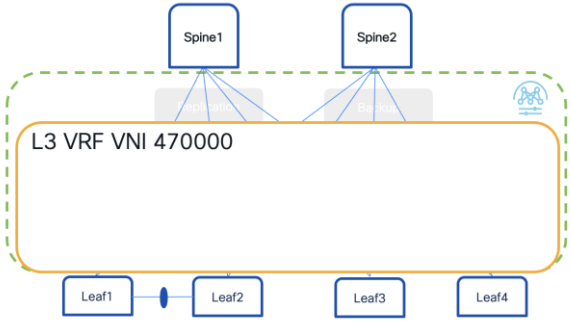


roles/overlay/tasks/main.yml

```
---
# 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
<snip>
```

- Creates VRF objects
- Attaches and deploys VRF config to all fabric leaf devices

Create and Deploy VRF Objects



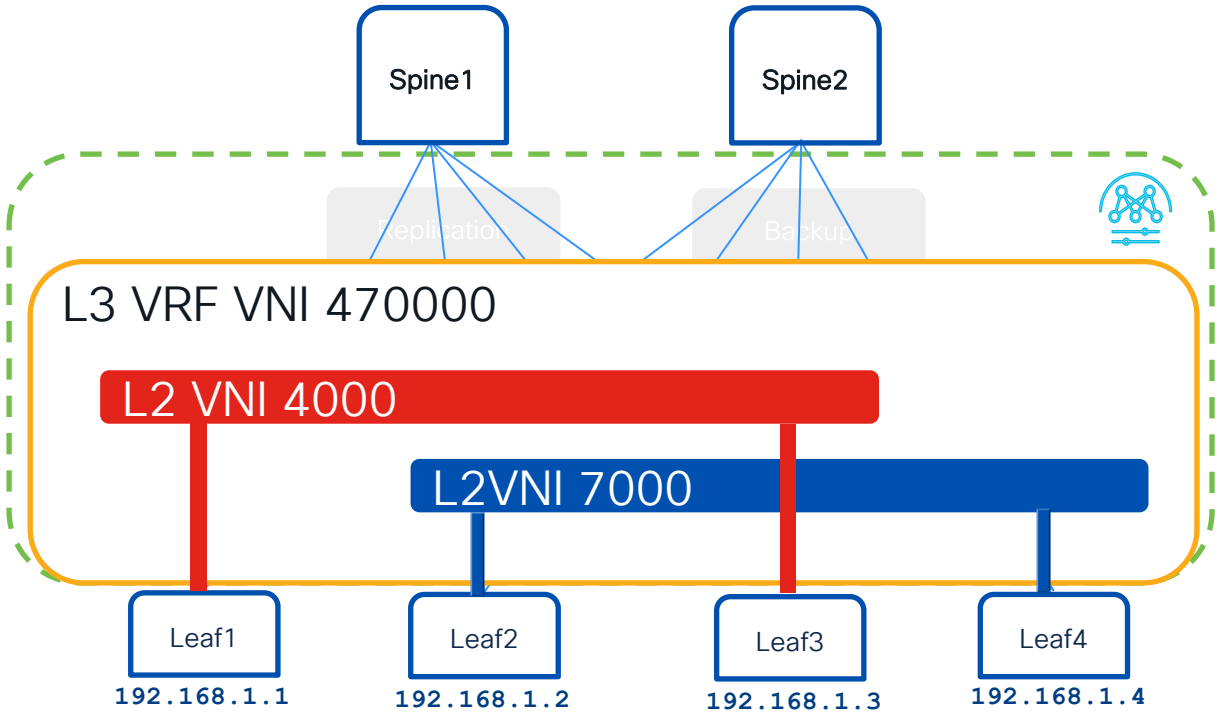
- Invoke module call to cisco.dcnm.dcnm_vrf
  - Supported states: merged, replaced, overridden
- Supports a list of VRFs
- Supports a list of Devices for Attach and Deploy

```
- name: Add VRFs to Fabric
  cisco.dcnm.dcnm_vrf:
    fabric: "{{ fabric_name }}"
    state: merged
    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
```

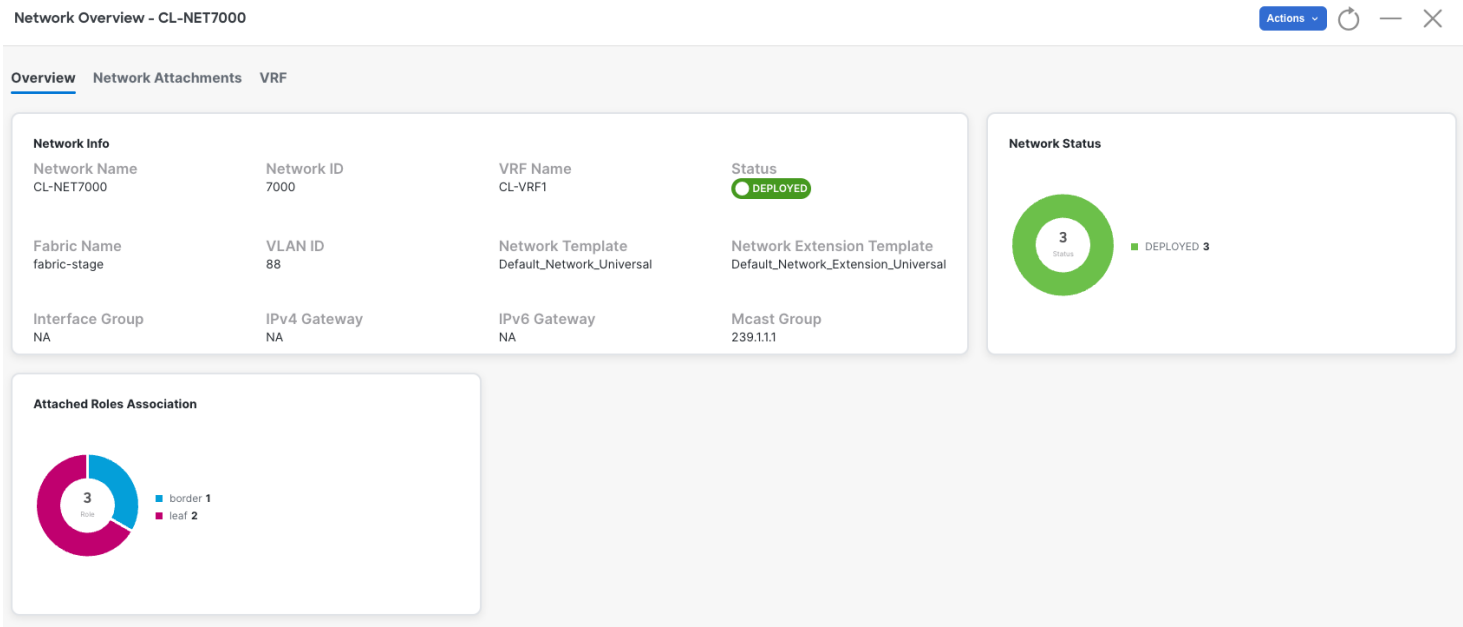
cisco.dcnm.dcnm_vrf

ANSIBLE

Workflow Step4: Add Overlay – Networks

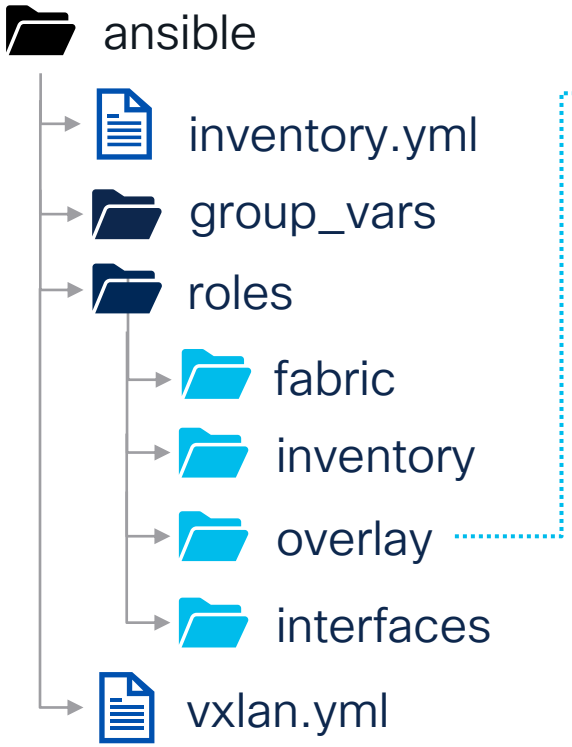


Workflow Step4: Add Overlay – Networks



Ansible Roles – Tasks

How to do it

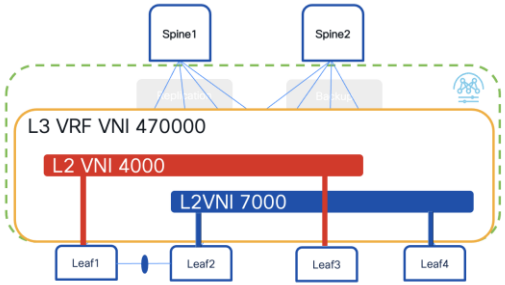


roles/overlay/tasks/main.yml

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

- 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
        <snip>
        attach:
          - 192.168.1.2
          - 192.168.1.4
```

- Creates Network objects
- Attaches and deploys Network config to fabric devices to leaf2 and leaf4

Create Network Objects



- Invoke module call to cisco.dcnm.dcnm_network
  - Supported states: merged, replaced, overridden
- Supports a list of Networks
- Supports a list of Devices for Attach and Deploy

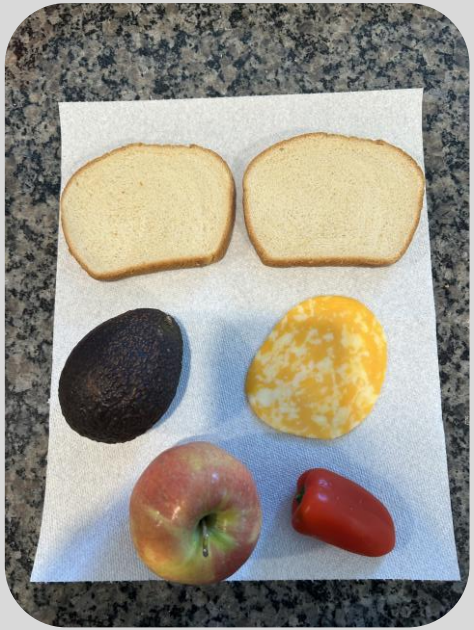
```
- name: Add Networks
  cisco.dcnm.dcnm_network:
    fabric: "{{ fabric_name }}"
    state: merged
    config:
      - net_name: CL-NET4000
        vrf_name: CL-VRF1
        net_id: 4000
        vlan_id: 55
        attach:
          - 192.168.1.1
          - 192.168.1.3
      - net_name: CL-NET7000
        vrf_name: CL-VRF1
        net_id: 7000
        vlan_id: 88
        attach:
          - 192.168.1.2
          - 192.168.1.4
```

cisco.dcnm.dcnm_network

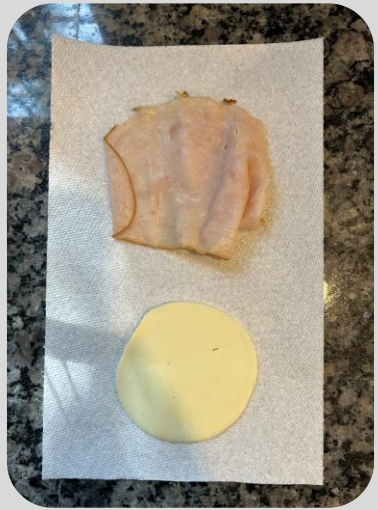
ANSIBLE

How Does One Merge A Sandwhich?

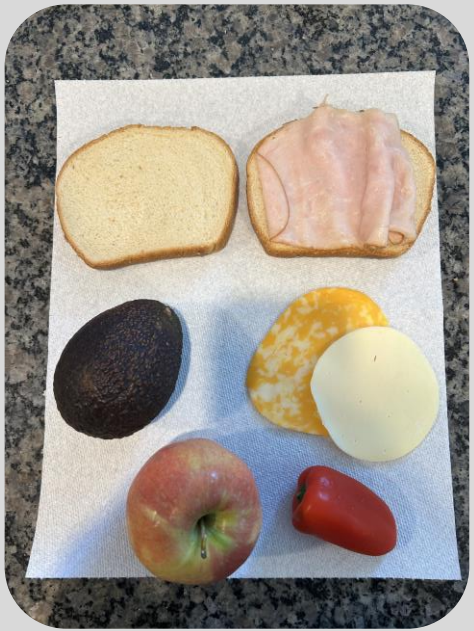
Initial State



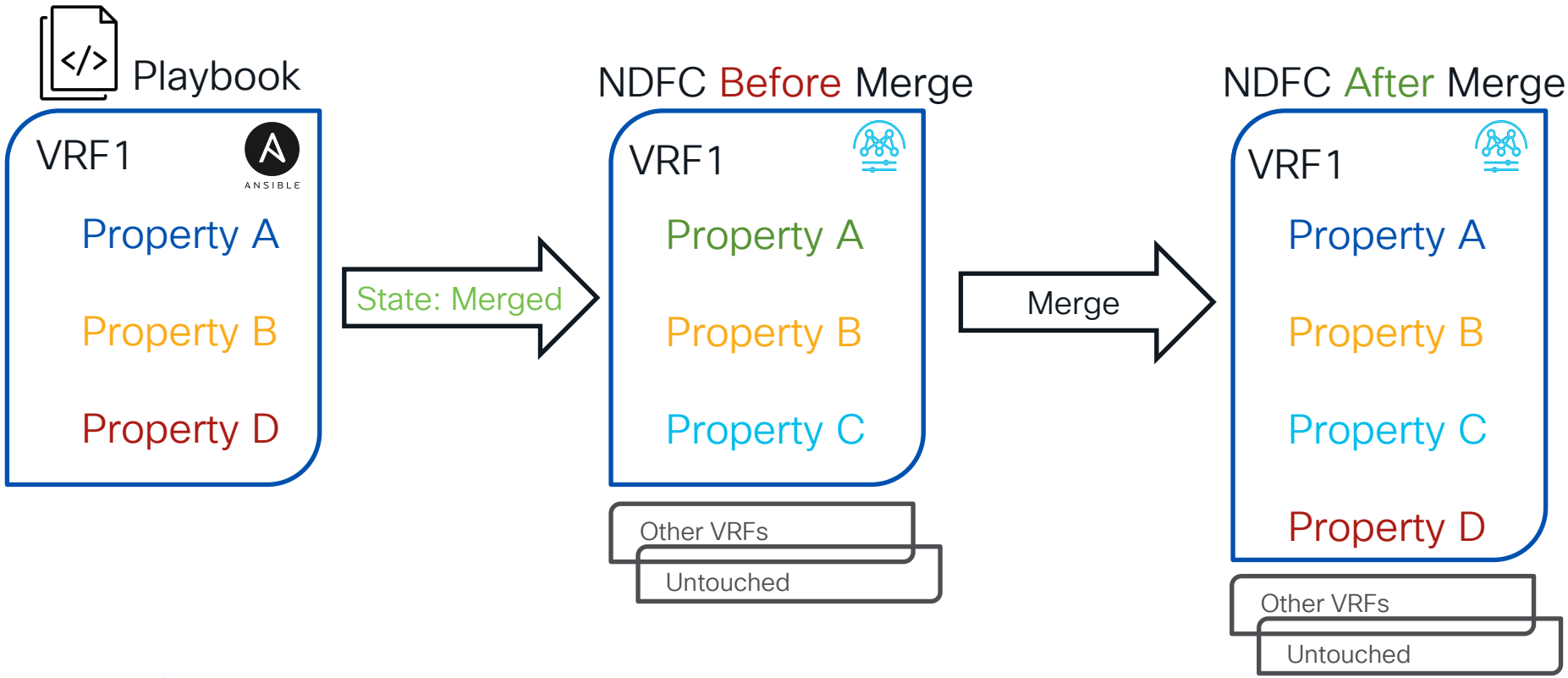
Merge Operation



Merged State



A Word About Module States - (Merged)

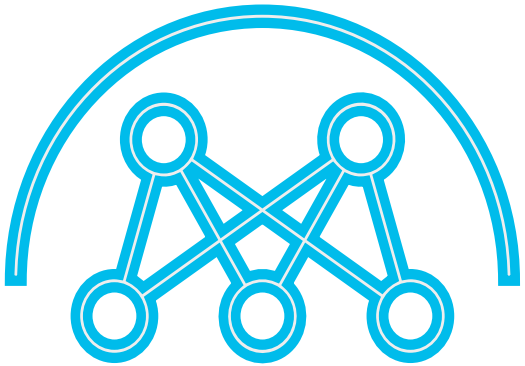


Who Is The Source Of Truth?

It's Me!

ANSIBLE

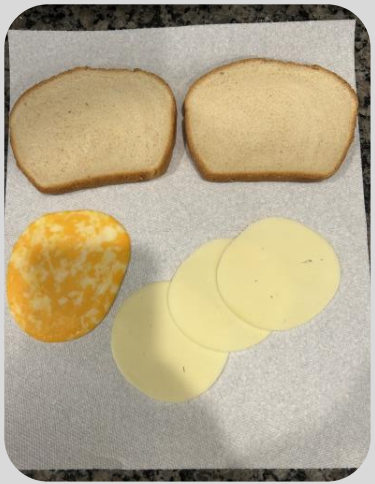
It's Me!



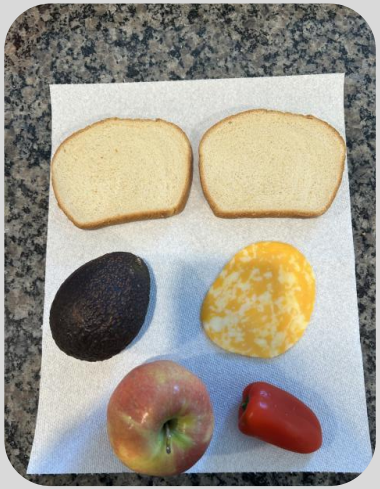
NDFC

How Does One Replace A Sandwhich?

Desired State

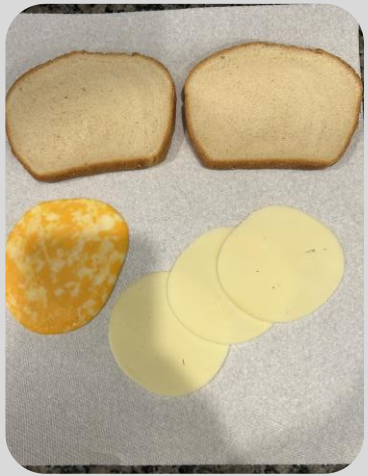


Actual State

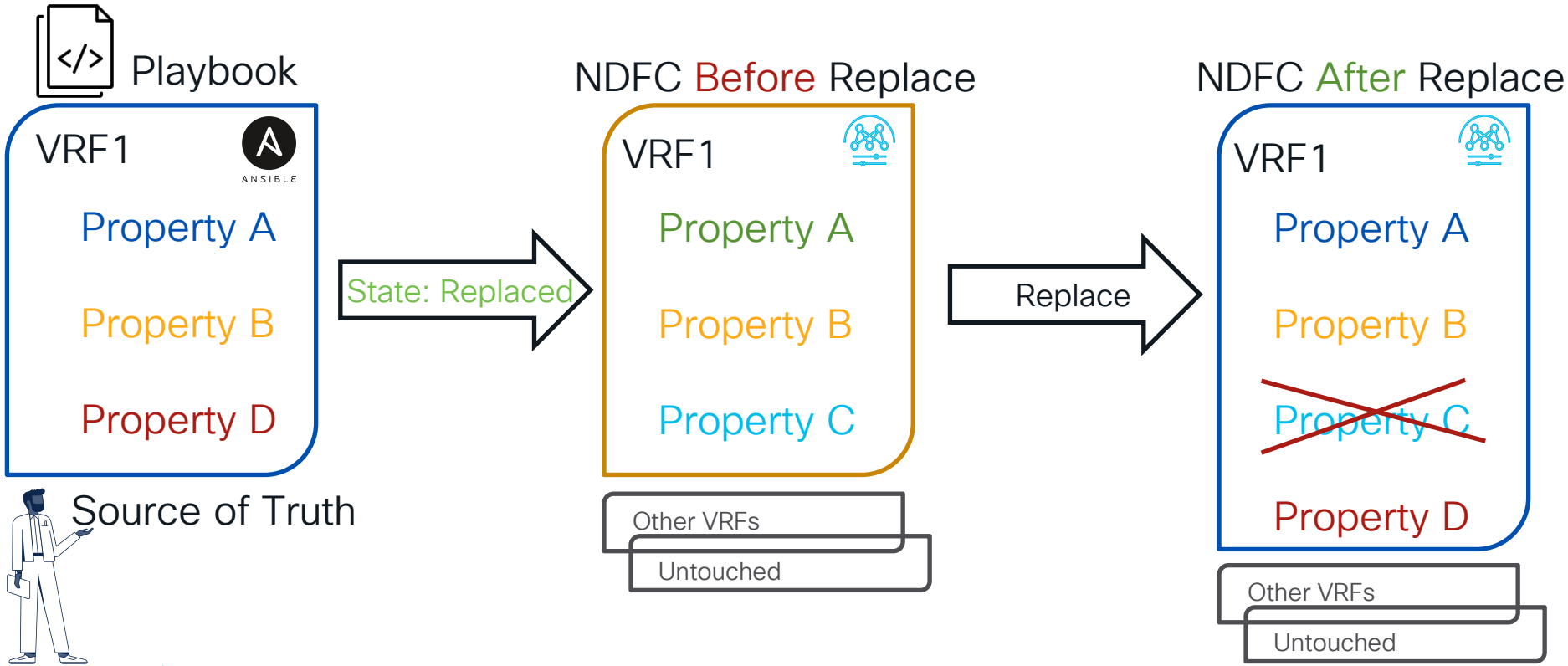


Replace Operation

Replaced State

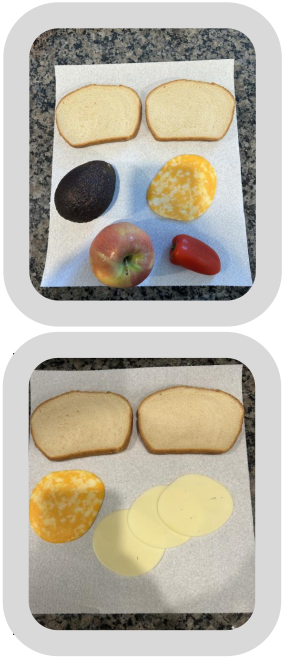


A Word About Module States - (Replaced)

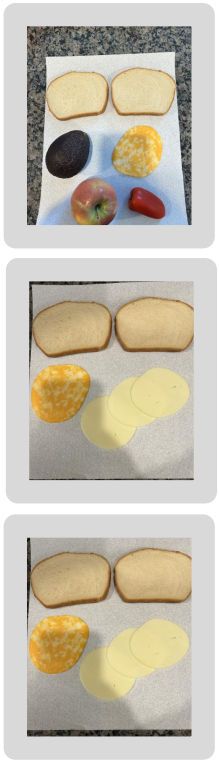


How Does One Override A Sandwhich?

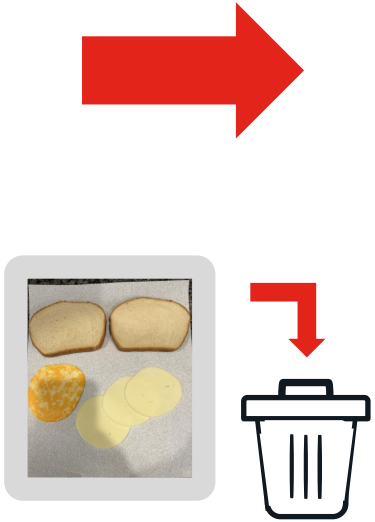
Desired State



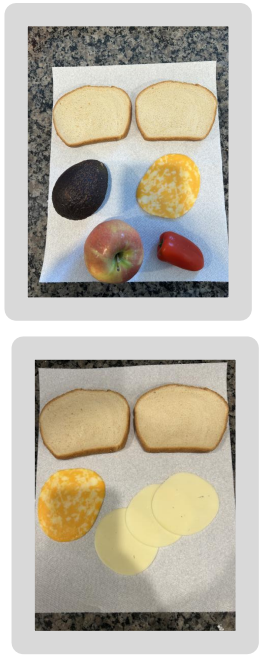
Actual State



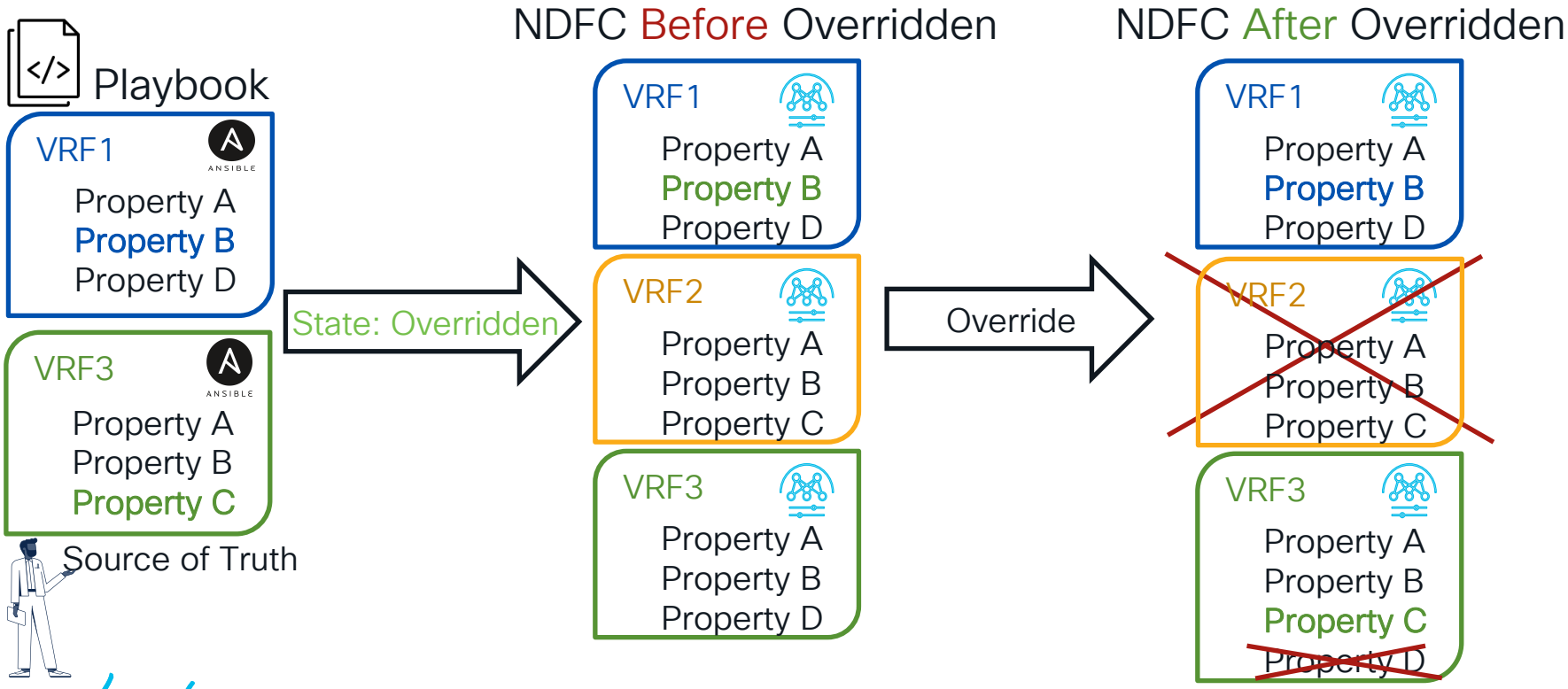
Override Operation



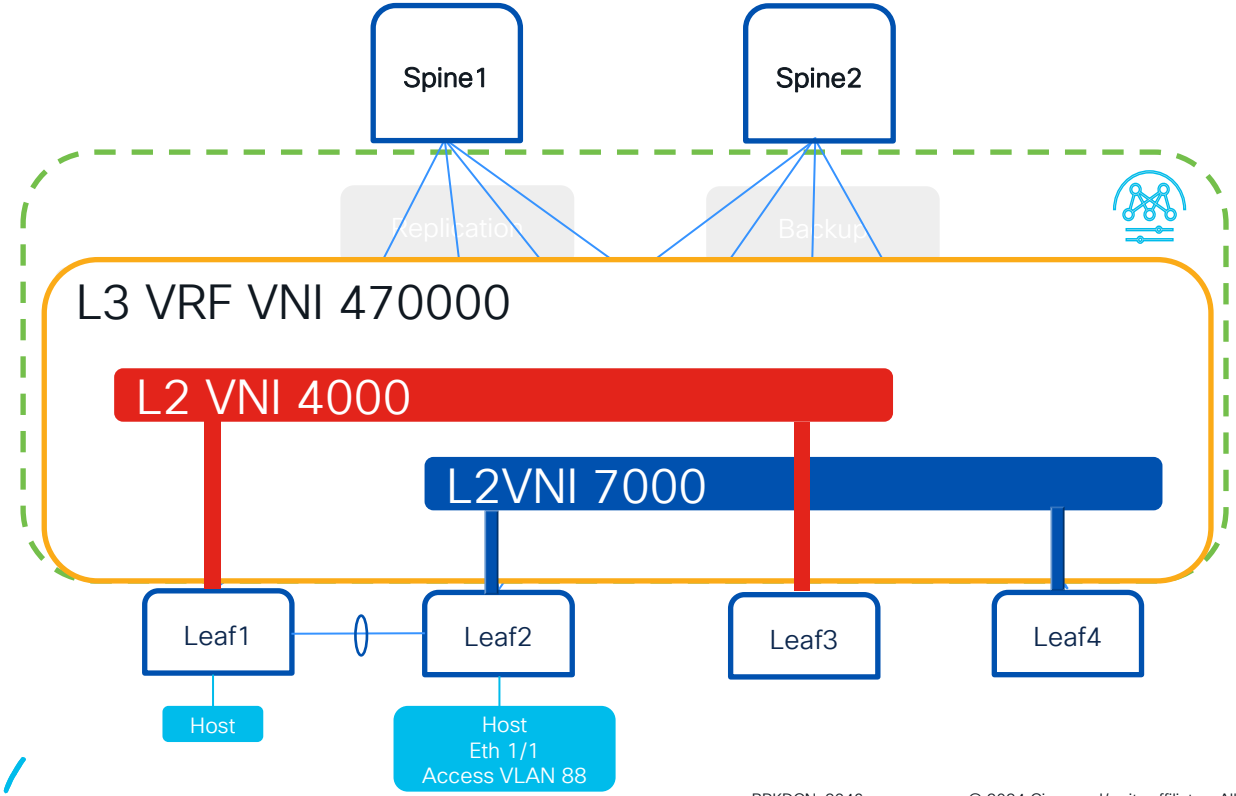
Overridden State



A Word About Module States - (Overridden)

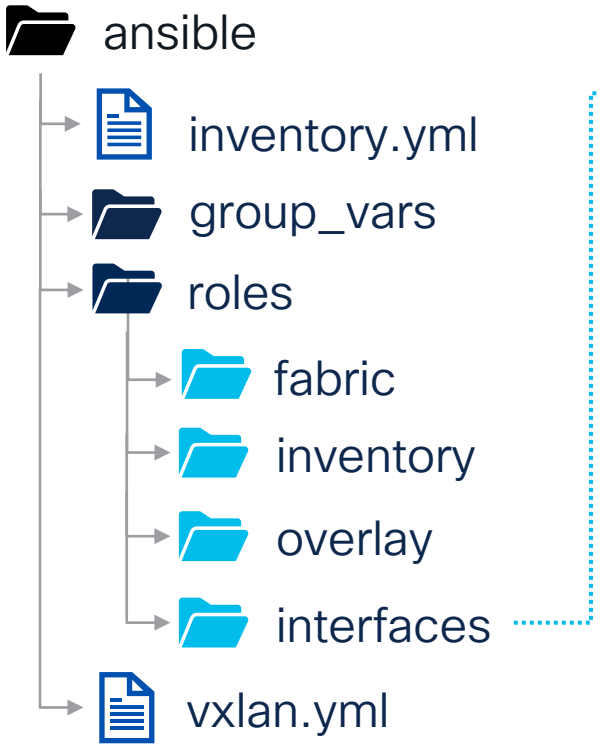


Workflow Step5: Configure Host Interfaces



How to do it

Ansible Roles – Tasks



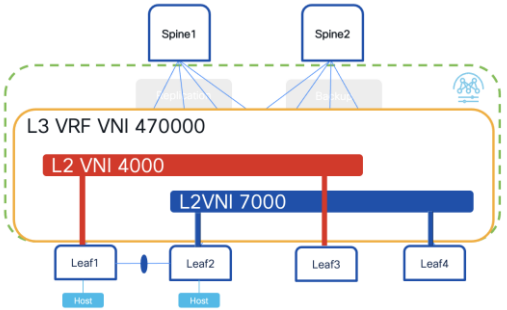
roles/interfaces/tasks/main.yml

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

- name: Add Overlay Networks
  cisco.dcnm.dcnm_interface:
    fabric: "{{ fabric.name }}"
    state: merged
    config:
      - name: eth1/1
        type: eth
        profile:
          admin_state: true
          mode: access
          access_vlan: 88
          <snip>
        switch:
          - 192.168.1.2
```

- Configure Access Ethernet Interface
- Deploy changes to leaf2

Configure Host Interfaces

```
- name: Add Networks
  cisco.dcnm.dcnm_interface:
    fabric: "{{ fabric_name }}"
    state: merged
    config:
      - name: "eth1/1"
        type: eth
        switch:
          - 192.168.1.2
        deploy: true
        profile:
          admin_state: true
          mode: access
          speed: 'Auto'
          bpdu_guard: false
          port_type_fast: true
          mtu: jumbo
          access_vlan: 88
          cmds:
            - no shutdown
          description: "VLAN 88 Access"
```

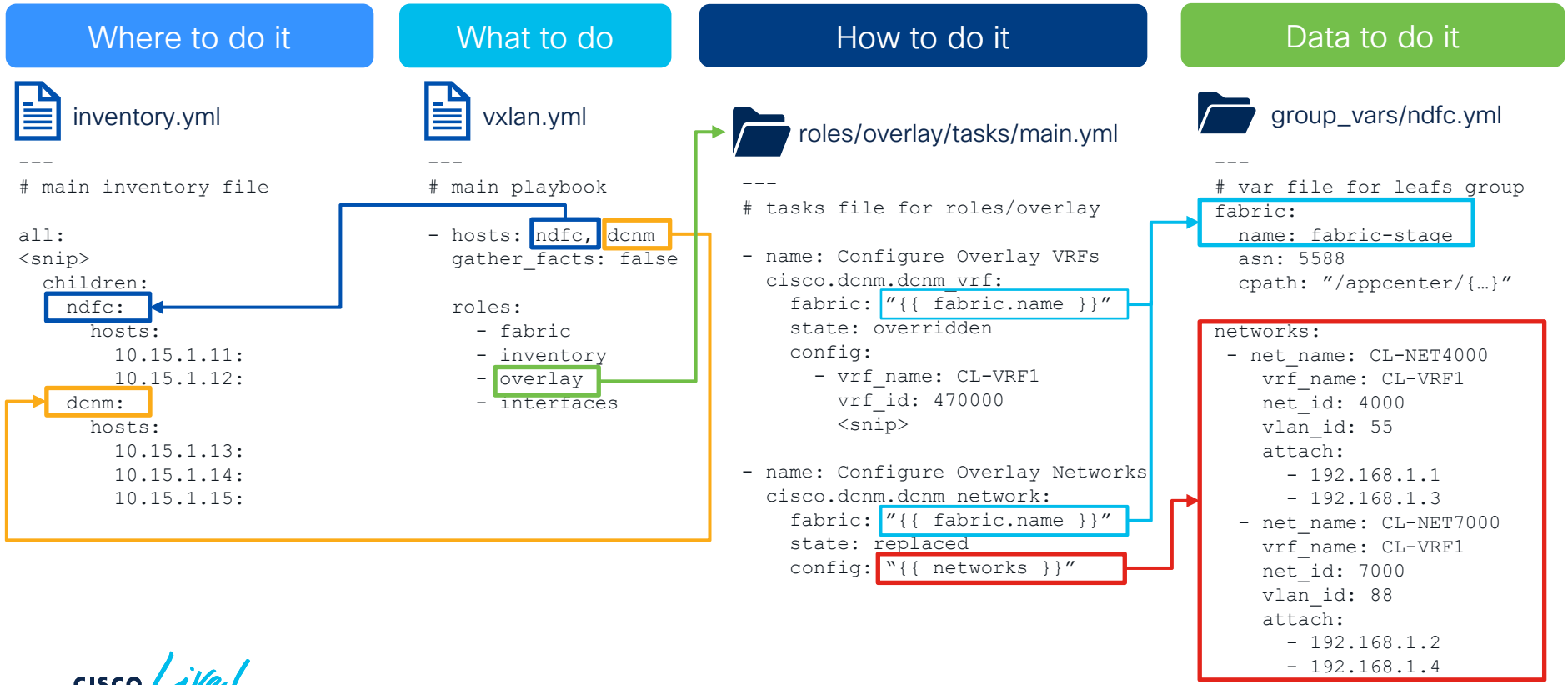


Invoke module call to cisco.dcnm.dcnm_interface

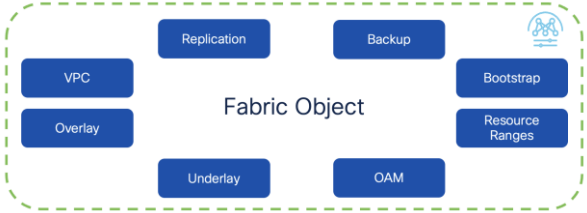
Configure and assign access vlan for host facing interfaces on leaf devices

cisco.dcnm.dcnm_interface

Putting It All Together



Create VXLAN EVPN Fabric



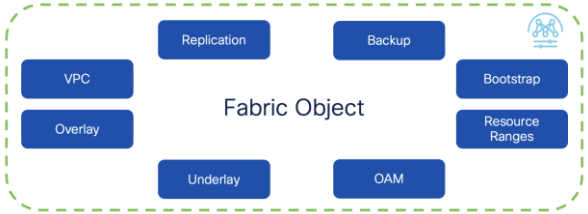
- Ansible Task Description

- Fabric Variables – Overrides Default Values
  - Set BGP Autonomous System
  - Green Field Fabric – Impacts Device Reload
  - Use CLI instead of Profiles

```
- name: Create Fabric {{ fabric.name }} on NDFC
  vars:
    payload:
      BGP_AS: "{{ fabric.asn }}"
      GRFIELD_DEBUG_FLAG: "Enable"
      OVERLAY_MODE: "cli"
```

cisco.dcnm.dcnm_rest

Create VXLAN EVPN Fabric

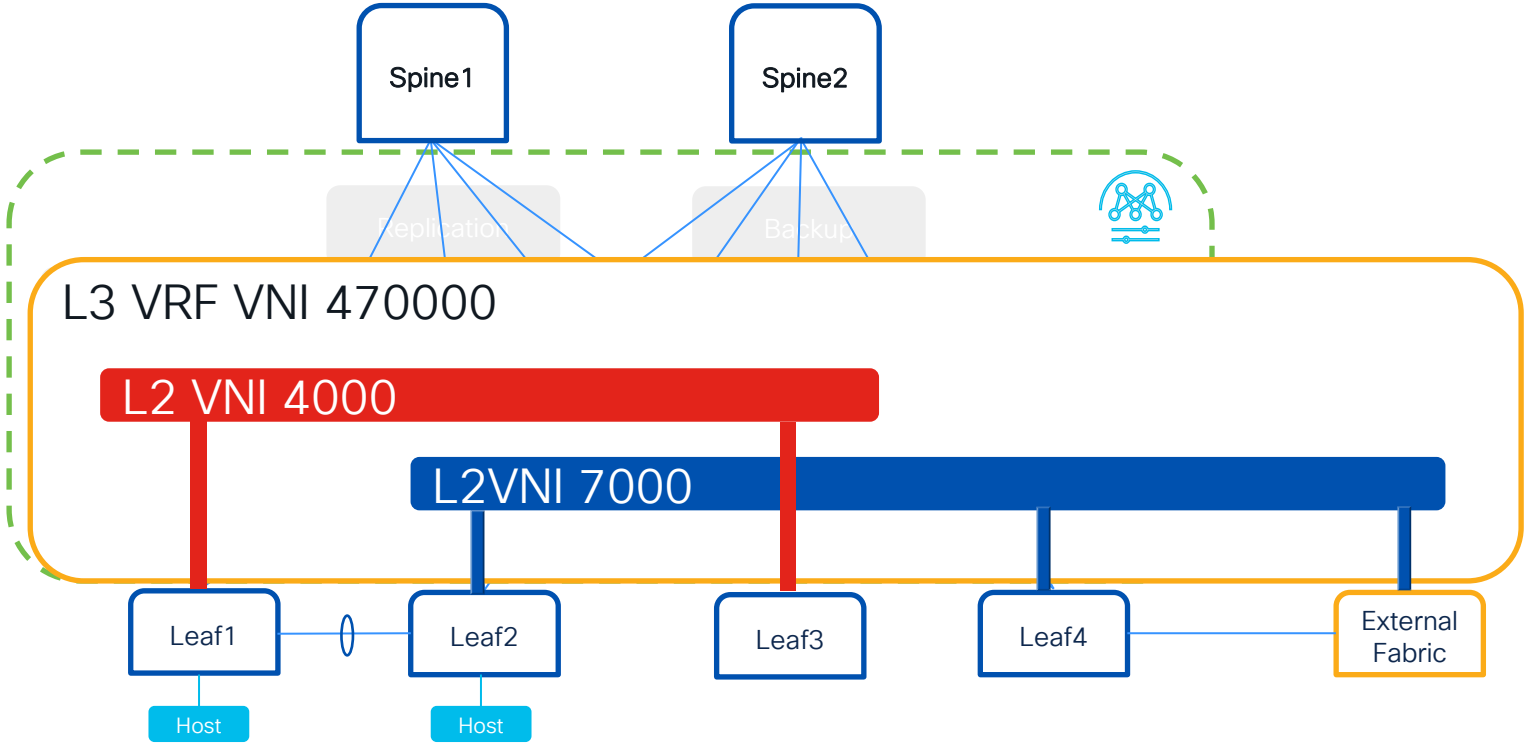


- Invoke module call to cisco.dcnm.dcnm_rest
  - Stop gap module – Invoke any NDFC REST API
  - Use when intent module not available
  - Fabric module on the roadmap
- Important Note: Does not check error conditions

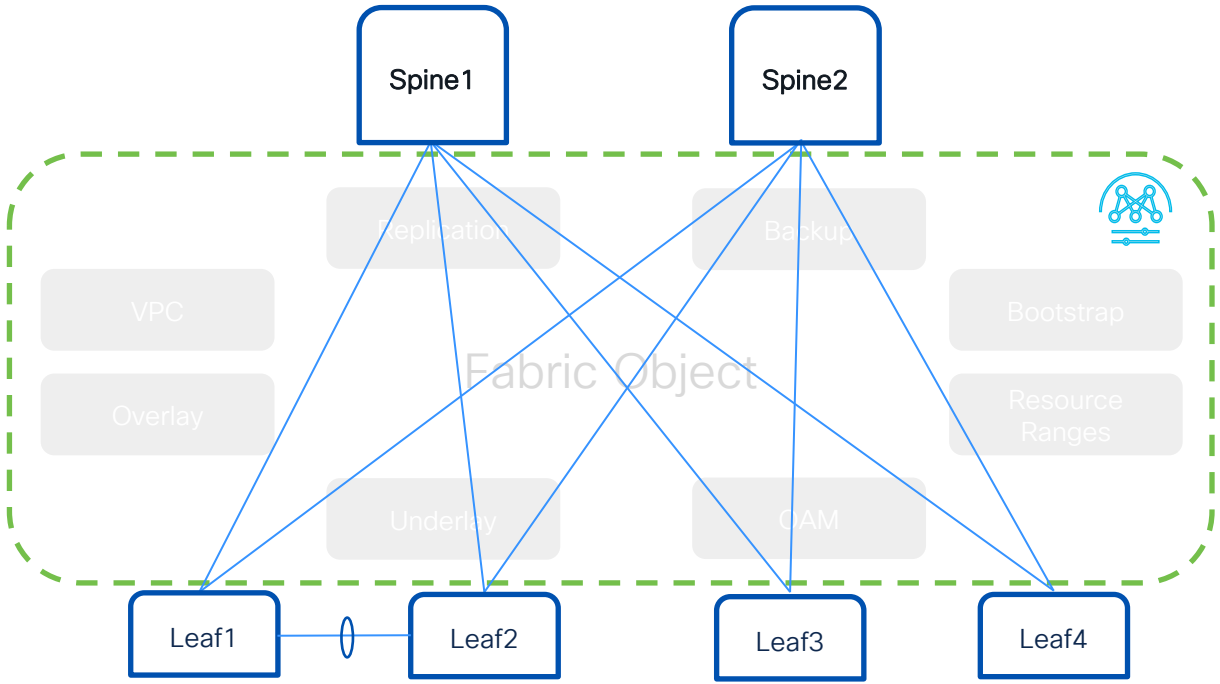
cisco.dcnm.dcnm_rest

```
- name: Create Fabric {{ fabric.name }} on NDFC
  vars:
    payload:
      BGP_AS: "{{ fabric.asn }}"
      GRFIELD_DEBUG_FLAG: "Enable"
      OVERLAY_MODE: "cli"
  cisco.dcnm.dcnm_rest:
    method: POST
    path: "/appcenter/cisco/ndfc/api/v1/lan-fabric/rest/control/fabrics/{{ fabric.name }}/Easy_Fabric"
    json_data: "{{ payload| to_json }}"
  ignore_errors: true
```

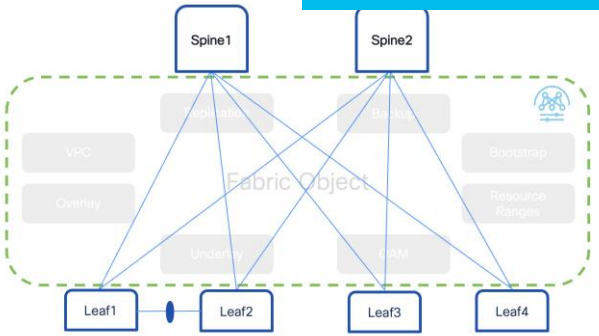
Workflow Step5: Extend The Fabric



Workflow Step6: Configure vPC



Enable vPC Peer Link



- Payload specifying each peer serial number
  - useVirtualPeerLink set to false – use physical link
- Invoke module call to cisco.dcnm.dcnm_rest
  - useVirtualPeerLink set to false
  - Use POST method and vpcpair path + Payload

```
- name: create VPC peer on leaf {{ leaf1_ip }}|{{ leaf_2.ip }}
  vars:
    payload:
      peerOneId: "{{leaf_sws.response[0].serialNumber}}"
      peerTwoId: "{{leaf_sws.response[1].serialNumber}}"
      useVirtualPeerlink: false
  cisco.dcnm.dcnm_rest:
    method: POST
    path: "/appcenter/cisco/ndfc/api/v1/lan-fabric/rest/vpcpair"
    json_data: "{{ payload | to_json }}"
  ignore_errors: true
```

cisco.dcnm.dcnm_rest

ANSIBLE

Configure vPC Interfaces

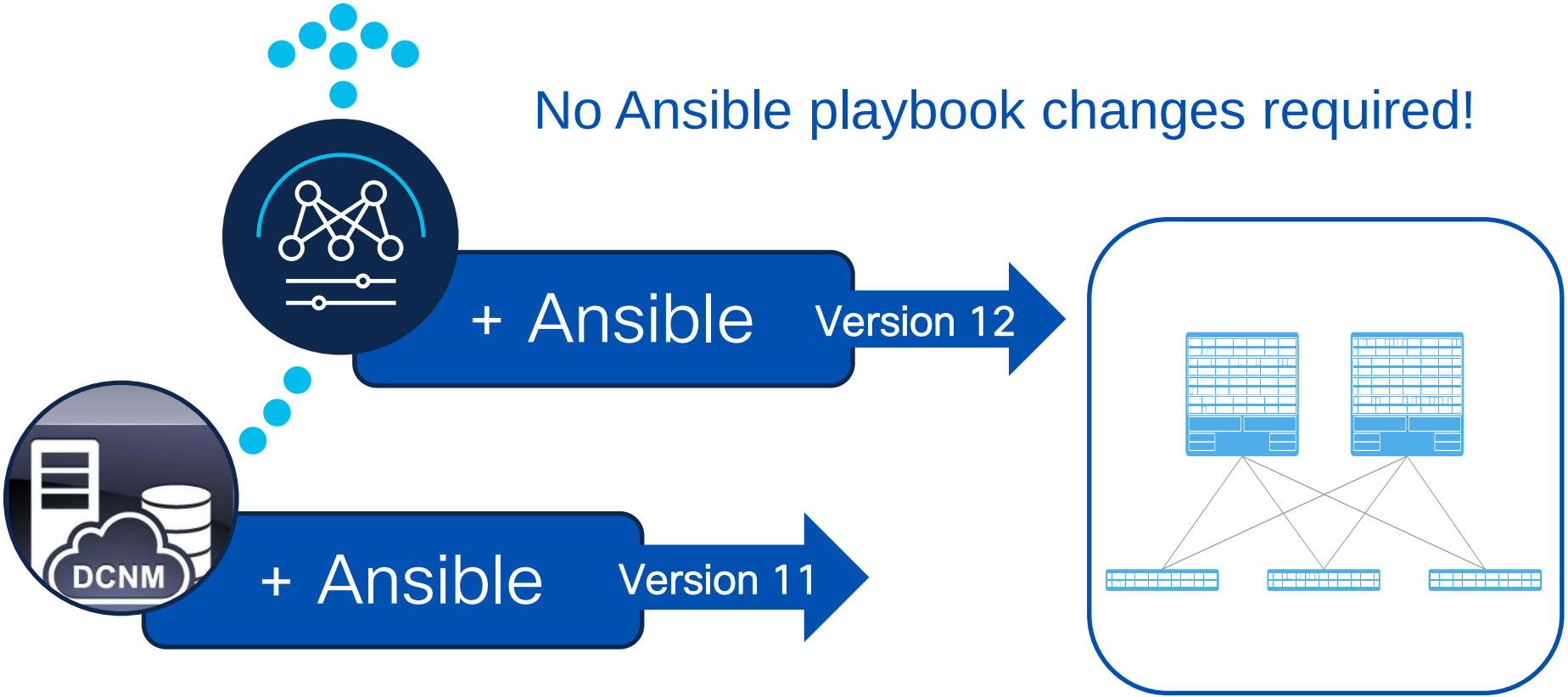
```
- name: Configure vPC Interfaces
  cisco.dcnm.dcnm_interface:
    fabric: "{{ fabric.name }}"
    state: merged
    config:
      - name: eth1/8
        type: eth
        admin_state: true
        switch:
          - "{{ leaf1.ip }}"
        profile:
          mode: trunk
      - name: eth1/9
        type: eth
        admin_state: true
        switch:
          - "{{ leaf2.ip }}"
        profile:
          mode: trunk
```

- Invoke module call to cisco.dcnm.dcnm_interface
  - Supported states: merged, replaced, overridden
- Supports a list of different types of interfaces
  - Ethernet, Loopback, FEX, PortChannel
  - SubInterfaces, VPC
- VPC Interfaces
  - Setup using Ethernet Interfaces
  - Specify Leaf IP
  - Use Trunk Mode Profile

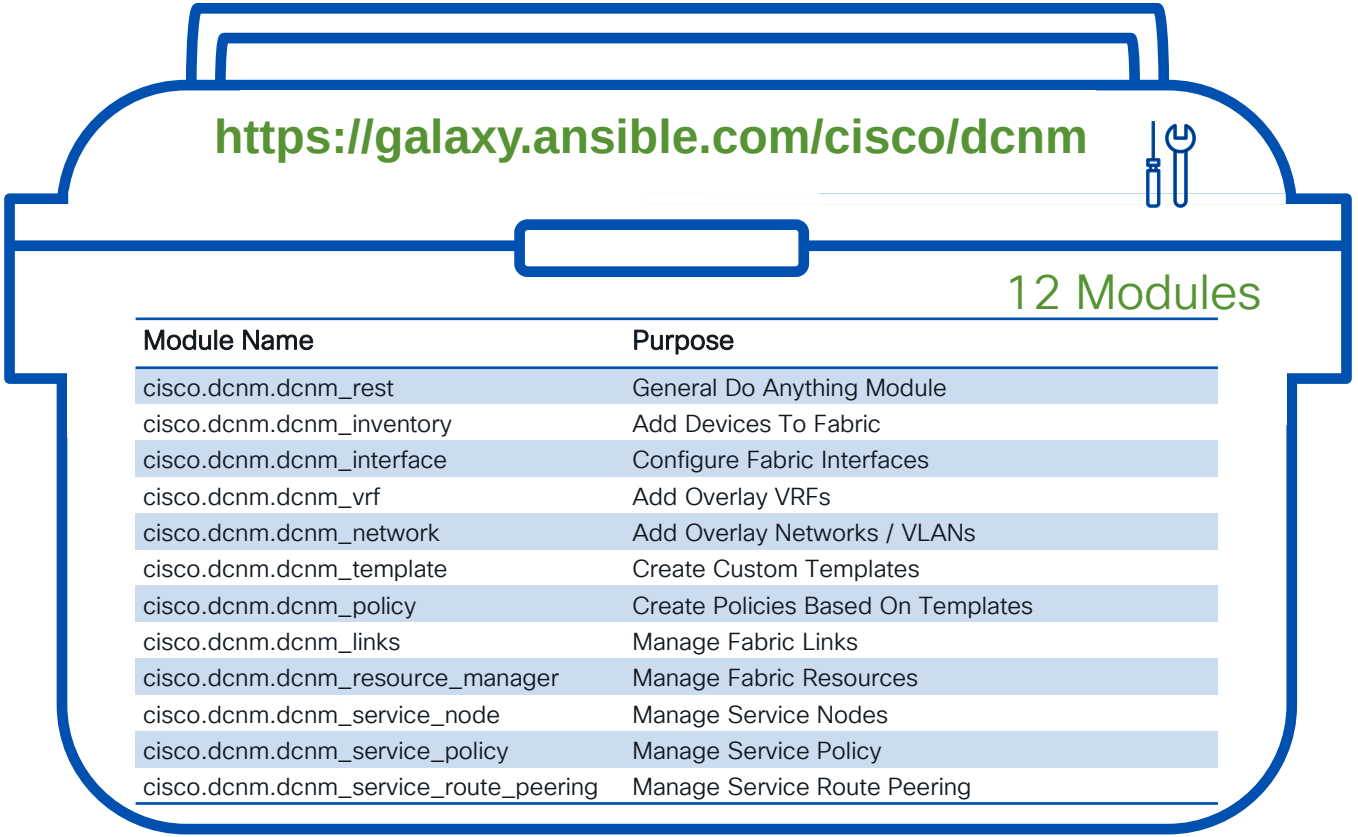
cisco.dcnm.dcnm_interface

Transparent NDFC/DCNM Controller Support

No Ansible playbook changes required!

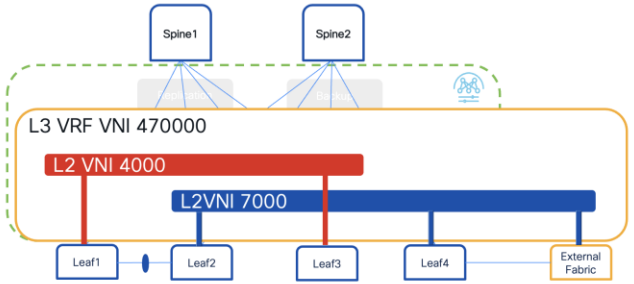


NDFC Collection Modules – Toolbox



Module Name	Purpose
cisco.dcnm.dcnm_rest	General Do Anything Module
cisco.dcnm.dcnm_inventory	Add Devices To Fabric
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_service_node	Manage Service Nodes
cisco.dcnm.dcnm_service_policy	Manage Service Policy
cisco.dcnm.dcnm_service_route_peering	Manage Service Route Peering

VRF-Lite Support



- Attach Using Specfic Device for VRF-Lite
- Specify Peer VRF and Network Parameters

cisco.dcnm.dcnm_vrf

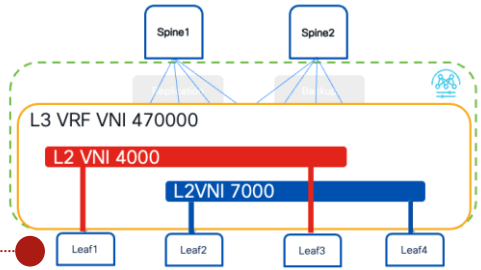
```
- name: Add VRFs to Fabric
  cisco.dcnm.dcnm_vrf:
    { … }
    config:
      { … }
      attach:
       - 192.168.0.1
       - 192.168.0.2
         vrf_lite:
           - peer_vrf: CL_VRF1 # peer_vrf is mandatory
             interface: Ethernet1/16 # optional
             ipv4_addr: 10.33.0.2/30 # optional
             neighbor_ipv4: 10.33.0.1 # optional
             ipv6_addr: 2010::10:34:0:7/64 # optional
             neighbor_ipv6: 2010::10:34:0:3 # optional
             dot1q: 2 # dot1q can be got from dcnm/optional
```

Custom Templates and Policies

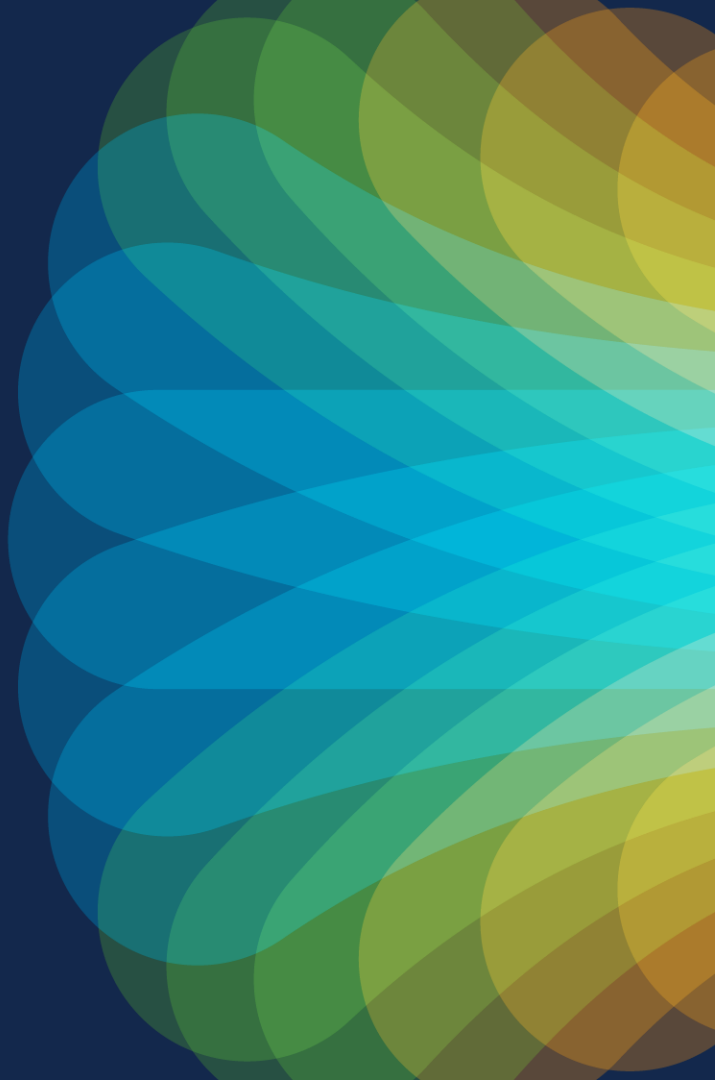
```
- name: Create Telemetry Template
  cisco.dcnm.dcnm_template:
    state: merged | deleted | query
    config:
      - name: template_telemetry
        description: "Telemetry_Base_Template"
        tags: "internal policy 101"
       content: | (config content follows)
```

```
- name: Create Telemetry Policy
  cisco.dcnm.dcnm_policy:
    fabric: "{{ ndfc_fabric_name }}"
    state: merged | deleted | query
    deploy: true
    config:
      - name: template_telemetry
        create_additional_policy: false
        priority: 101
      - switch:
        - ip: "{{ device_leaf1}}"
```

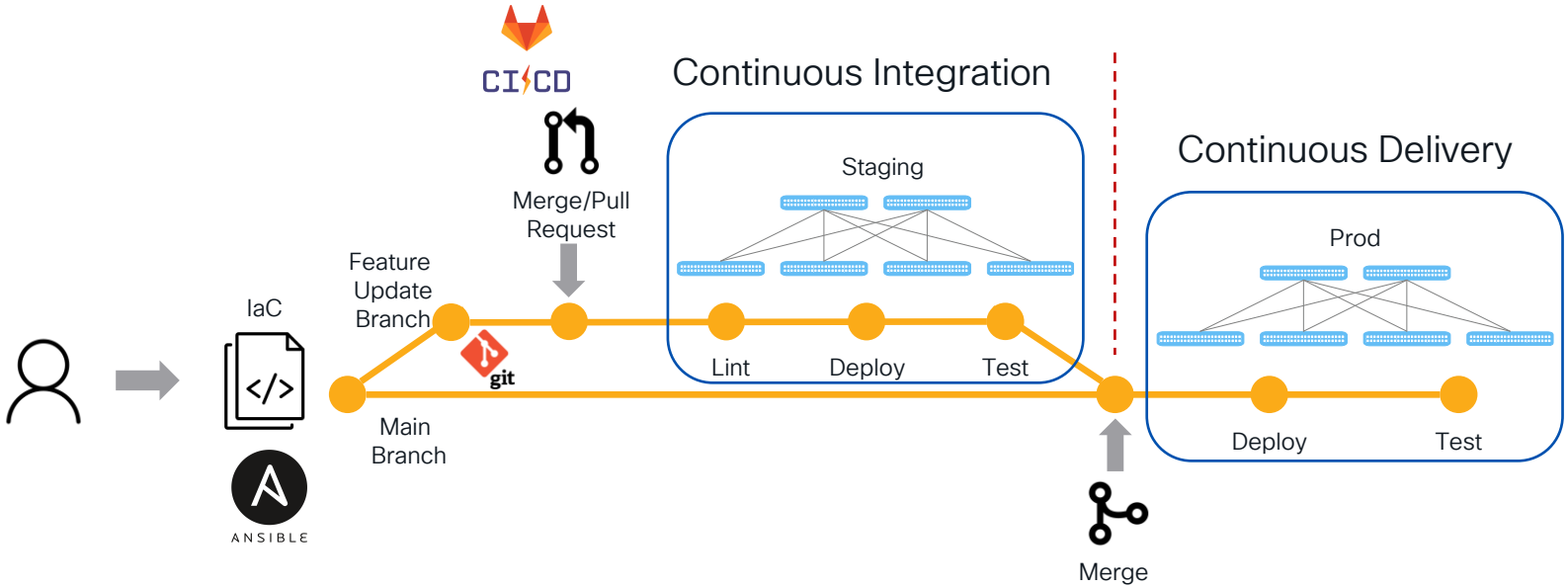
```
content: |
    telemetry
      certificate /bootflash/telegraf.crt telegraf
      destination-profile
        use-vrf management
      destination-group 101
        ip address 10.195.225.176 port 57101 protocol gRPC…
      sensor-group 101
        data-source DME
        path sys/ch depth unbounded
      subscription 101
        dst-grp 101 snsr-grp 101 sample-interval 10101
```



CI/CD Pipeline Demo



NDFC & GitLab



NetDevOps!

References to Start Your Journey

Ansible for NXOS and NDFC

- Install Ansible and cisco.nxos and cisco.dcnm Ansible Collections
- Session Code (https://github.com/mtarking/BRKDCN-2946)
  - Full Examples from the Session and More
- Play with module examples

  https://github.com/ansible-collections/cisco.nxos/blob/main/docs/cisco.nxos.nxos_bgp_module.rst#examples

  https://github.com/CiscoDevNet/ansible-dcnm/blob/develop/docs/cisco.dcnm.dcnm_inventory_module.rst#examples
- Play with roles (https://github.com/allenrobel/ndfc-roles)
- Think big ..... Start small

More information – Other sessions/labs

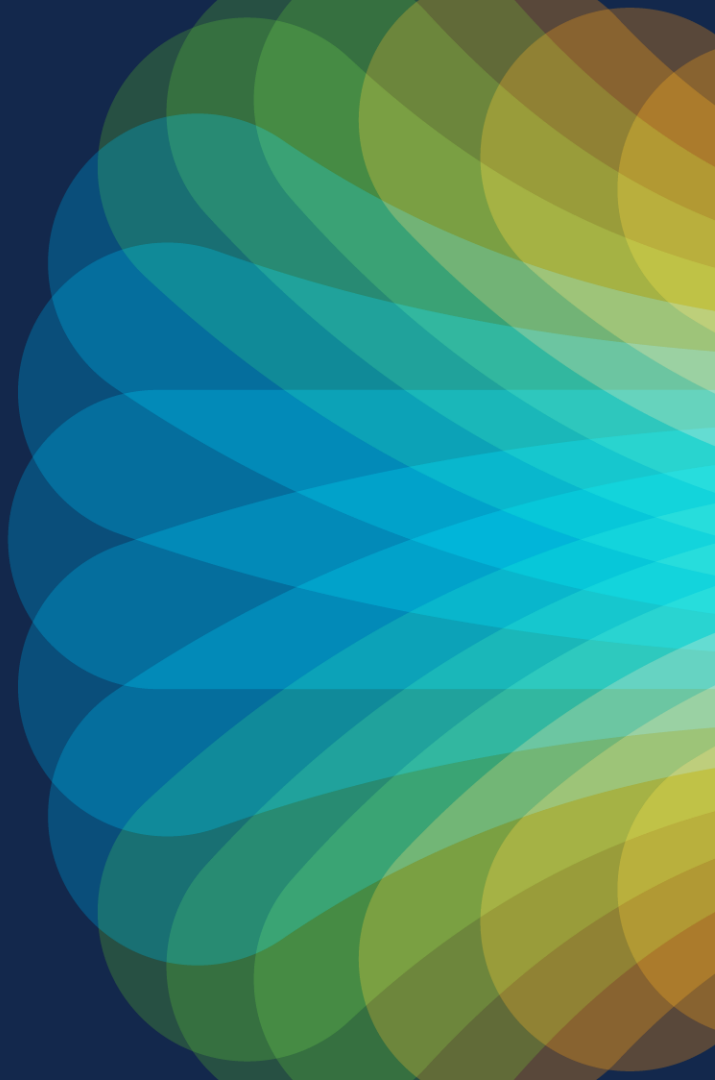
- DEVWKS-3928: Build VXLAN Fabric with NDFC and Ansible
- LABDCN-2229 (Ansible for VXLAN EVPN Fabric Automation)
- DEVNET-2117 (CI/CD Pipelines for Infrastructure Automation)
- BRKDCN-2929 (Simple VXLAN/EVPN Fabric Setup with Nexus Dashboard)
- BRKDCN-1619 (Introduction to NDFC: Simplifying Management of Your Data Center)
- BRKDCN-2988 (Design, Automate, and Manage Next-Gen Data Center VXLAN BGP EVPN Fabric with NDFC)

More information – Ansible

- https://www.ansible.com/resources/get-started
- https://docs.ansible.com/ansible/latest/collections_guide/index.html
- https://galaxy.ansible.com/cisco/dcnm
- https://galaxy.ansible.com/cisco/nxos
- https://developer.cisco.com/docs/nexus-as-code/#!nx-os-with-ansible
- https://developer.cisco.com/docs/nexus-as-code/#!ndfc-with-ansible

The bridge to possible

Thank you



cisco Live!


cisco Live!
Let's go