Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat
with the speaker during and/or after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Follow the instructions on the screen

Webex Teams will be moderated
by the speaker until November 15, 2019.

Download Webex Teams

webex.com/download

↓

cs.co/track-6

Arjan Toxopeus
Systems Architect DevNet

Updated May 2017

CISCO

DEVNET express

LISTEN > LEARN >> PUT IT INTO PRACTICE

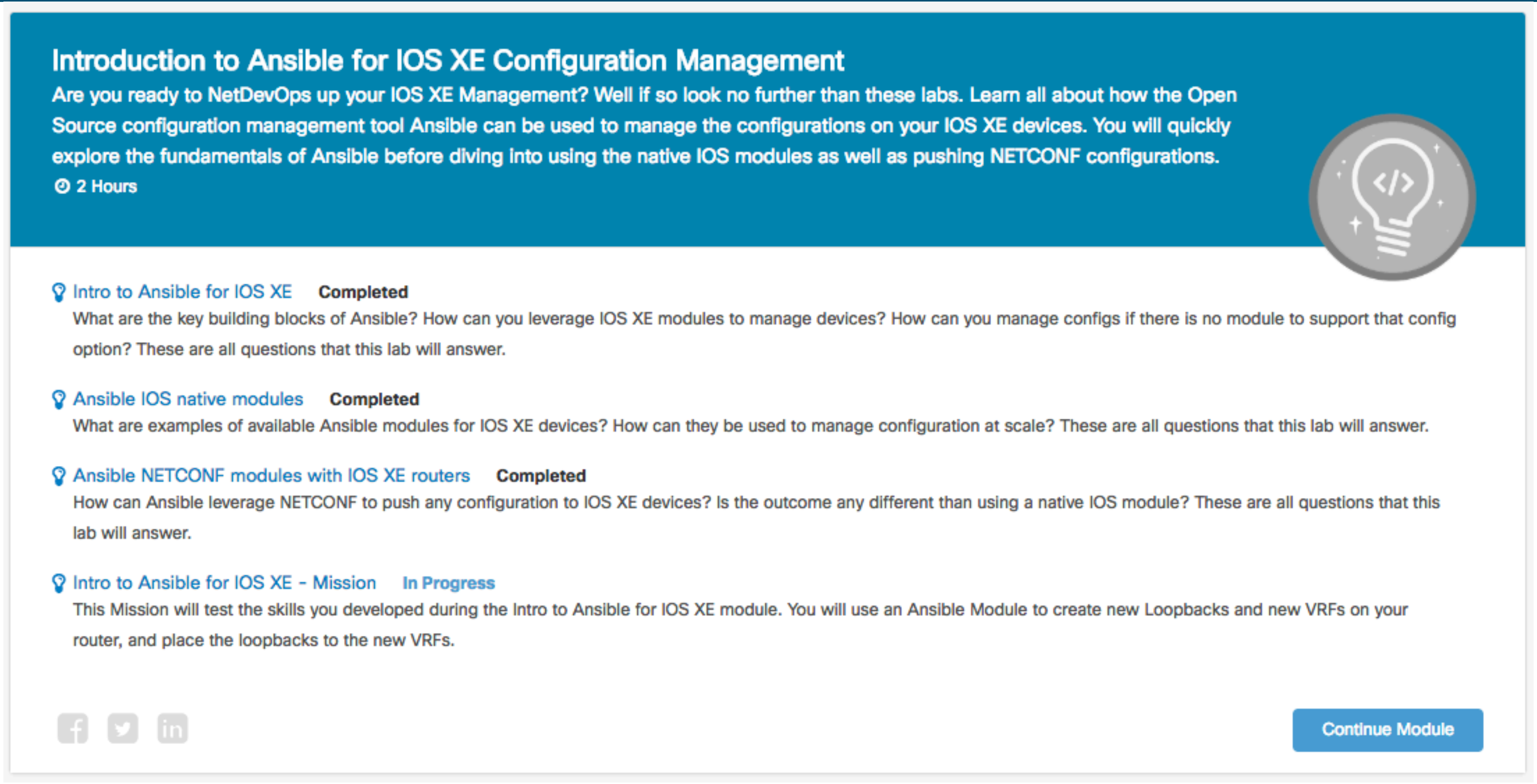
Introduction to Ansible for IOS XE Configuration Management

Arjan Toxopeus
Systems Architect

11-November-2019

ALL ABOUT DIGITAL NETWORK ARCHITECTURE

https://learninglabs.cisco.com/tracks/dnav3-track



Introduction to Ansible for IOS XE Configuration Management

Are you ready to NetDevOps up your IOS XE Management? Well if so look no further than these labs. Learn all about how the Open Source configuration management tool Ansible can be used to manage the configurations on your IOS XE devices. You will quickly explore the fundamentals of Ansible before diving into using the native IOS modules as well as pushing NETCONF configurations.

2 Hours

Intro to Ansible for IOS XE **Completed**

What are the key building blocks of Ansible? How can you leverage IOS XE modules to manage devices? How can you manage configs if there is no module to support that config option? These are all questions that this lab will answer.

Ansible IOS native modules **Completed**

What are examples of available Ansible modules for IOS XE devices? How can they be used to manage configuration at scale? These are all questions that this lab will answer.

Ansible NETCONF modules with IOS XE routers **Completed**

How can Ansible leverage NETCONF to push any configuration to IOS XE devices? Is the outcome any different than using a native IOS module? These are all questions that this lab will answer.

Intro to Ansible for IOS XE - Mission **In Progress**

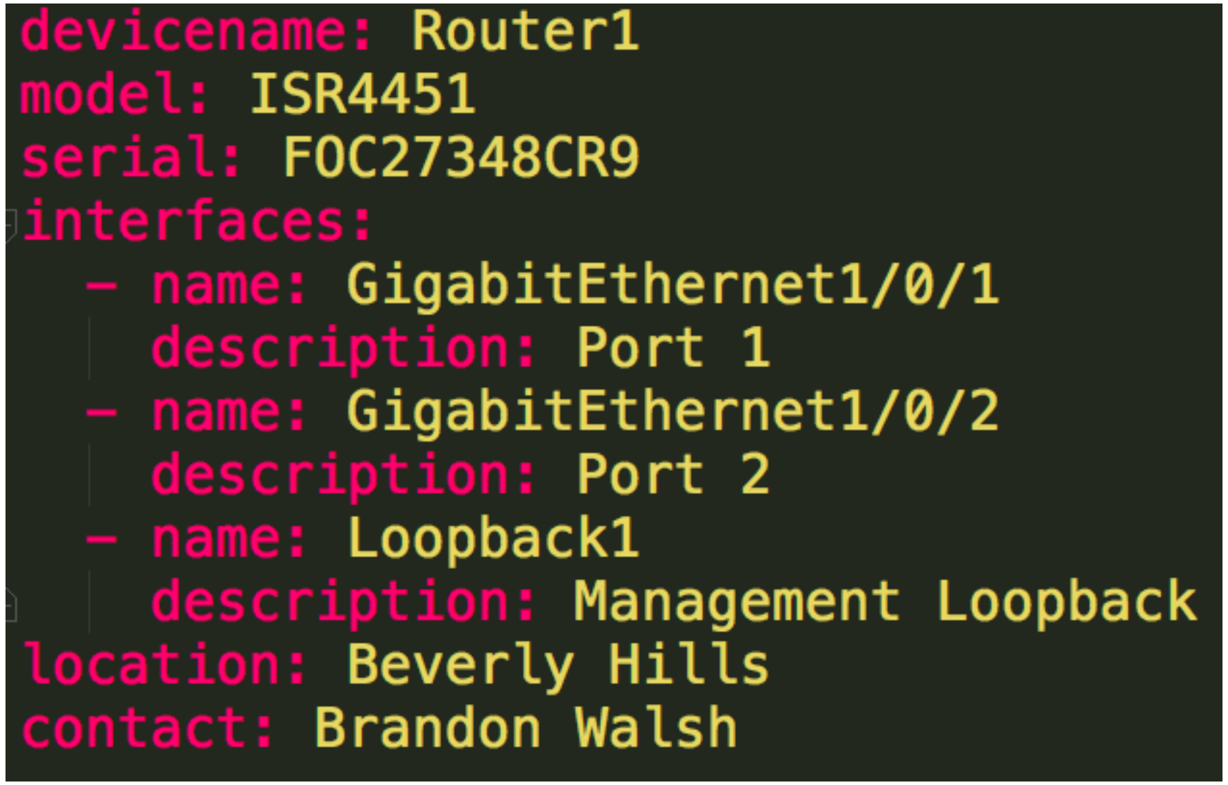
This Mission will test the skills you developed during the Intro to Ansible for IOS XE module. You will use an Ansible Module to create new Loopbacks and new VRFs on your router, and place the loopbacks to the new VRFs.

YAML Overview

What is YAML

- What is YAML?
  - "YAML Ain't Markup Language"
  - YAML is a human readable data serialization language
    - YAML files are easily parsed into software data structures
  - YAML is a common basis for a number of domain specific languages
    - Ansible
    - Heat
    - Saltstack
    - cloud-init
    - Many more!
  - In this module we will focus on YAML as it relates to Ansible

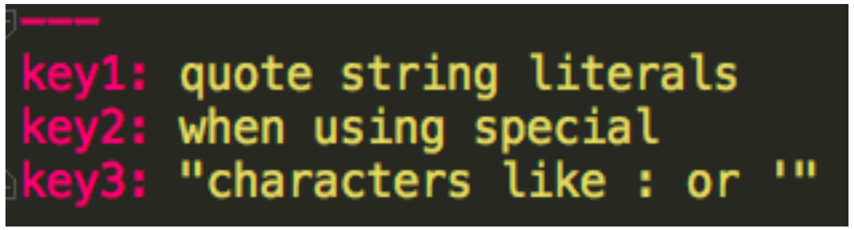
Sample YAML data structure



```yaml
devicename: Router1
model: ISR4451
serial: FOC27348CR9
interfaces:
  - name: GigabitEthernet1/0/1
    description: Port 1
  - name: GigabitEthernet1/0/2
    description: Port 2
  - name: Loopback1
    description: Management Loopback
location: Beverly Hills
contact: Brandon Walsh
```

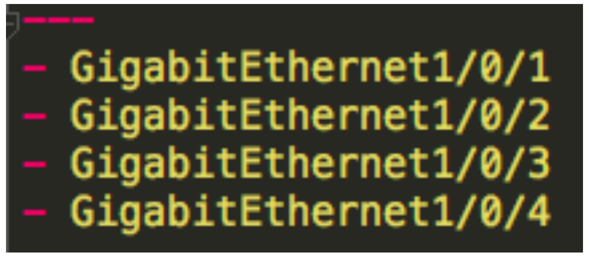
- This is an example of how common router attributes could be described in YAML format
- We will drill into the specific data types shown in more detail, but this illustrates how the hierarchy of an infrastructure device can be modeled in YAML

Scalars, sequences, and mappings



```
---
key1: quote string literals
key2: when using special
key3: "characters like : or '"
```

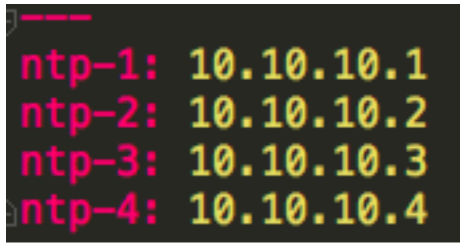
YAML scalars typically do not need quotes, except when containing special chars



```
---
- GigabitEthernet1/0/1
- GigabitEthernet1/0/2
- GigabitEthernet1/0/3
- GigabitEthernet1/0/4
```

Sequence items start with a '-'

YAML sequences become Python lists

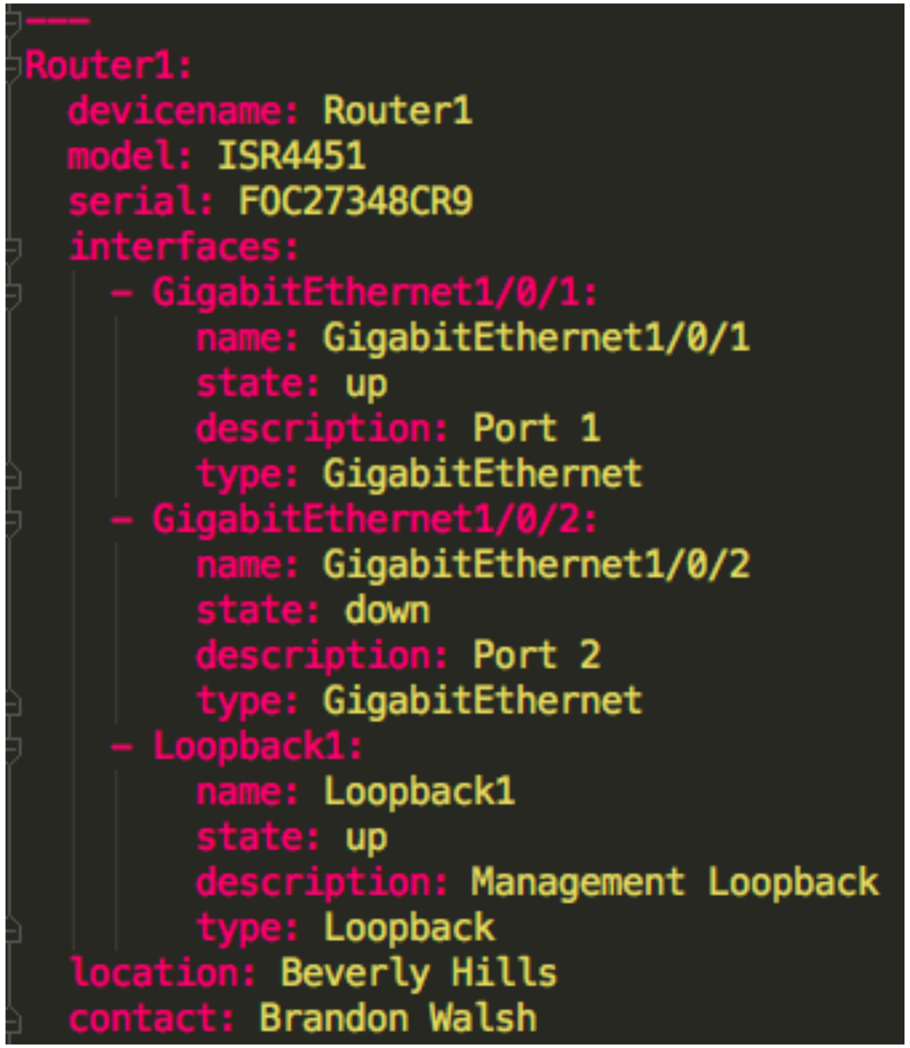


```
---
ntp-1: 10.10.10.1
ntp-2: 10.10.10.2
ntp-3: 10.10.10.3
ntp-4: 10.10.10.4
```

Mappings use a ':' to separate the key and value being mapped

YAML mappings become Python dictionaries

Nested Data Structures



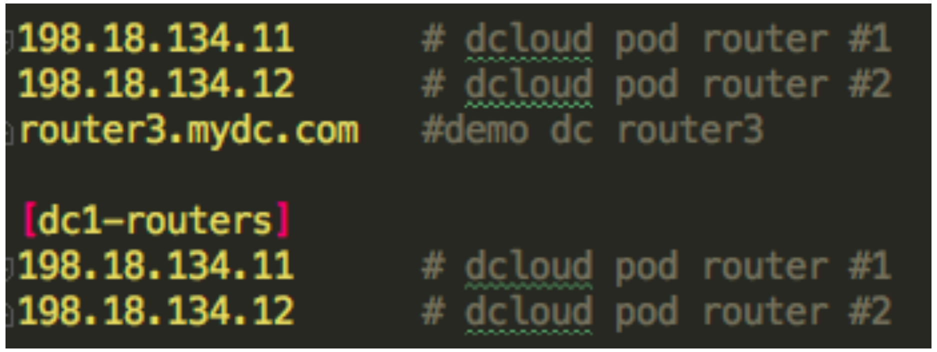
```
---
Router1:
  devicename: Router1
  model: ISR4451
  serial: FOC27348CR9
  interfaces:
    - GigabitEthernet1/0/1:
        name: GigabitEthernet1/0/1
        state: up
        description: Port 1
        type: GigabitEthernet
    - GigabitEthernet1/0/2:
        name: GigabitEthernet1/0/2
        state: down
        description: Port 2
        type: GigabitEthernet
    - Loopback1:
        name: Loopback1
        state: up
        description: Management Loopback
        type: Loopback
  location: Beverly Hills
  contact: Brandon Walsh
```

- YAML is sensitive to white space
- Introducing white space within a sequence or mapping creates a nested data structure
- This example contains a mapping for Router1
- Router1 contains mappings of several top level attributes.
- Attribute interfaces contains a sequence of interfaces
- Each interface item contains mappings of specific attributes

Ansible Building Blocks

Ansible Inventory File – Hosts & Groups

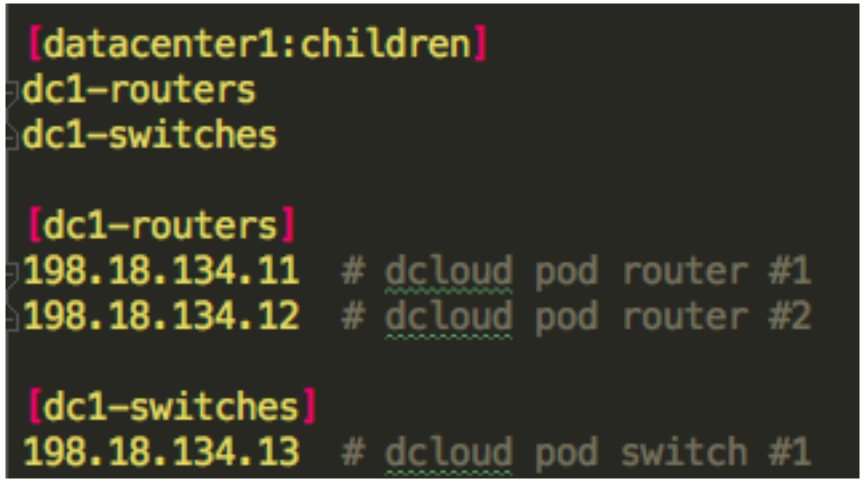
- Inventory files identify hosts, and groups of hosts under management
  - Host definitions can be by IP or FQDN
  - Group names are enclosed in []
- As seen below, 2 hosts are defined individually by IP, and one by FQDN
- Router1 and Router2 are also defined as part of group ‘dc1-routers’



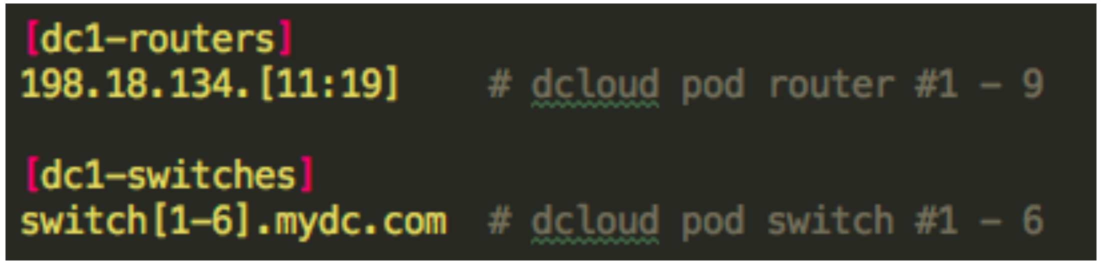
```
198.18.134.11       # dcloud pod router #1
198.18.134.12       # dcloud pod router #2
router3.mydc.com    #demo dc router3

[dc1-routers]
198.18.134.11       # dcloud pod router #1
198.18.134.12       # dcloud pod router #2
```

Ansible Inventory File – Nested Groups & Ranges

- Ansible supports nesting of groups, such as the below example showing group ‘datacenter1’ containing child groups ‘dc1-routers’ and ‘dc1-switches’
- Ansible also supports grouping of similar purpose hosts by defining a range.
  - Routers 1-9 are grouped based on a range of IPs
  - Switches 1-6 are grouped based on a range of FQDNs
  - Ranges save users from listing out every individual host for a group of similar hosts



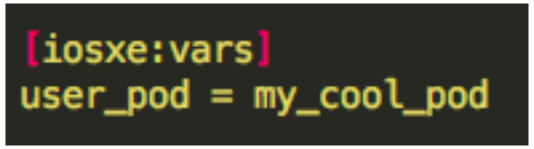
```
[datacenter1:children]
dc1-routers
dc1-switches

[dc1-routers]
198.18.134.11  # dcloud pod router #1
198.18.134.12  # dcloud pod router #2

[dc1-switches]
198.18.134.13  # dcloud pod switch #1
```



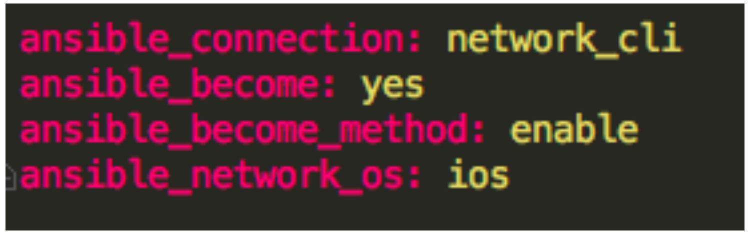
```
[dc1-routers]
198.18.134.[11:19]     # dcloud pod router #1 - 9

[dc1-switches]
switch[1-6].mydc.com   # dcloud pod switch #1 - 6
```

Ansible Variables

- Ansible supports defining variables in a number of ways. For this lab we will cover inventory file variables and group_vars files
- In your lab, your hosts file sets the variable ‘user_pod’ for group iosxe as shown below on the left.
  - In your lab you will set this variable to a unique string
- In your lab, there is also a file ‘iosxe.yml’ within the ‘group_vars’ folder
  - As shown below on the right, this file contains additional variables for hosts in the ‘iosxe’ group, which Ansible will read in automatically upon execution.



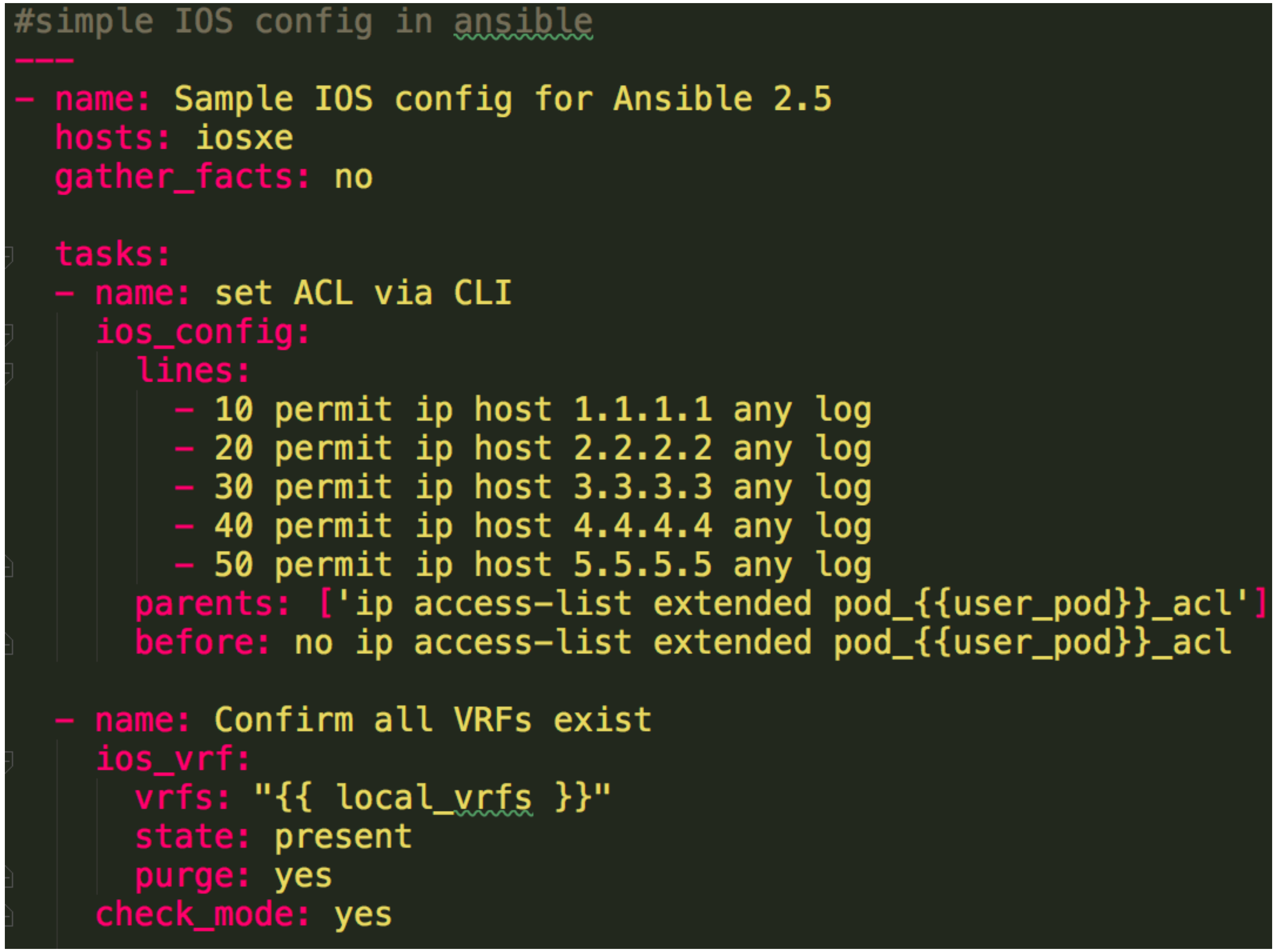
```
[iosxe:vars]
user_pod = my_cool_pod
```



```
ansible_connection: network_cli
ansible_become: yes
ansible_become_method: enable
ansible_network_os: ios
```

Understand a sample playbook structure

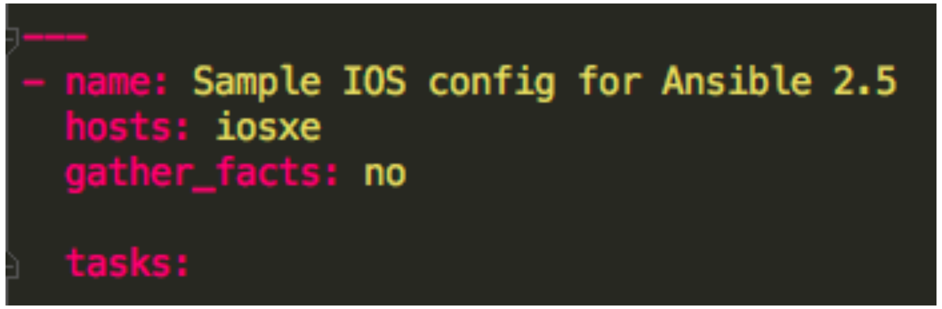
Example Ansible Playbook



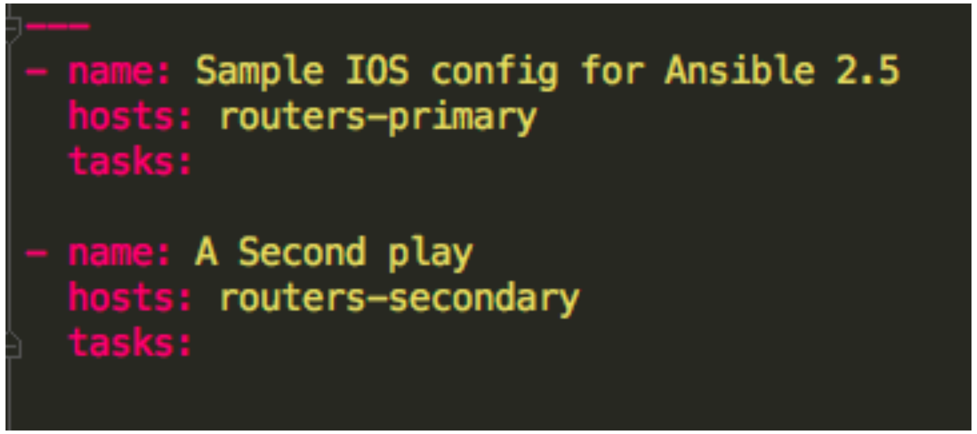
```
#simple IOS config in ansible
---
- name: Sample IOS config for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:
  - name: set ACL via CLI
    ios_config:
      lines:
        - 10 permit ip host 1.1.1.1 any log
        - 20 permit ip host 2.2.2.2 any log
        - 30 permit ip host 3.3.3.3 any log
        - 40 permit ip host 4.4.4.4 any log
        - 50 permit ip host 5.5.5.5 any log
      parents: ['ip access-list extended pod_{{user_pod}}_acl']
      before: no ip access-list extended pod_{{user_pod}}_acl

  - name: Confirm all VRFs exist
    ios_vrf:
      vrfs: "{{ local_vrfs }}"
      state: present
      purge: yes
    check_mode: yes
```

- Playbooks are YAML documents that contain a set of tasks to be performed by Ansible.
- We will break this down into its components in the upcoming slides, but observe how it is intuitive what the playbook is doing.

Plays



```
---
- name: Sample IOS config for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:
```

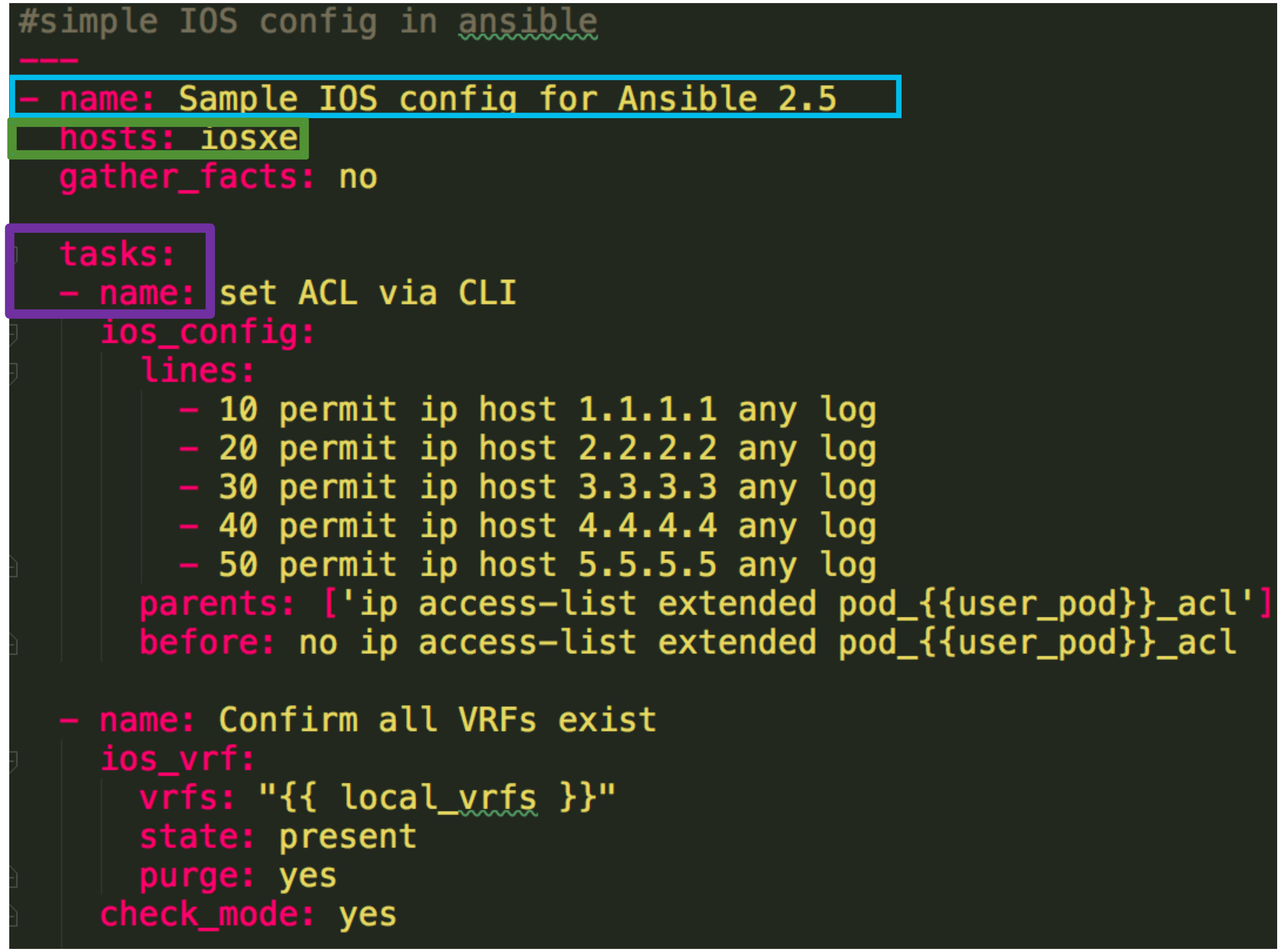


```
---
- name: Sample IOS config for Ansible 2.5
  hosts: routers-primary
  tasks:

- name: A Second play
  hosts: routers-secondary
  tasks:
```

- Each playbook is made up of one or more plays.
- Each play will reference a specific group from the inventory, and will apply one or more tasks to that group of devices.
- In the upper example, you can see a single '-' denotes there is one play in our list.
- In the lower example, you can see how the playbook contains a list of two unique plays. The first play applies to group routers-primary, and the second to routers-secondary

*outputs edited for brevity

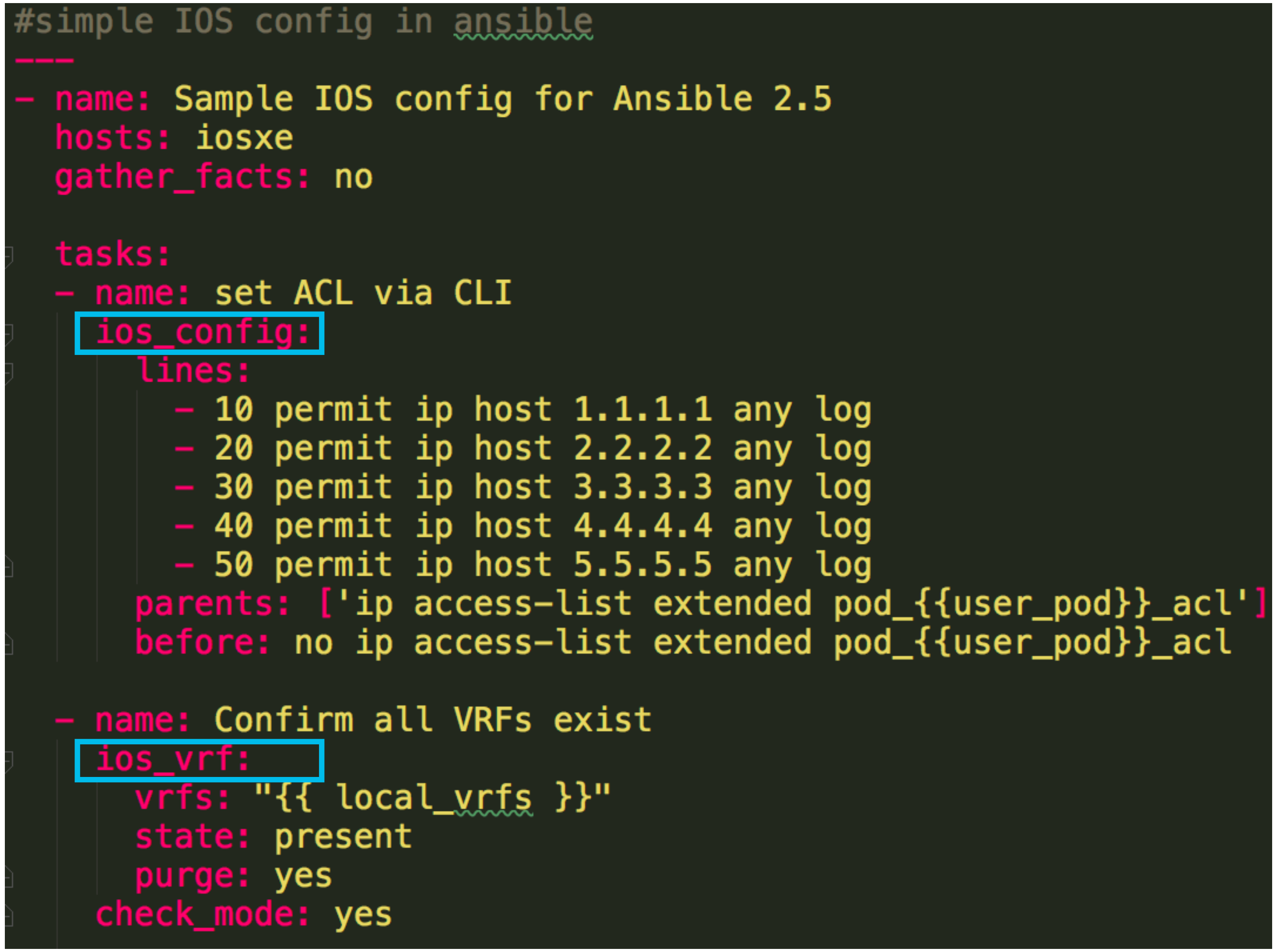
Names, Hosts, Tasks



```
#simple IOS config in ansible
---
- name: Sample IOS config for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:
  - name: set ACL via CLI
    ios_config:
      lines:
        - 10 permit ip host 1.1.1.1 any log
        - 20 permit ip host 2.2.2.2 any log
        - 30 permit ip host 3.3.3.3 any log
        - 40 permit ip host 4.4.4.4 any log
        - 50 permit ip host 5.5.5.5 any log
      parents: ['ip access-list extended pod_{{user_pod}}_acl']
      before: no ip access-list extended pod_{{user_pod}}_acl

  - name: Confirm all VRFs exist
    ios_vrf:
      vrfs: "{{ local_vrfs }}"
      state: present
      purge: yes
    check_mode: yes
```

- Name is an optional component, but it is a good practice to
- Hosts tells the play which host or group from our inventory to run against.
- Tasks defines a list of one or more actions to complete on each target host. Note in this example there are 2 tasks, denoted by the '-'

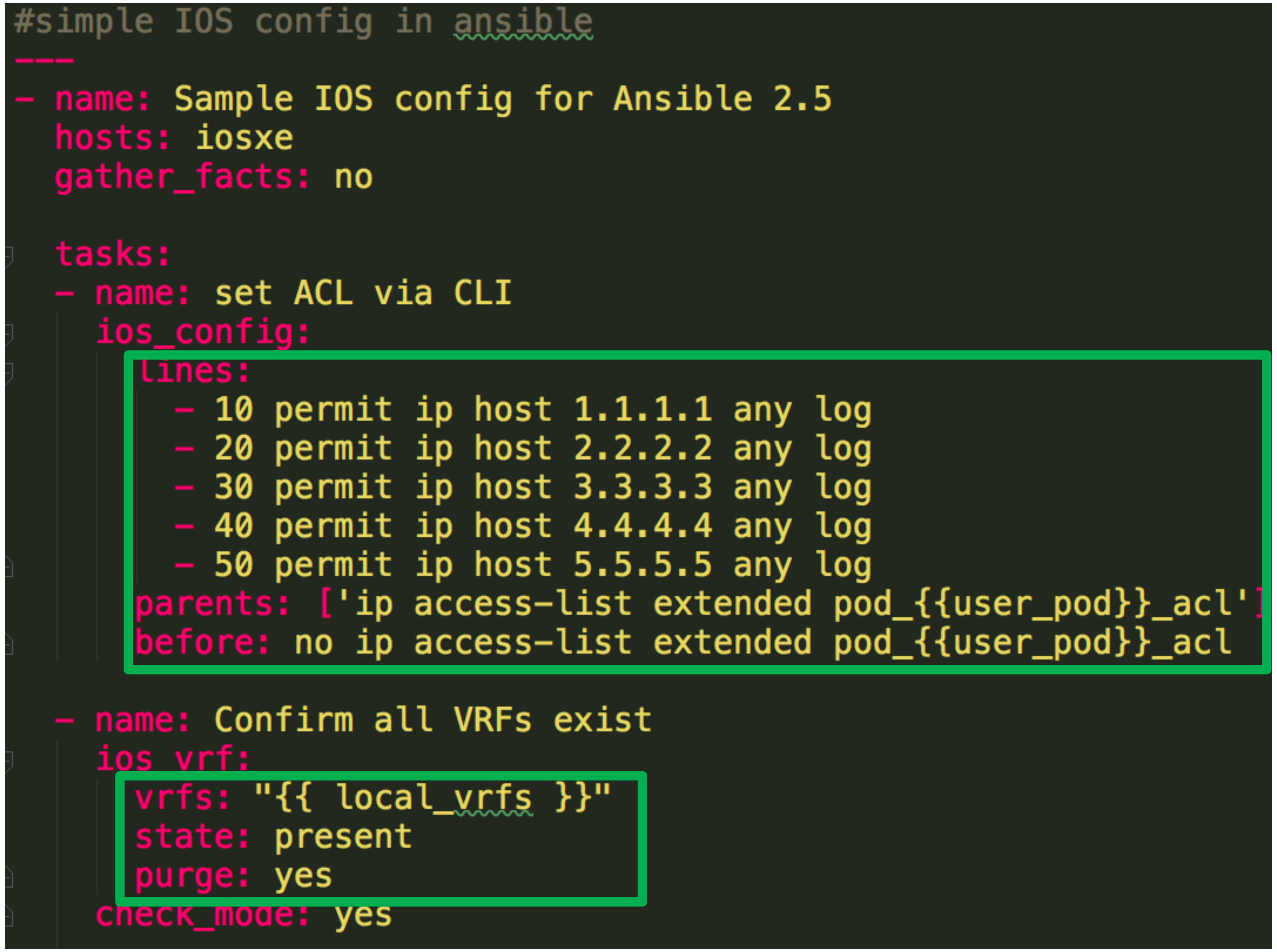
Modules



```
#simple IOS config in ansible
---
- name: Sample IOS config for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:
  - name: set ACL via CLI
    ios_config:
      lines:
        - 10 permit ip host 1.1.1.1 any log
        - 20 permit ip host 2.2.2.2 any log
        - 30 permit ip host 3.3.3.3 any log
        - 40 permit ip host 4.4.4.4 any log
        - 50 permit ip host 5.5.5.5 any log
      parents: ['ip access-list extended pod_{{user_pod}}_acl']
      before: no ip access-list extended pod_{{user_pod}}_acl

  - name: Confirm all VRFs exist
    ios_vrf:
      vrfs: "{{ local_vrfs }}"
      state: present
      purge: yes
    check_mode: yes
```

- Modules are the underlying code, usually python, to perform actions on hosts.
- Ansible has hundreds of included modules, but users can build their own as well.
- Within each task, a module must be referenced to define the desired action.
- In this example the first task uses module ios_config and the second uses ios_vrf

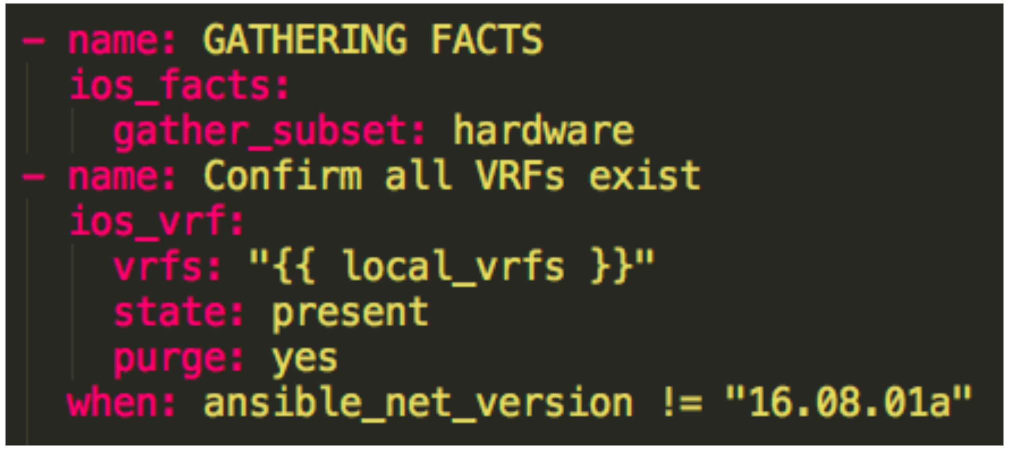
Parameters



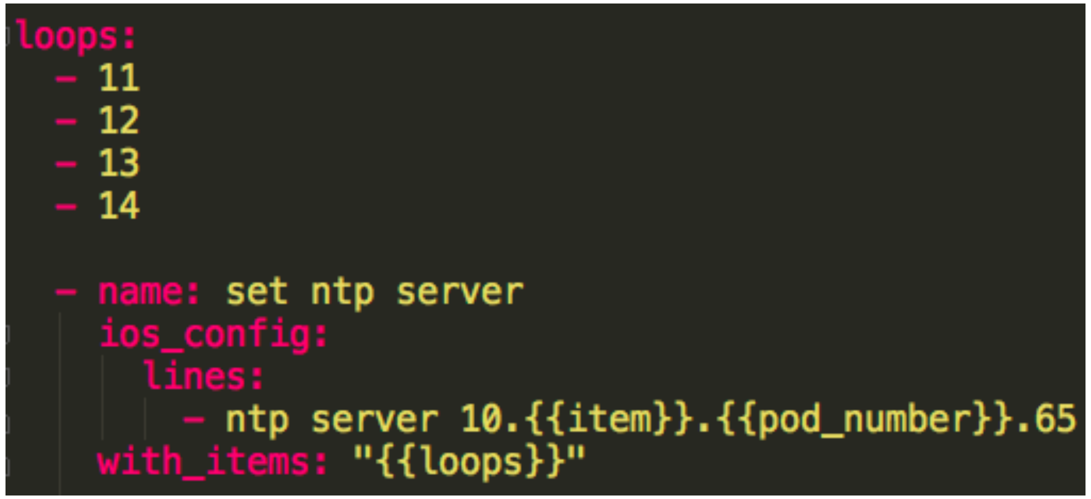
```
#simple IOS config in ansible
---
- name: Sample IOS config for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:
  - name: set ACL via CLI
    ios_config:
      lines:
        - 10 permit ip host 1.1.1.1 any log
        - 20 permit ip host 2.2.2.2 any log
        - 30 permit ip host 3.3.3.3 any log
        - 40 permit ip host 4.4.4.4 any log
        - 50 permit ip host 5.5.5.5 any log
      parents: ['ip access-list extended pod_{{user_pod}}_acl']
      before: no ip access-list extended pod_{{user_pod}}_acl

  - name: Confirm all VRFs exist
    ios_vrf:
      vrfs: "{{ local_vrfs }}"
      state: present
      purge: yes
    check_mode: yes
```

- In the first task, ios_config is passed a list of lines to be executed on the devices.
- ios_config will attempt to apply each line to each device
- ios_vrf shows 3 key parameters:
  - Vrfs is a list of VRFs to act on
  - State tells the module if the vrfs should exist (present) or not (absent)
  - Purge: tells the module to delete any vrfs not mentioned in the vrfs parameter

Conditionals and Loops



```
- name: GATHERING FACTS
  ios_facts:
    gather_subset: hardware
- name: Confirm all VRFs exist
  ios_vrf:
    vrfs: "{{ local_vrfs }}"
    state: present
    purge: yes
  when: ansible_net_version != "16.08.01a"
```



```
loops:
  - 11
  - 12
  - 13
  - 14

  - name: set ntp server
    ios_config:
      lines:
        - ntp server 10.{{item}}.{{pod_number}}.65
    with_items: "{{loops}}"
```

- The when parameter allows a task to run only when specific conditions are met.
- In this example, it will execute if the device is not the correct IOS version “16.08.01a”
- It can often be helpful to loop over a list within a task to reduce lines of redundant code.
- In the second example, we define a list of 4 numbers, and define 4 NTP servers, each using one of the numbers as the second octet
- When using a loop, the variable “{{item}}” captures the value of the list item for that loop.
-

*outputs edited for brevity

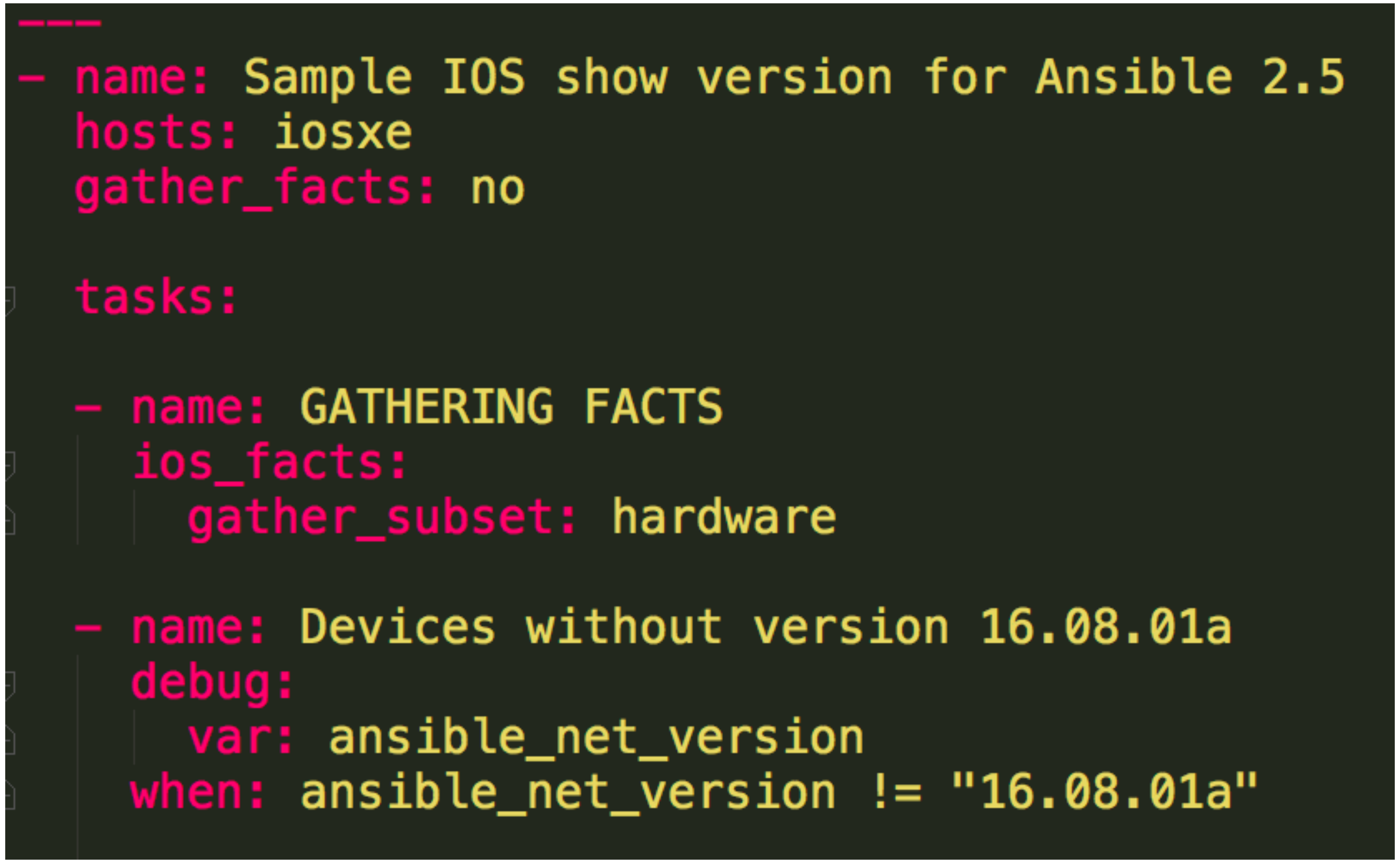
Understand the IOS Modules in Ansible

Available Cisco Modules

- Explore network modules for Cisco in Ansible Docs:
- Cisco ACI
- Cisco AireOS
- Cisco IOS
- Cisco IOS XR
- Cisco Meraki
- Cisco NSO
- Cisco NX-OS
- Cisco UCS

ios_facts

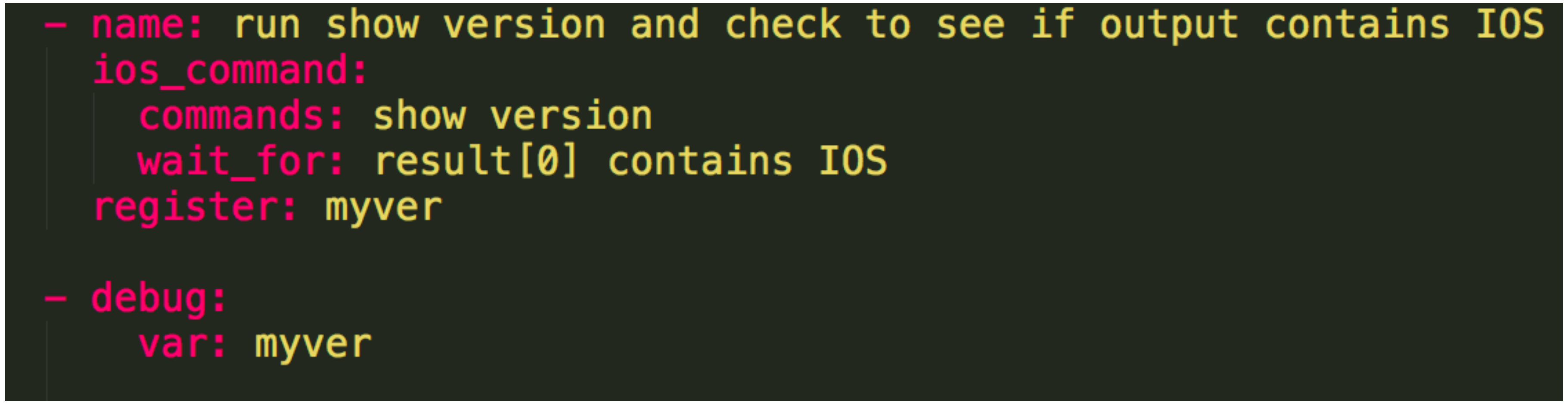
- Used for gathering information about the platform
- Sets standard variables for use in subsequent tasks, such as ansible_net_version
- Subsets of facts can be gathered as needed. For example, subset: hardware



```yaml
---
- name: Sample IOS show version for Ansible 2.5
  hosts: iosxe
  gather_facts: no

  tasks:

    - name: GATHERING FACTS
      ios_facts:
        gather_subset: hardware

    - name: Devices without version 16.08.01a
      debug:
        var: ansible_net_version
      when: ansible_net_version != "16.08.01a"
```

ios_command

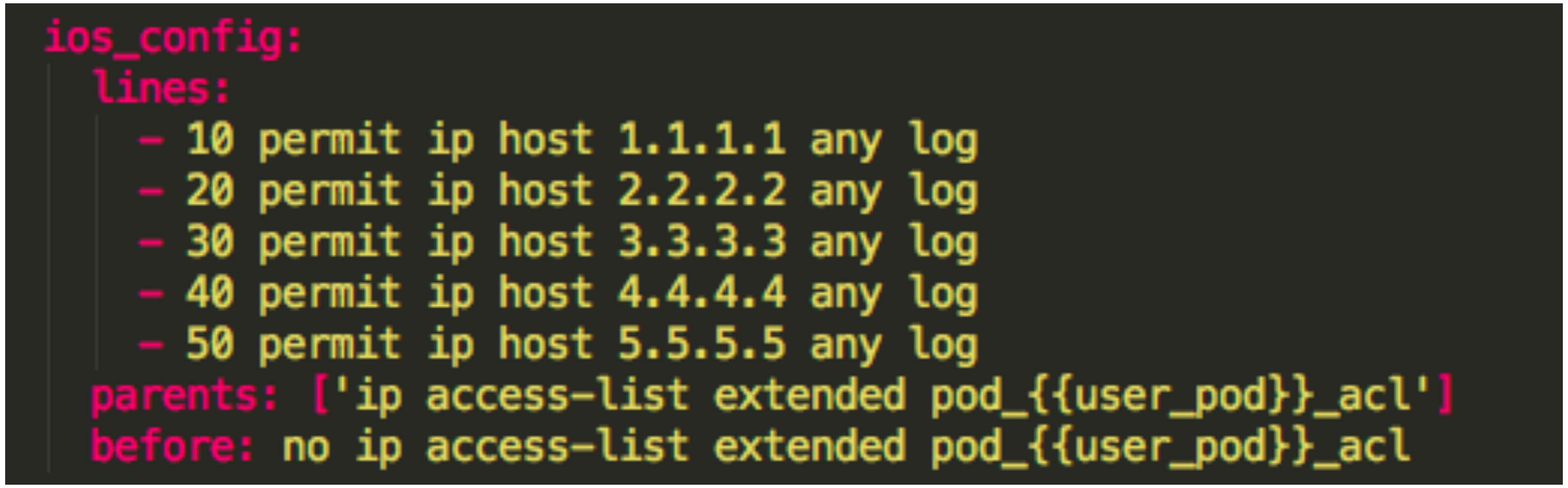
- Catch-all for running arbitrary commands on IOS
- Can run one command, or a list of multiple commands
- Returns raw text, so less convenient to work with than something like ios_facts



```yaml
- name: run show version and check to see if output contains IOS
  ios_command:
    commands: show version
    wait_for: result[0] contains IOS
  register: myver

- debug:
    var: myver
```

ios_config

- Catch-all for running arbitrary config commands on IOS
- Can run a single config command, or multiple config commands
- Can run [nearly] every config command, but many will not be idempotent
- Has optional parameters “parents” to handle nested commands such as lines within an ACL and “before” to do a task first such as deleting any existing ACL before creating

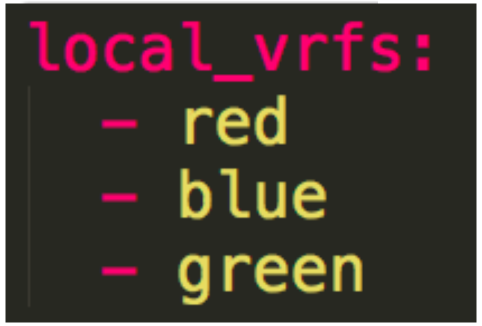


```
ios_config:
  lines:
    - 10 permit ip host 1.1.1.1 any log
    - 20 permit ip host 2.2.2.2 any log
    - 30 permit ip host 3.3.3.3 any log
    - 40 permit ip host 4.4.4.4 any log
    - 50 permit ip host 5.5.5.5 any log
  parents: ['ip access-list extended pod_{{user_pod}}_acl']
  before: no ip access-list extended pod_{{user_pod}}_acl
```

ios_vrf

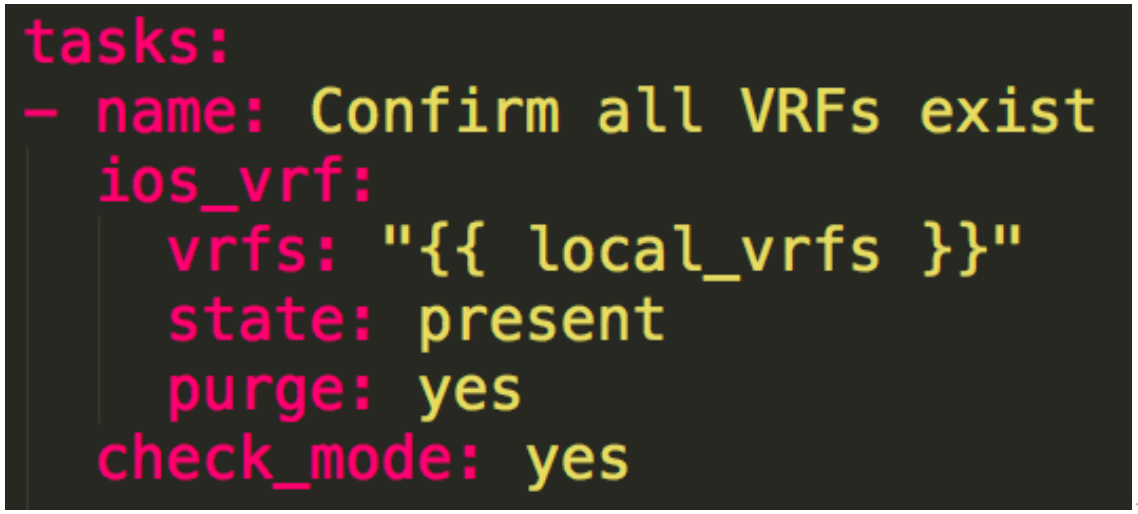
- Configure VRFs on an IOS device
- Supports passing in a hash for multiple VRFs
- Supports "purge", aka declarative mode
- This play shows optional "check_mode", which only reports on what it would do. It does not actually make changes when set.

vars.yml:



```
local_vrfs:
  - red
  - blue
  - green
```

vrfs.yml:

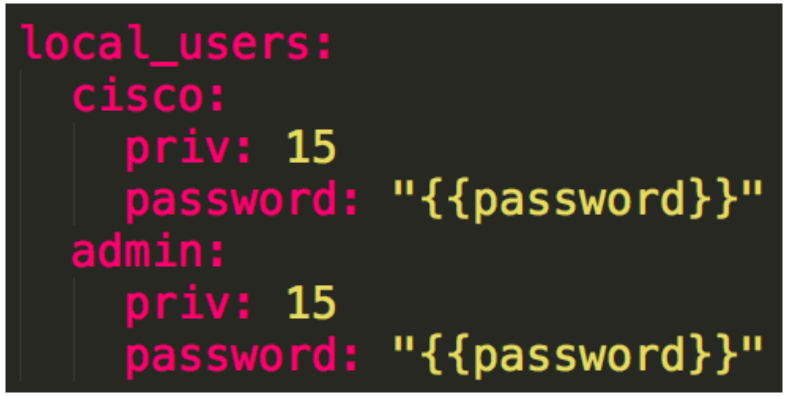


```
tasks:
- name: Confirm all VRFs exist
  ios_vrf:
    vrfs: "{{ local_vrfs }}"
    state: present
    purge: yes
  check_mode: yes
```

ios_user

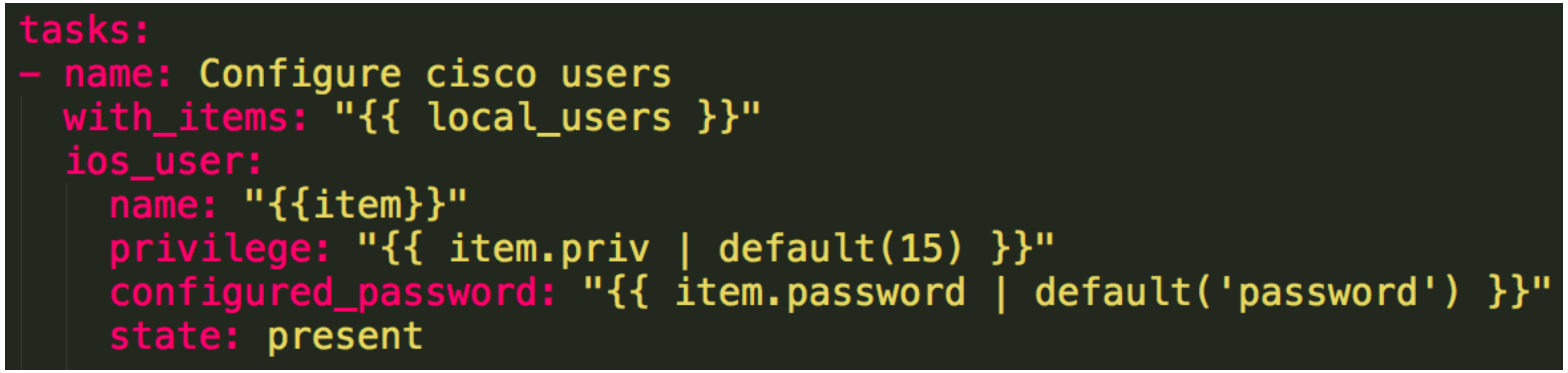
- Configure local users on IOS device
- This play shows an example of looping over a nested data structure to configure multiple users

vars.yml:



```
local_users:
  cisco:
    priv: 15
    password: "{{password}}"
  admin:
    priv: 15
    password: "{{password}}"
```

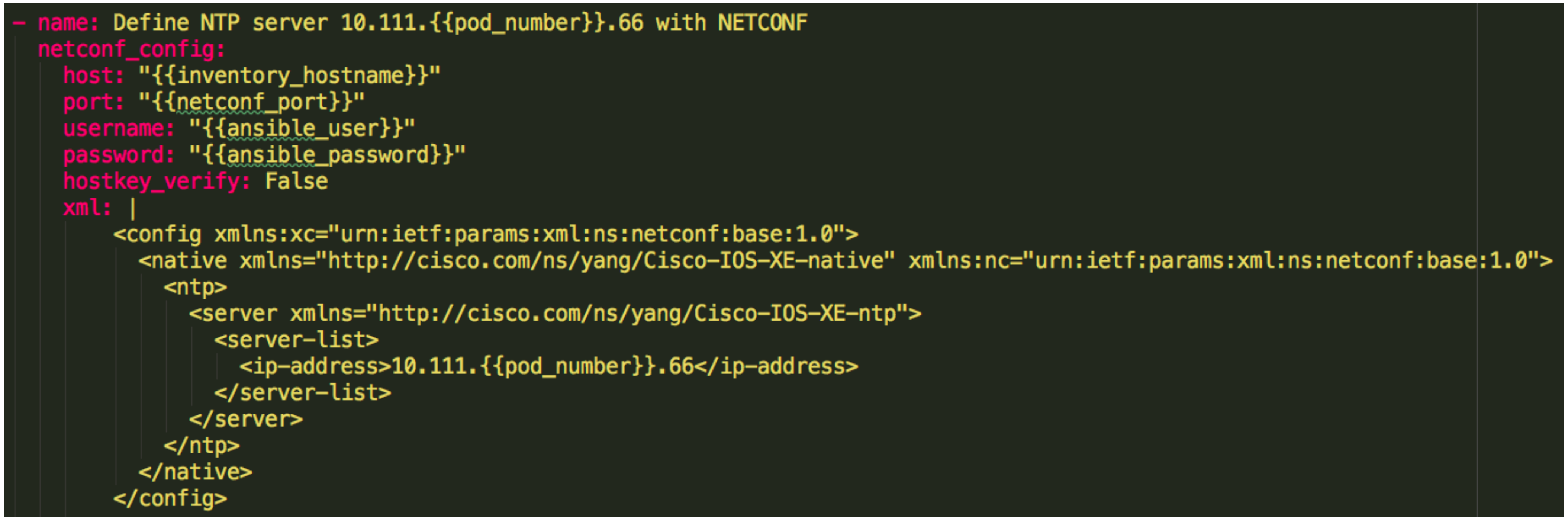
users.yml:



```
tasks:
- name: Configure cisco users
  with_items: "{{ local_users }}"
  ios_user:
    name: "{{item}}"
    privilege: "{{ item.priv | default(15) }}"
    configured_password: "{{ item.password | default('password') }}"
    state: present
```

netconf_config

- Allows configuration of multiple different platforms leveraging a single supported YANG data model



```
- name: Define NTP server 10.111.{{pod_number}}.66 with NETCONF
  netconf_config:
    host: "{{inventory_hostname}}"
    port: "{{netconf_port}}"
    username: "{{ansible_user}}"
    password: "{{ansible_password}}"
    hostkey_verify: False
    xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <ntp>
              <server xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ntp">
                <server-list>
                  <ip-address>10.111.{{pod_number}}.66</ip-address>
                </server-list>
              </server>
            </ntp>
          </native>
        </config>
```

Wrap Up

What you learned in this module…

- What are the basic building blocks of Ansible
- What are the common IOS modules and how can they be used
- How to leverage various Ansible capabilities including loops.
- How can NETCONF be used to run playbooks across multiple platforms

Connect with Us and share Your Story…

CISCO DEVNET express

LISTEN > LEARN >> PUT IT INTO PRACTICE

@CiscoDevNet

#DevNetExpress

facebook.com/ciscodevnet/

http://github.com/CiscoDevNet

developer.cisco.com

CISCO

DEVNET

express

LISTEN > LEARN >> PUT IT INTO PRACTICE

@CiscoDevNet | #DevNetExpress