pue
IMPULSANDO EL CONOCIMIENTO
TIC CUALIFICADO

EDUCACIÓN
CALIDAD Y CERTIFICACIÓN
EN SUS PROGRAMAS EDUCATIVOS

Programación de Redes – Becas Digitaliza - 2019

PUE – ITC – Formación de Instructores

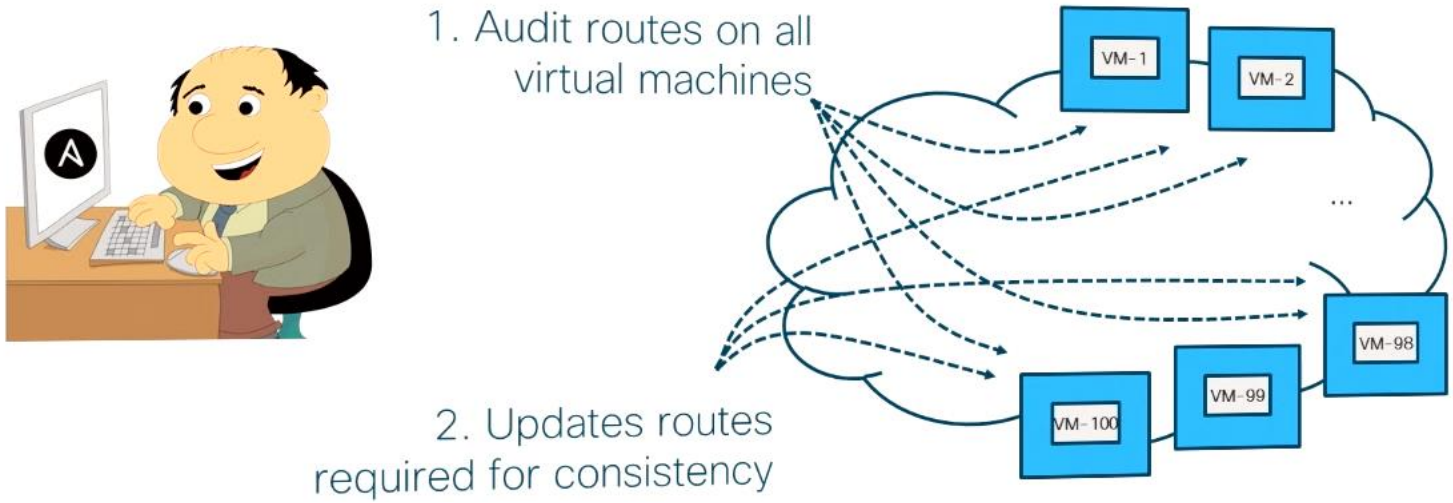
Sesión 12 – Ansible – Closing

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA

Index

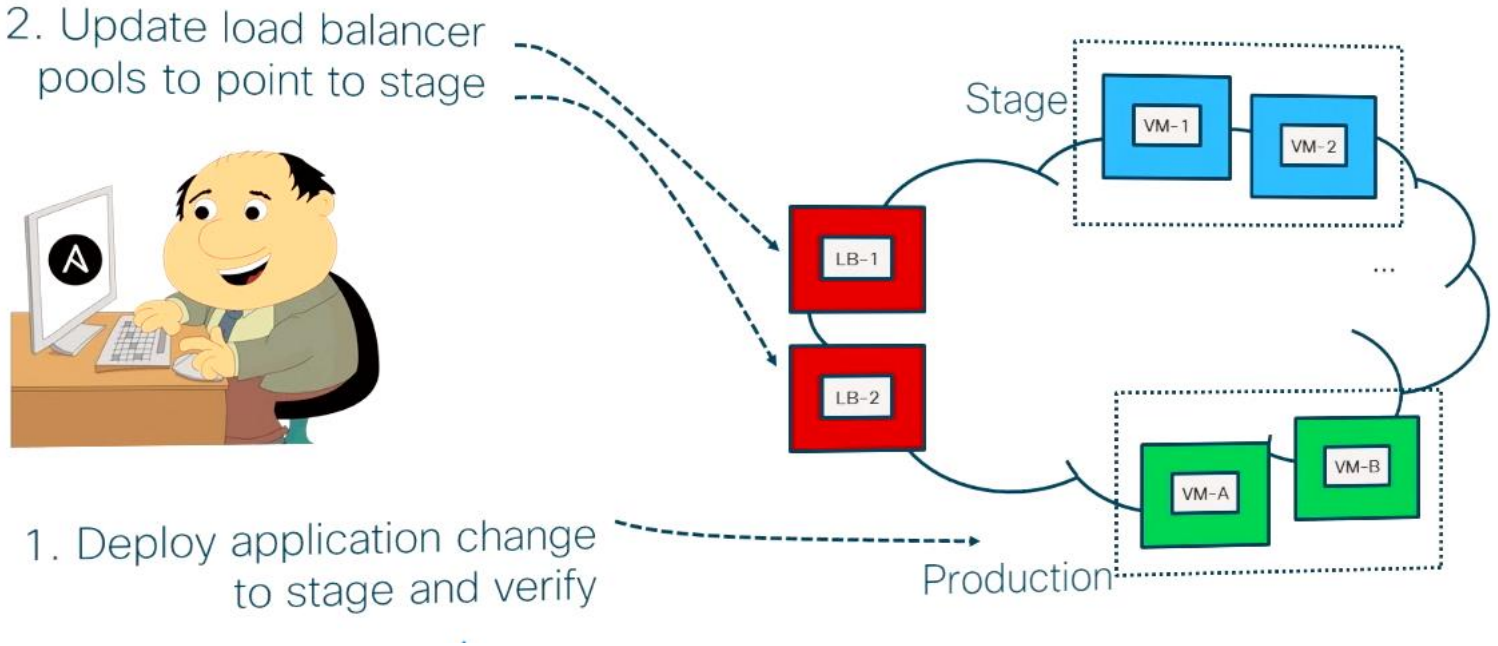
Why Learn Ansible?

Why Learn Ansible?



1. ansible –m shell –a “netstat –rn” datacenter-east
2. ansible –m shell –a “route add X.X.X.X” datacenter-east

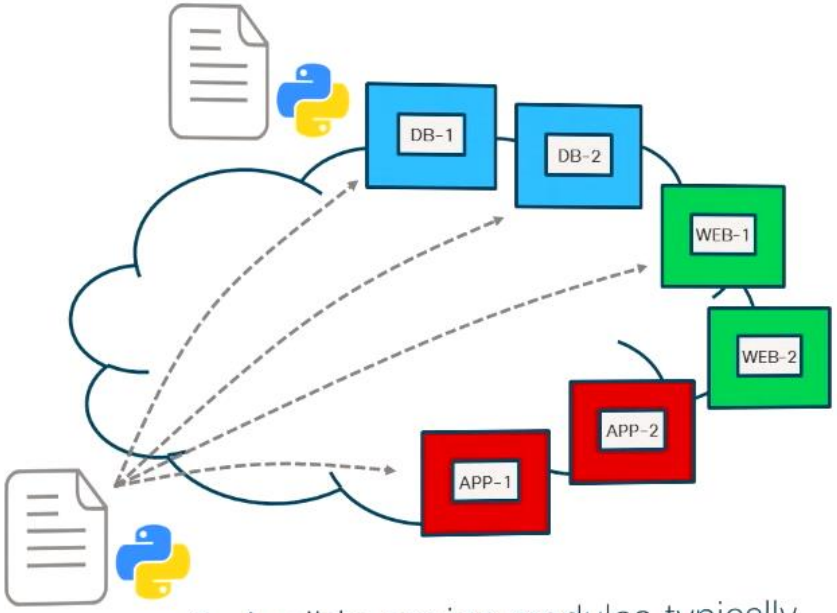
Change Control Workflow Orchestration



How does Ansible work?

1. Engineers deploy Ansible playbooks written in YAML to a control station

Ansible Control Station



2. Ansible copies modules typically written in Python to remote hosts to execute tasks

Ansible Environment Setup

Inside the Ansible Control Station

- Linux host with a Python and the Ansible installed
- Transport is typically SSH, but could use an API
- Ansible Components
  - Ansible configuration file (how the own instance of ansible works)
  - Inventory files (groups of hosts)
  - Ansible modules
  - Playbooks (what do we want to do)
- Python 2.7 (Python3.x is in “tech preview”)
  - apt install software-properties-common
  - apt-add-repository ppa:ansible/ansible
  - apt update
  - apt install ansible
  - apt install python-pip
  - pip install ansible

Ansible Configuration File

- Control operation of Ansible
- Default configuration */etc/ansible/ansible.cfg*
- Override default settings
  - ANSIBLE_CONFIG ENV
  - ansible.cfg in current directory
  - .ansible.cfg in home directory

```
DevNet$ cat ansible.cfg

# config file for ansible
# override global certain global settings

[defaults]
# default to inventory file of ./hosts
inventory      = ./hosts

# disable host checking to automatically add
# hosts to known_hosts
host_key_checking = False

# set the roles path to the local directory
roles_path     = ./
```

Ansible Authentication Basics

- Tipically, Ansible uses SSH for authentication and assumes keys are in place
- Setting up and transferring SSH keys allows playbooks to be run automatically
- Using passwords is ossible
  - Network Devices often use passwords

```
DevNet$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key:
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_rsa.
Your public key has been saved in ~/.ssh/id_rsa.pub.
```

```
DevNet$ ssh-copy-id root@10.10.20.20
.
Number of key(s) added:        1

Now try logging into the machine, with:
"ssh 'root@10.10.20.20'"
```

```
DevNet$ ssh root@10.10.20.20
Last login: Fri Jul 28 13:33:46 2017 from 10.10.20.7
(python2) [root@localhost sbx_nxos]#
```

Ansible Inventory File

- Inventory file identifies hosts, and groups of hosts under management
  - Hosts can be IP or FQDN
  - Groups enclosed in []
- Can include host specific parameters as well
  - Example: Instructing Ansible to use the active Python Interpreter when using Python Virtual Environments

```
[servers]
10.10.20.20 ansible_python_interpreter="/usr/bin/env python"

[switches]
172.16.30.101 ansible_python_interpreter="/usr/bin/env python"
172.16.30.102 ansible_python_interpreter="/usr/bin/env python"
172.16.30.103 ansible_python_interpreter="/usr/bin/env python"
172.16.30.104 ansible_python_interpreter="/usr/bin/env python"
```

Ansible CLI Tool Overview

- Ansible --> Executes modules against targeted hosts without creating playbooks
- ansible-playbook --> Run playbooks against targeted hosts
- ansible-vault --> Encrypt sensitive data into an encrypted YAML file
- ansible-pull --> Reverses the normal “push” model and lets clients “pull” from a centralized server for execution
- ansible-docs --> Parses the docstrings of Ansible modules to see example syntax and the parameters modules require
- ansible-galaxy --> Creates or downloads roles from the Ansible community

Using Ansible CLI for ad-hoc Commands

- Quickly run a command against a set of hosts
- Specify the module with –m module
- Specify the username to use with –u user
  - Default is to use local username
- Specify the server or group to target
- Provide module arguments with –a argument

```
DevNet$ ansible -m setup -u root servers

10.10.20.20 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.10.20.20",
            "172.17.0.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::250:56ff:febb:3a3f"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
.
.
```

Modules, Tasks, Plays, Playbooks, & Roles

Ansible Terms

- **module** --> Code, typically written in Python, that will perform some action on a host (example: yum -> manages packages with the yum package manager)
- **task** --> A single action that references a module to run along with any input arguments and actions
- **play** --> Matching a set of tasks to a host or group of hosts
- **playbook** --> A YAML file that includes one or more play
- **role** --> A pre-built set of playbooks designed to perform some standard configuration in a repeatable fashion. A play could leverage a role rather han tasks (example: a role to configure a web server would install Apache, configure the firewall and copy application files).

Ansible Playbooks

- Written in YAML
- One or more plays that contain hosts and tasks
- Tasks have a name & module keys
- Modules have parameters
- Variables referenced with {{name}}
  - Ansible gathers “facts”
  - Create your own by registering output from another task

```
---
- name: Report Hostname and Operating System Details
  hosts: servers

  tasks:
    - name: "Get hostname from server"
      debug:
        msg: "{{ansible_hostname}}"

    - name: "Operating System"
      debug: msg="{{ansible_distribution}}"

- name: Report Network Details of Servers
  hosts: servers

  tasks:
    - name: "Default IPv4 Interface"
      debug: msg="{{ansible_default_ipv4.interface}}"

    - name: "Retrieve network routes"
      command: "netstat -rn"
      register: routes

    - name: "Network routes installed"
      debug: msg="{{routes}}"
```

Ansible: Example

```
DevNet$ ansible-playbook -u root example1.yaml

PLAY [Report Hostname and Operating System Details]
****************************************************************************************************
TASK [Gathering Facts]
****************************************************************************************************
ok: [10.10.20.20]

TASK [Get hostname from server]
****************************************************************************************************
ok: [10.10.20.20] => {
    "msg": "localhost"
}

PLAY [Report Network Details of Servers]
****************************************************************************************************
TASK [Network routes installed]
****************************************************************************************************ok:
[10.10.20.20] => {
        "stdout_lines": [
            "Kernel IP routing table",
            "Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface",
            "0.0.0.0         10.10.20.254    0.0.0.0         UG        0 0          0 ens160",
            "10.10.20.0      0.0.0.0         255.255.255.0   U         0 0          0 ens160",
            "172.16.30.0     10.10.20.160    255.255.255.0   UG        0 0          0 ens160",
        ]

PLAY RECAP
****************************************************************************************************
********************
10.10.20.20                : ok=7    changed=1    unreachable=0    failed=0
```

Variables, Loops, and Templates

Using Variable Files and Loops with Ansible

- Include external variable files using vars_files: filename.yaml
- Reference variables with {{name}}
- YAML supports lists and hashes (i.e.: key/value)
- Loop to repeat actions with with_items: variable

Hands-on

Example2.yaml, &
example2_vars.yaml

Jinja2 Templating – Variables to the Max!

- Not just for Ansible templates
- Powerful templating language
  - Loops, conditionals and more supported
- Leverage template module
  - Attributes
    - src: The template file
    - dest: where to sabe generated template

Hands-on

example3.yaml, example3.j2, & example3.conf

```
# example3.yaml
# Illustrate the following concepts:
# - Using Jinja templates
#########################################################
---
- name: Generate Configuration from Template
  hosts: localhost
  gather_facts: false
  vars:
    feature: bgp
    asn: 65001
    router_id: 10.10.10.1

  tasks:
    - name: "Generate config"
      template:
        src: "example3.j2"
        dest: "./example3.conf"
```

example3.yaml

```
feature {{feature}}
router bgp {{asn}}
  router-id {{router_id}}
```

example3.j2

Host and Group Variables

- Ansible allows for Group and Host specific variables
  - Group_vars/groupname.yaml
  - Host_vars/host.yaml
- Variables automatically available

```
ansible@ubuntu:~/Documents/netprog_basics/netdevops/ansible_part_1$ ls -la
total 60
drwxrwxr-x 3 ansible ansible 4096 Apr  1 08:15 .
drwxrwxr-x 7 ansible ansible 4096 Apr  1 07:53 ..
-rw-rw-r-- 1 ansible ansible  300 Apr  1 07:53 ansible.cfg
-rw-rw-r-- 1 ansible ansible   12 Apr  1 07:53 example1.retry
-rw-rw-r-- 1 ansible ansible 1012 Apr  1 07:53 example1.yaml
-rw-rw-r-- 1 ansible ansible   12 Apr  1 08:05 example2.retry
-rw-rw-r-- 1 ansible ansible  530 Apr  1 07:53 example2_vars.yaml
-rw-rw-r-- 1 ansible ansible  698 Apr  1 08:10 example2.yaml
-rw-rw-r-- 1 ansible ansible   52 Apr  1 08:15 example3.conf
-rw-rw-r-- 1 ansible ansible   65 Apr  1 07:53 example3.j2
-rw-rw-r-- 1 ansible ansible  656 Apr  1 07:53 example3.yaml
-rw-rw-r-- 1 ansible ansible  335 Apr  1 07:53 hosts
drwxrwxr-x 2 ansible ansible 4096 Apr  1 07:53 host_vars
-rw-rw-r-- 1 ansible ansible 1881 Apr  1 07:53 README.md
-rw-rw-r-- 1 ansible ansible   66 Apr  1 07:53 requirements.txt
```

Core Networking Modules

Core Networking Modules

https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

- Ansible includes many network modules by default
  - Includes Cisco as well as many other vendors
- Other modules maybe available for manual installation

- Examples: Aci, Citrix, Cli, Ios, Iosxr, Meraki, Netconf, Nxos…

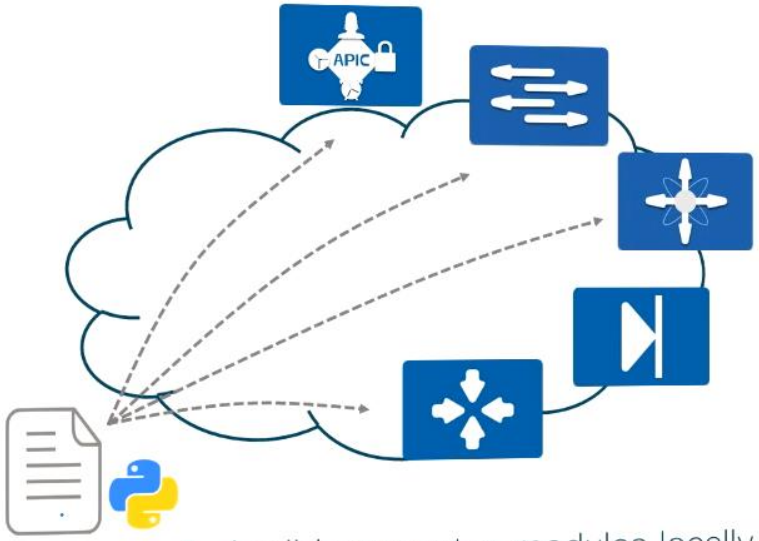
Ansible Environment Setup for Networking

Ansible and Networking

1. Engineers deploy Ansible playbooks written in YAML to a control station

```
---
- name: Retrieve facts from Switches
  hosts: switches
  connection: local
```

Ansible Control Station



2. Ansible executes modules locally using APIs to interface with devices

Network Authentication

- Most connections through SSH, but many network modules and require username and password for authentication
- Use Ansible environment variables to set:
  - export ANSIBLE_NET_USERNAME=cisco
  - export ANSIBLE_NET_PASSWORD=cisco
- Suggest creating and sourcing a file to simplify

```
# Setup environment for ansible playbooks
# Credentials for DevNet Open NX-OS Sandbox
#    https://devnetsandbox.cisco.com/

# usage: source .ansible_env

# Authentication for ansible network modules
export ANSIBLE_NET_USERNAME=cisco
export ANSIBLE_NET_PASSWORD=cisco
ansible@ubuntu:~/Documents/netprog_basics/netdevops/ansible_part_2$ source .ansible_env
ansible@ubuntu:~/Documents/netprog_basics/netdevops/ansible_part_2$ echo $ANSIBLE_NET_USERNAME
cisco
```

Configuring Cisco Network Devices with Ansible

Ansible NX-OS Integration

- NX-OS modules included in Ansible core
  - Robust module list across features
- Transport API Options
  - cli – default
  - Nxapi
- To Use NX-API must enable feature
  - *conf t*
  - *feature nxapi*
  - *exit*

Nxos

- nxos_aaa_server – Manages AAA server global configuration.
- nxos_aaa_server_host – Manages AAA server host-specific configuration.
- nxos_acl – Manages access list entries for ACLs.
- nxos_acl_interface – Manages applying ACLs to interfaces.
- nxos_banner – Manage multiline banners on Cisco NXOS devices
- nxos_bgp – Manages BGP configuration.
- nxos_bgp_af – Manages BGP Address-family configuration.
- nxos_bgp_neighbor – Manages BGP neighbors configurations.
- nxos_bgp_neighbor_af – Manages BGP address-family's neighbors configuration.
- nxos_command – Run arbitrary command on Cisco NXOS devices
- nxos_config – Manage Cisco NXOS configuration sections
- nxos_evpn_global – Handles the EVPN control plane for VXLAN.
- nxos_evpn_vni – Manages Cisco EVPN VXLAN Network Identifier (VNI).
- nxos_facts – Gets facts about NX-OS switches
- nxos_feature – Manage features in NX-OS switches.
- nxos_file_copy – Copy a file to a remote NXOS device.
- nxos_gir – Trigger a graceful removal or insertion (GIR) of the switch.
- nxos_gir_profile_management – Create a maintenance-mode or normal-mode profile for GIR.
- nxos_hsrp – Manages HSRP configuration on NX-OS switches.
- nxos_igmp – Manages IGMP global configuration.
- nxos_igmp_interface – Manages IGMP interface configuration.
- nxos_igmp_snooping – Manages IGMP snooping global configuration.
- nxos_install_os – Set boot options like boot, kickstart image and issu.
- nxos_interface – Manages physical attributes of interfaces.
- nxos_interface_ospf – Manages configuration of an OSPF interface instance.
- nxos_ip_interface – Manages L3 attributes for IPv4 and IPv6 interfaces. (D)
- nxos_l2_interface – Manage Layer-2 interface on Cisco NXOS devices.
- nxos_l3_interface – Manage L3 interfaces on Cisco NXOS network devices
- nxos_linkagg – Manage link aggregation groups on Cisco NXOS devices.
- nxos_lldp – Manage LLDP configuration on Cisco NXOS network devices.
- nxos_logging – Manage logging on network devices

Ansible Playbook for NX-OS

- connection: local
  - Run playbook on control station
- Modules all in format nxos_XXX
- host: “{{inventory_hostname}}”
  - Run against current host
- transport: nxapi or default to SSH (if we omit the transport)

```
---
- name: Retrieve facts from Switches
  hosts: switches
  connection: local

  tasks:
    - name: "Retrieving NX-OS Facts"
      nxos_facts:
        host: "{{inventory_hostname}}"
        transport: nxapi
      register: nxos_info

    - name: "Print NX-OS Facts"
      debug: msg="{{nxos_info}}"
```

Ansible Playbook: Example

ansible_part_2/example1.yaml

```
DevNet$ ansible-playbook example1.yaml
PLAY [Retrieve facts from Switches] ********************************************

TASK [Retrieving NX-OS Facts] **************************************************

TASK [Print NX-OS Facts] *******************************************************
ok: [172.16.30.104] => {
    "msg": {
        "ansible_facts": {
            "ansible_net_all_ipv4_addresses": [
                "172.16.30.104",
                "192.168.0.4"
            ],
            "ansible_net_hostname": "nx-osv9000-4",
            "ansible_net_interfaces": {
                "Ethernet1/1": {
                    "bandwidth": 1000000,
                    "description": "Ethernet1/1",
                    "duplex": "full",
                    "macaddress": "fa16.3e00.4001",

PLAY RECAP *********************************************************************
172.16.30.101              : ok=3    changed=0    unreachable=0    failed=0
172.16.30.102              : ok=3    changed=0    unreachable=0    failed=0
172.16.30.103              : ok=3    changed=0    unreachable=0    failed=0
172.16.30.104              : ok=3    changed=0    unreachable=0    failed=0
```

Configuring NX-OS Interfaces with Ansible

- Reference host variable for a list of loopbacks
- Within each loop {{item.X}} allows access to details
- Order of operations
  - Create new Loopback
  - Configure IP address

host_vars/172.16.30.101

```
local_loopbacks:
    - name: Loopback11
      desc: Sample Network Route Injection
      ip_address: 172.21.1.1
      prefix: 24
    - name: Loopback12
      desc: Sample Network Route Injection
```

```
---
- name: Configure Loopback Networks
  hosts: switches
  connection: local

  tasks:
  - name: Create Loopback Interface
    with_items: "{{ local_loopbacks }}"
    nxos_interface:
      host: "{{ inventory_hostname }}"
      interface: "{{ item.name }}"
      mode: layer3
      description: "{{ item.desc }}"
      admin_state: up

  - name: Configure IPv4 Address on Interface
    with_items: "{{ local_loopbacks }}"
    nxos_ip_interface:
      host: "{{ inventory_hostname }}"
      state: present
      interface: "{{ item.name }}"
      version: v4
      addr: "{{ item.ip_address }}"
      mask: "{{ item.prefix }}"
```

Configuring NX-OS Interfaces with Ansible: Example

```
DevNet$ ansible-playbook example2.yaml

PLAY [Configure Loopback Networks on Each Switch] ***********************

TASK [Gathering Facts] ***************************************************
ok: [172.16.30.101]
ok: [172.16.30.104]
ok: [172.16.30.102]
ok: [172.16.30.103]

TASK [Create Loopback Interface] *****************************************
ok: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.1.1',
ok: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.2.1',
ok: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.3.1',
ok: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.4.1',

TASK [Configure IPv4 Address on Interface] *******************************
changed: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.1.1',
changed: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.2.1',
changed: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.3.1',
changed: [172.16.30.101] => (item={u'prefix': 24, u'ip_address': u'172.21.4.1',

PLAY RECAP ***************************************************************
172.16.30.101              : ok=3    changed=1    unreachable=0    failed=0
172.16.30.102              : ok=3    changed=1    unreachable=0    failed=0
172.16.30.103              : ok=3    changed=1    unreachable=0    failed=0
172.16.30.104              : ok=3    changed=1    unreachable=0    failed=0
```

“Infrastructure as Code”… What does that mean?

IaC – Principals of "Network as Code"

- IaC is the process of managing and provisioning computer data centers through machine-readable definition files…
- Rather than directly targeting hardware, interfaces and so on to get information, we will use snippets, codes and scripts to do it and configure it.

- Store network configuration in source control systems (i.e.: git)
  - Use "machine readable" formats like YAML, JSON, and XML
- Treat the source control as single source of truth
  - Develop, test, and deploy to prod from same source
- Deploy configuration using programmatic APIs and tooling
  - Limit manual network configuration

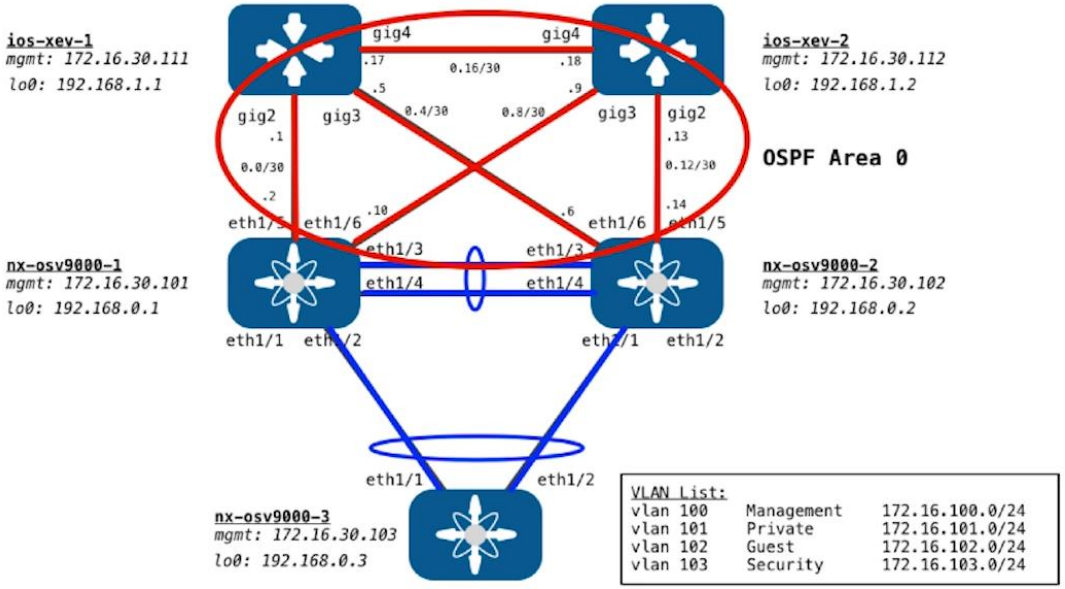
Our Network "Intent"

Starting Network Topology

- Physical Topology
  - “Core” –> IOS XE Routers
  - “Distribution” –> NX-OS Switches
  - “Access” –> NX-OS Switches
- Network has been cabled already
- Management access to devices enabled
  - No other configuration completed

Desired Network Configuration

- L3 links between Core/Dist
  - OSPF Area 0 Routing Configured
- Distribution configured for VPC Domain
- L2 port-cannel trunk to Access
- Set of VLANs Configured
  - SVIs at Distribution with HSRP Configured



Our Network as Code with Ansible

Ansible Mastery!

- Inventory File
  - List network devices
  - Logically group for configuration
- Host/Group Variable Files
  - Device specific details
  - General group details
- Ansible Roles
  - Align to network roles
- Ansible Playbook
  - Run roles against relevant groups

Network Inventory

- Groups for core / distribution / Access tiers
- Group to identify network operating systems
- Set global configuration

```
[all:vars]
ansible_python_interpreter="/usr/bin/env python"

[iosxe:children]
core

[nxos:children]
distribution
access

[core]
172.16.30.111
172.16.30.112

[distribution]
172.16.30.101
172.16.30.102

[access]
172.16.30.103
```

Host Variables Files

- Device specific configuration details
  - OSPF Router Id
  - VPC keepalive details
  - Layer 3 Interfaces
- Manage network configuration by updating details
  - Example: Configure addition layer 3 interfaces by adding to list in file

Group Variable Files

- Configuration details consistent across group
  - HSRP configuration
  - OSPF and VPC general details
  - Trunk port details
- Manage network configuration by updating details
  - Example: Add additional trunk link by adding to list

“All” Group Variable File

- Configuration details consistent across network
  - VLAN list
- Manage network configuration by updating details
  - Example: Add new VLANs just by adding to list

Ansible Roles Per Feature

- Reusable roles target specific network configuration
- Start with enabling APIs for use
- Different groups will get different roles

```
$ ls roles/

iosxe_mdp               < Enable NETCONF
nxos_nxapi              < Enable NX-API

netconf_l3_interfaces < Configure Interfaces
netconf_ospf            < Configure Routing

nxos_vlans              < Add VLAN
nxos_vpc                < Setup VPC
nxos_vpc_trunks         < Create VPC Trunk
nxos_po_trunks          < Create Po Trunk
nxos_l3_interfaces      < Configure Interfaces
nxos_hsrp               < Setup HSRP
nxos_ospf               < Configure Routing
```

Make it happen!

Gracias por vuestra atención

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA
Área de Proyectos de Educación