Network Automation using Ansible

LABPRG-1481

Powered by Cisco dCloud!

https://dcloud.cisco.com

Speakers:

Ambili Sasidharan – Leader, Customer Delivery

Anis Edavalath – Customer Delivery Architect

Table of Contents

Prerequisites

- Hands on experience with running commands to a Linux shell
- Basic Cisco router CLI configuration experience, knowledge in any programming language is a plus
- It is strongly recommended to go through the lab guide completely.

Disclaimer

This training document is to familiarize with Ansible and introduce the participant about automating Cisco NxOs and IOS devices automation using ansible. Although the lab design and configuration and ansible playbook examples could be used as a reference, it's not a real design, thus not all recommended features are used, or enabled optimally to use in production networks. For the design related questions please contact your representative at Cisco, or a Cisco partner. Else feel free to reach out to us.

Pre-Configuration

Due to limited duration of WISP lab, we have some pre-configuration, already configured to save your time.

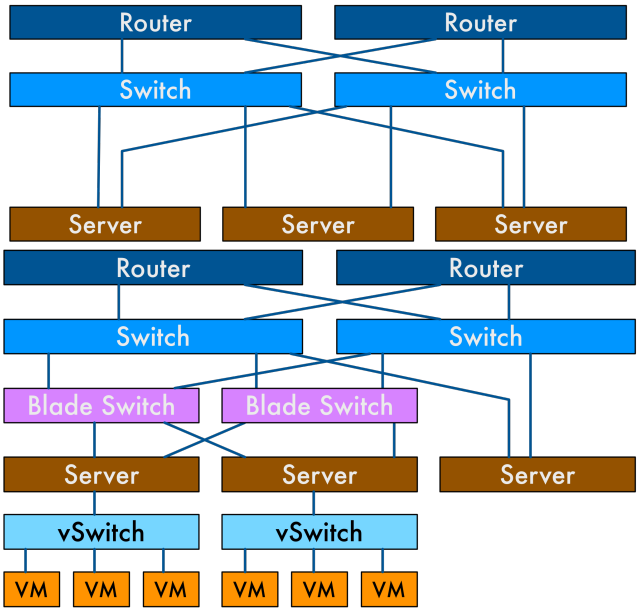
Lab completion

Upon completion of this lab, you will be able to:

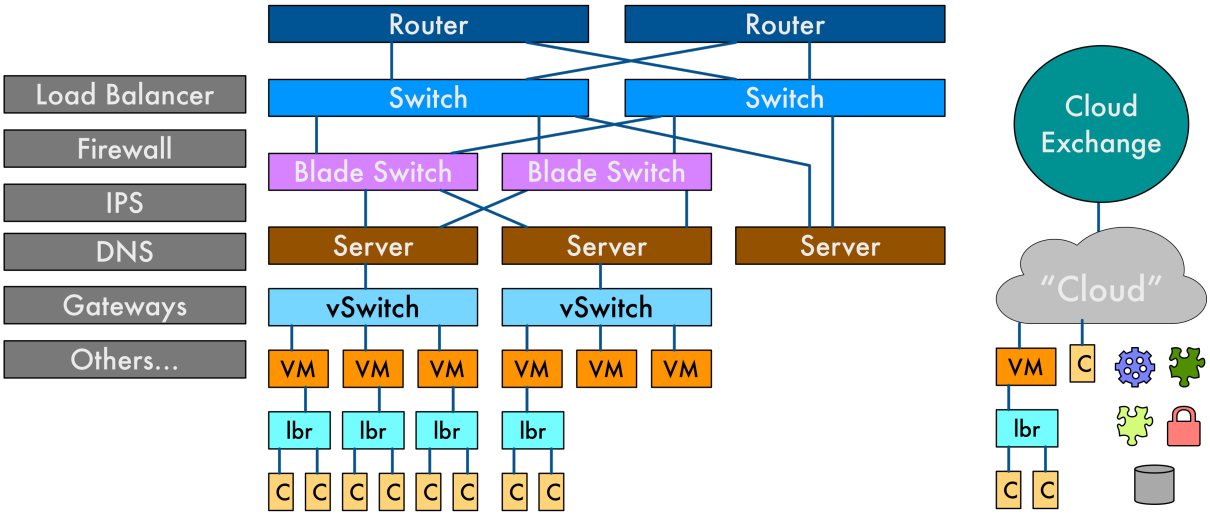
- Have a strong foundational knowledge of ansible
- Able to create basic playbook for cisco network automation tasks
- Learn how to find community resources in ansible
- Install and replicate Ansible environment in your Datacenter Production and QA environment

Introduction

In the SDN Era, the roles and responsibilities of the Network Engineers are drastically changing. Reducing Opex and Faster Time to market are the key business drivers for most of the IT organizations today. As the networks grow, it has increased the management and operations complexity to a very high level. State of the Network has changed a lot over the recent years from a simple three tier architecture to multi-level modern network architecture.



Simple Network



Modern Network

devices via API's. In today's world network administrators want programming capabilities in their networking devices for several reasons. The main reason is to automate provisioning, push configuration changes and troubleshoot complex network issues. Yet business and application teams demands the network devices to be provisioned faster, with less resources and no human-error.

Ansible is the latest Buzzword in the network programmability, and it is heavily used today in netops and devops for network automation, Orchestration, and management.

Ansible

Introduction: Ansible is simple open source software provisioning, configuration management, and application deployment tool which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Advantages of Ansible

Free: ansible is an open-source tool

Simple to setup and use: Basic Linux command skills are the only requirement to start using ansible

Powerful: Ansible lets you model complex network automation and configuration management workflows

Flexible: Entire network provisioning or application deployment can be orchestrated completely. Flexible to customize as per user needs

Agentless: No Separate software required to be installed on the target systems or network devices. Ansible uses ssh to securely connect and manage the devices. No need to operate any specific firewall ports to allow ansible communication

Ansible for network Automation

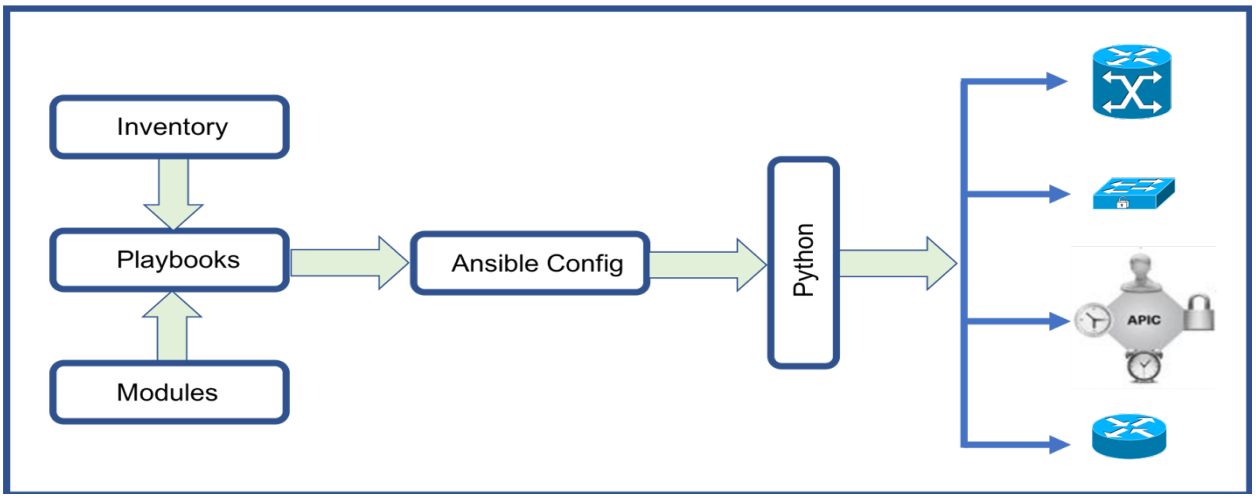
Entire Network can be managed using this single automation tool. With Ansible, we can

1. Automate routine tasks, to speed routine network changes and let employees engage in strategic tasks
2. Separate Ansible data model (represented by playbook or roles) from execution layer(via ansible modules- helps to manage heterogeneous network devices
3. Secure communication with network devices using http or https
4. Reuse of community or vendor generated playbooks and roles helps to accelerate network automation projects

Ansible Use Cases

1. Generate device configurations for new deployment or change management
2. Save and Collect device configurations from large number of devices
3. Push out configurations to a large number of devices
4. Upgrading operating systems and installing patches

Understanding Ansible Workflow Architecture



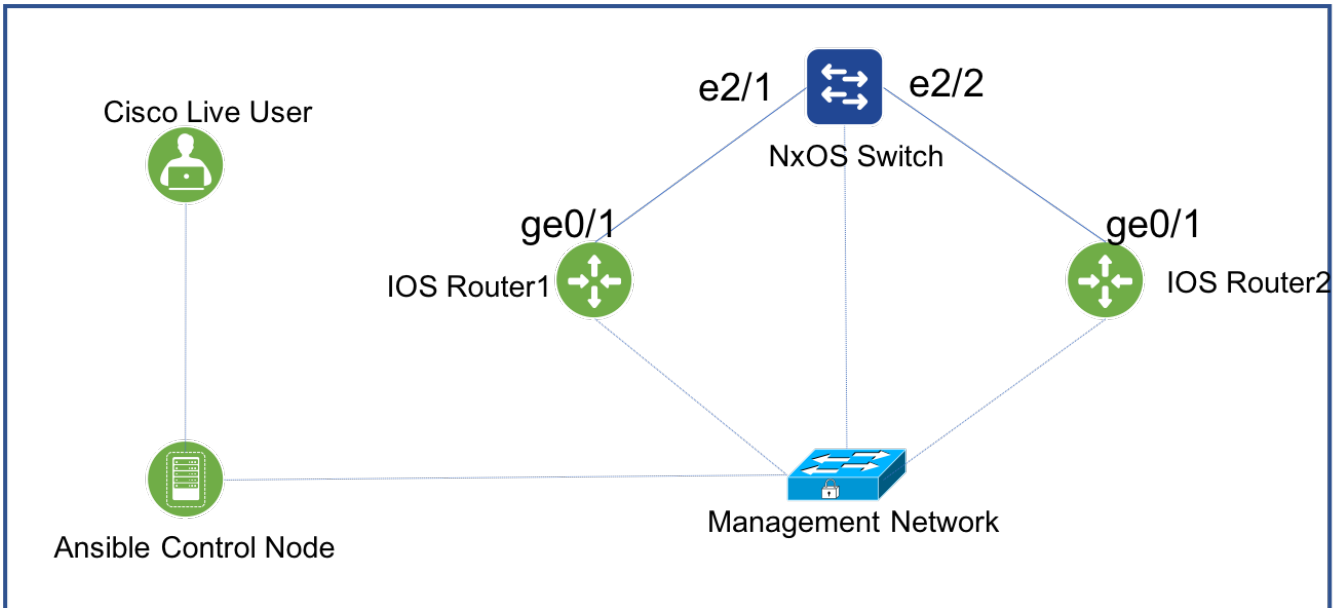
Ansible utilizes modules written in python by different vendors and community for individual devices or platforms to orchestrate the configurations and management of different network platforms.

Playbooks are the actual scripts, which points to the inventory files for the target hosts against which the scripts should be executed. Modules are called with in the playbook to perform individual tasks.

Lab Outline:

Lab 0: Connect to dcloud lab network and understand the lab topology and access details
Lab 1: Understand and familiarize with the ansible building blocks
Lab 2: Build and execute the first ansible playbook to automate network device operation
Lab 3: Build and execute more uses cases for network device management using ansible.
Lab 4: Build and execute ansible scripts to automate common use cases

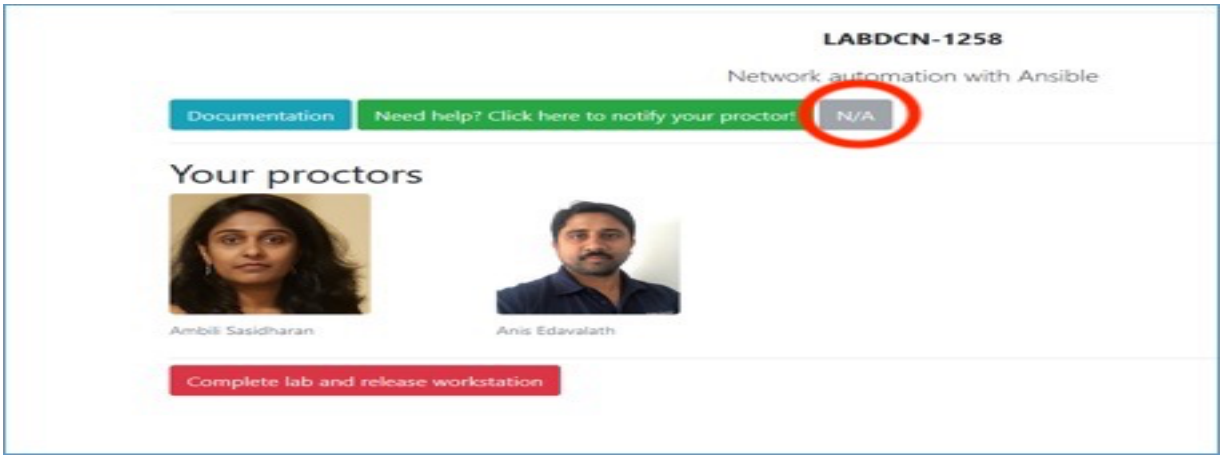
Lab Topology



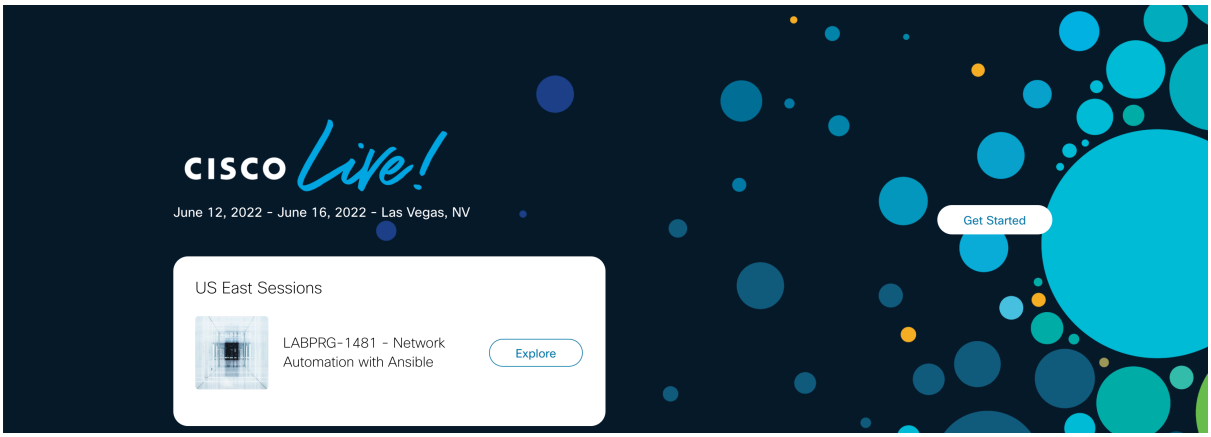
Note: The Ansible Control Node provides all the required access to the IOS and NxOS devices

Connecting to the WIL Assistant and accessing the Lab

Step 1: From the WIL assistance home page @ www.wilassiatance.com , click on START and search for the lab 1258 and click on START. You will see the page as below. Click on the buton N/A to select the Pod.



Step 2: Please click on explore to get started with the lab .

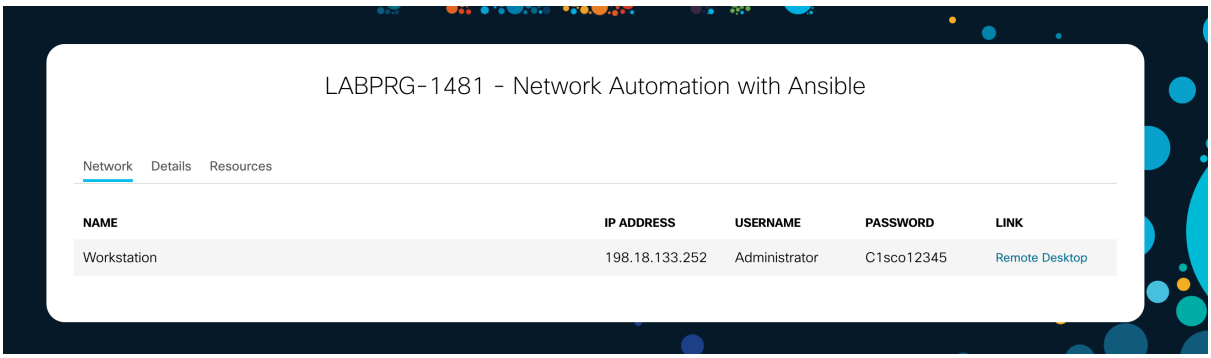


Step 3: Open the lab guide and follow the lab guide to connect to the RDP session and proceed to the lab exercises.

Lab 0: Connect to dCloud lab network and verify the connectivity

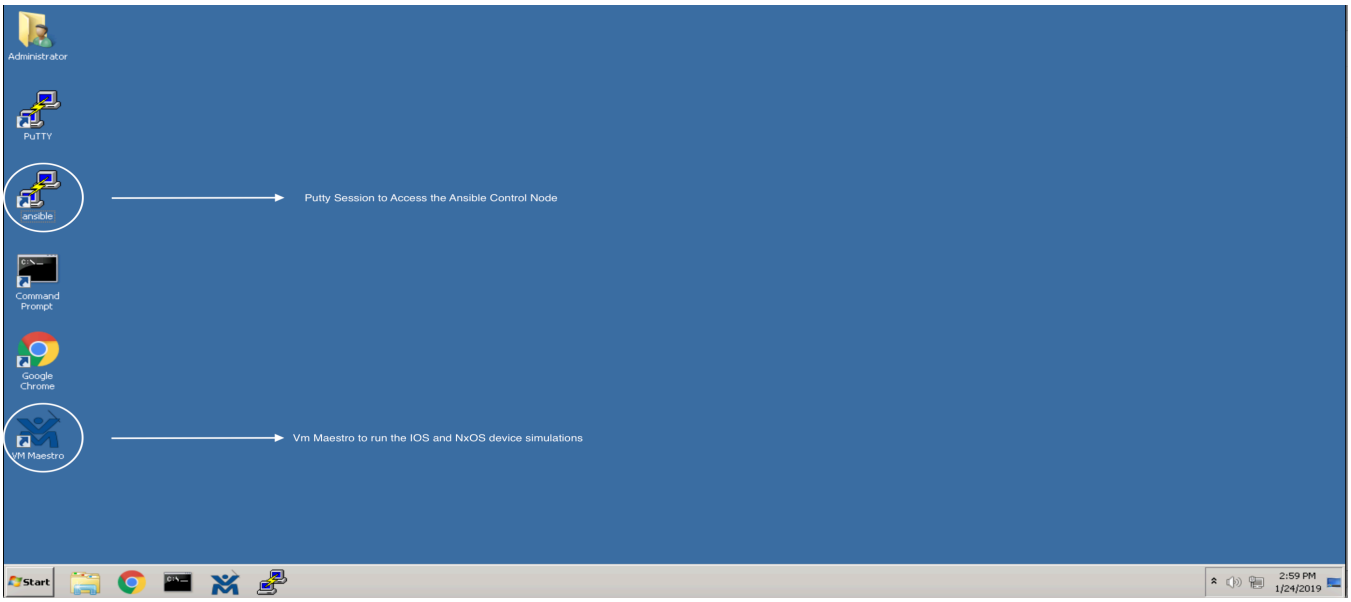
This lab leverages Cisco dCloud lab environment, which uses virtual servers and switches that runs on a simulator. These devices have most of the functions of actual hardware based devices, but there is no data plane traffic.

Click on Remote Desktop link to open a Remote Desktop Connection (RDP) to the lab workstation.

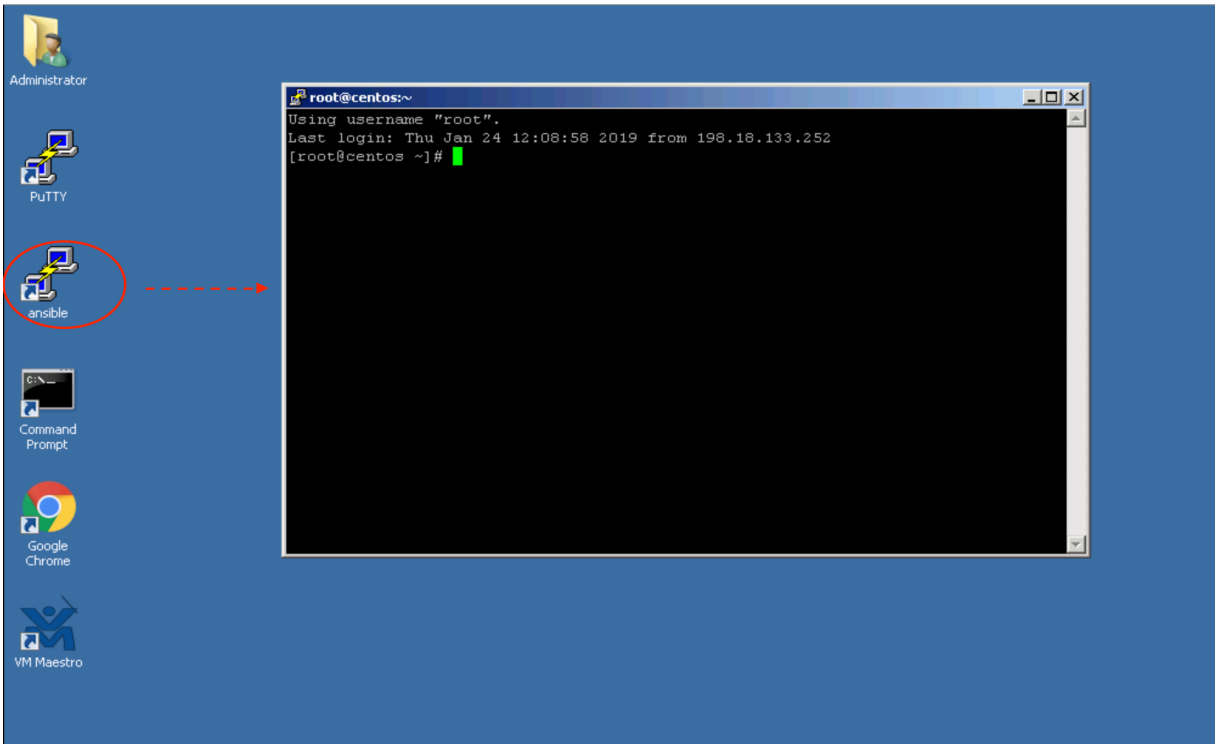


This will establish RDP session and open the windows screen as below. Proceed to the next section to verify the device connectivity

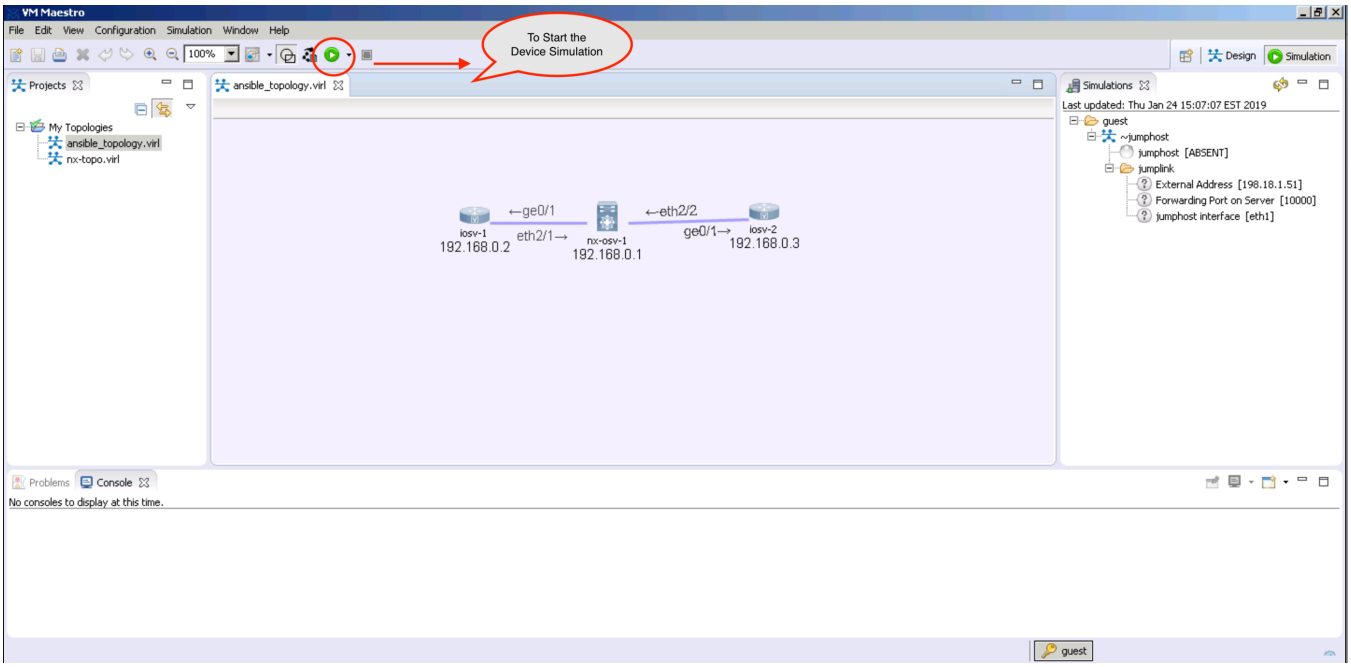
Step 1: After establishing the RDP connection, you will see a similar desktop screen setup as shown below. Use the tools as needed to complete the lab tasks.



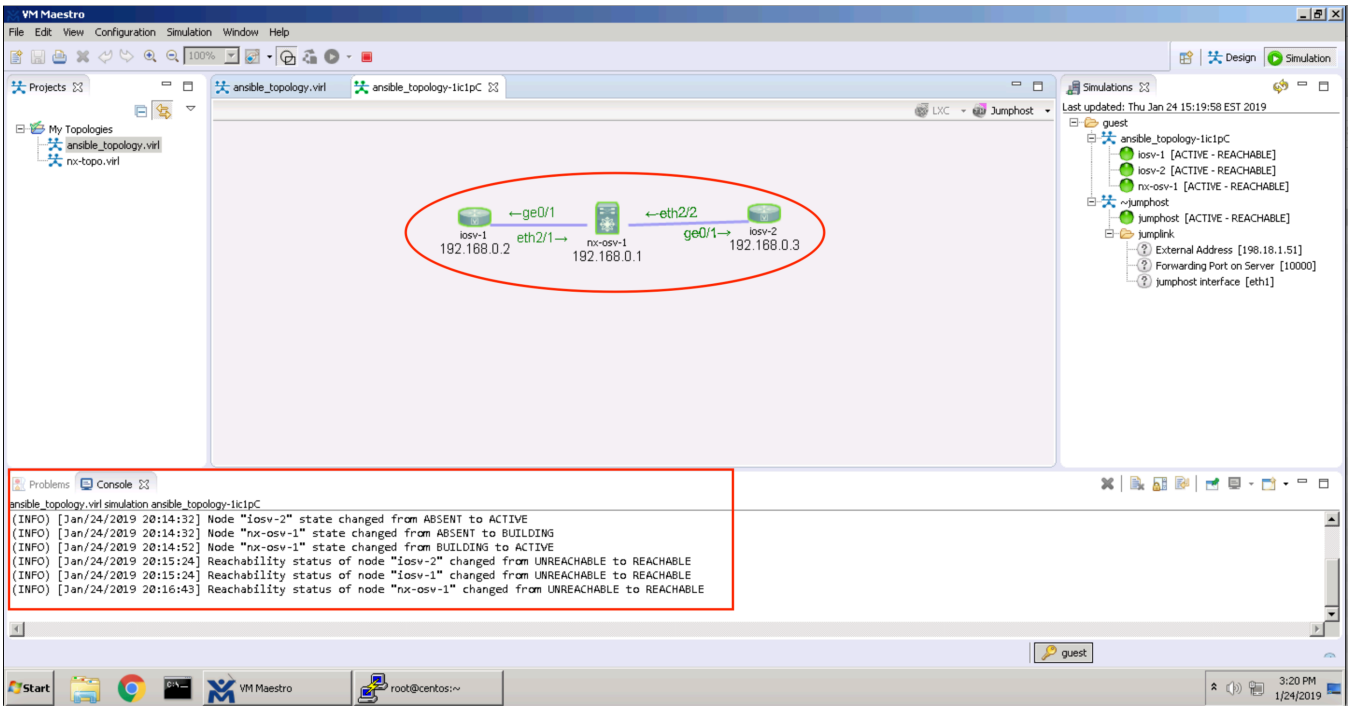
Step 2: In the above window, Double click on the ansible putty session . This will bring up the session to the ansible control node



Step 3: Open the VM Maestro application and click on the launch button to start the simulation. Wait for all the devices to come online.



Step 4: Wait for the devices to come online . The following messages will be displayed in the console of the VM Maestro, and the color of the devices will be changed to green.

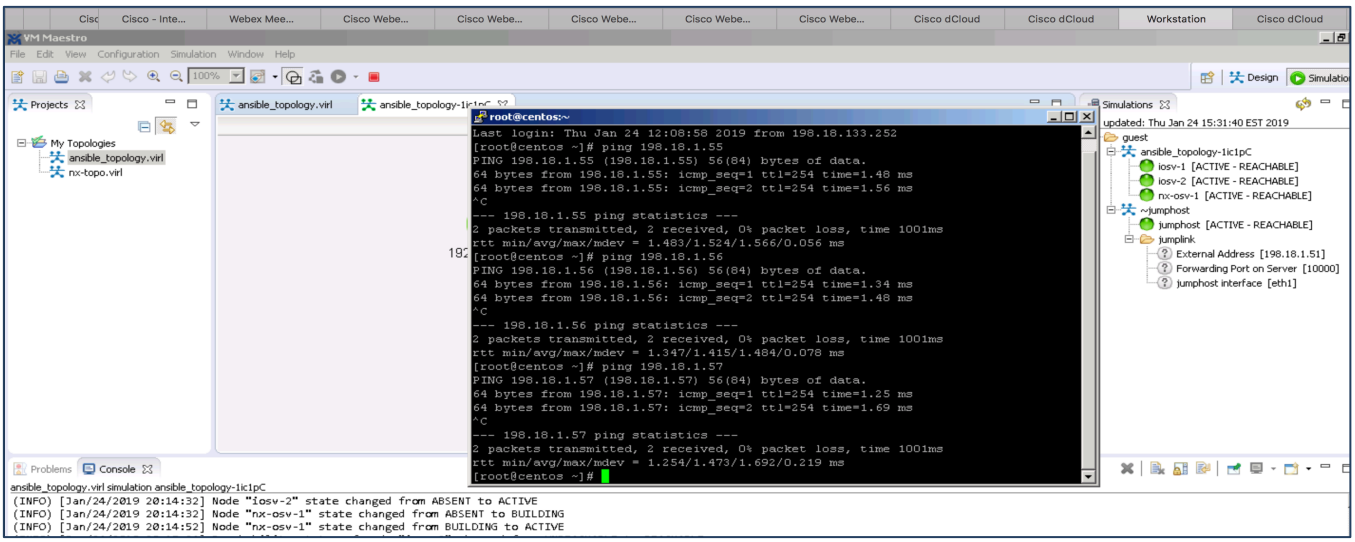


Step 5: Ping reachability from the ansible control node to each of the VMs.

iosv1: 198.18.1.55

iosv2: 198.18.1.56

nxosv1: 198.18.1.57



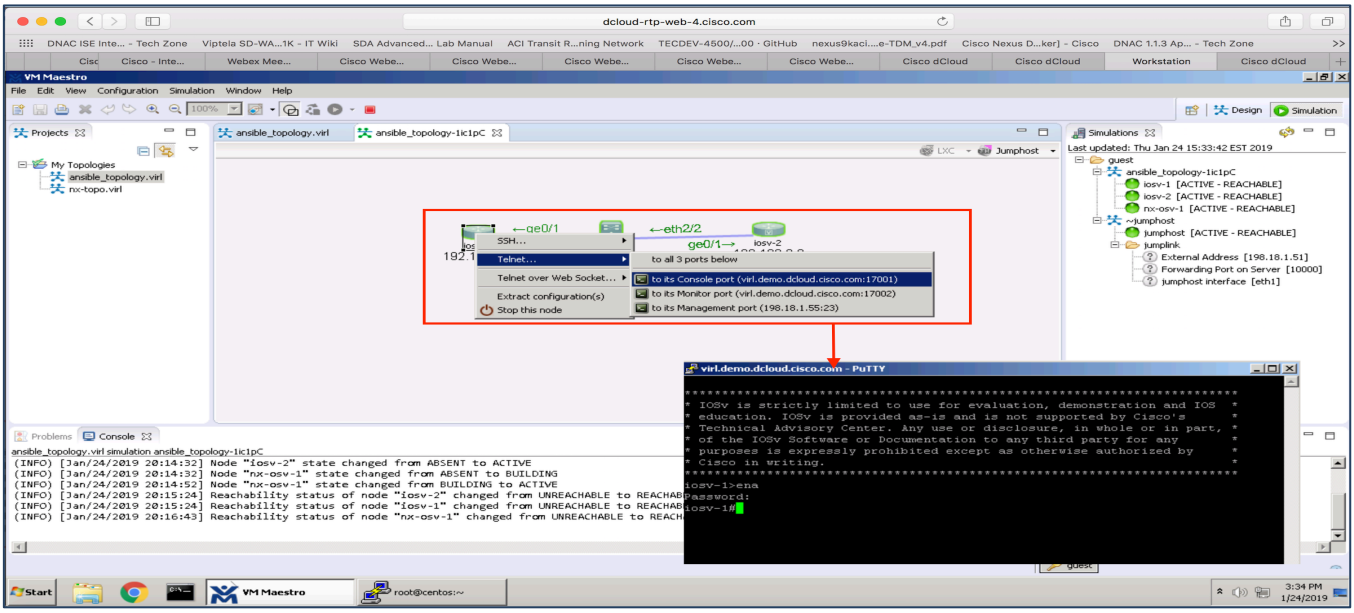
Note: In few instances, the NxOs device will not be reachable . In this case please issue the following sequence of commands to flap the management interface of nxos device

```
Enter configuration commands, one per line.  End with CNTL/Z.
nx-osv-1(config)# int mgmt 0
nx-osv-1(config-if)# shut
Shutting down this interface will drop all telnet and SSH sessions. Do you wish
to continue(y/n)? [no] y
nx-osv-1(config-if)# no shut
nx-osv-1(config-if)#
```

Step 6: Connect all the three devices by ricght clicking on the device and select → **telnet → to its console port(virl.demo.dcloud.cisco.com:17001)** to console and keep it ready to verify the config changes pushed by ansible

Userid: cisco

Password: cisco



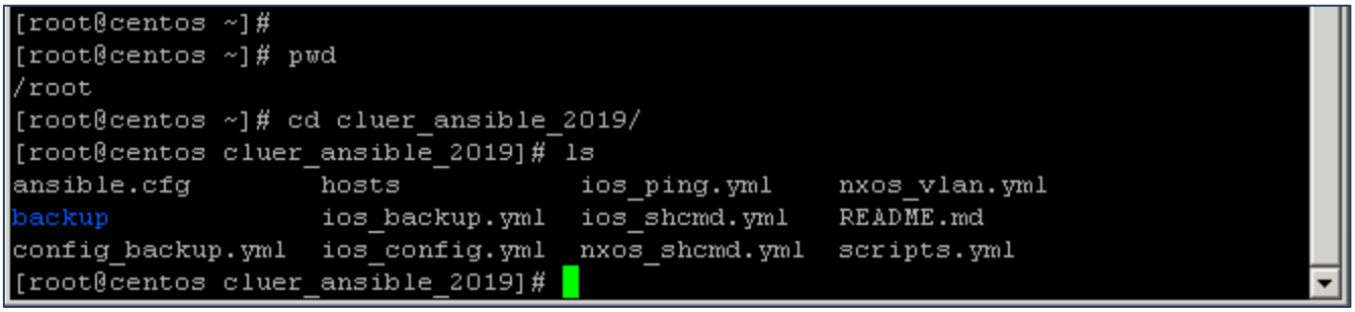
Lab 1: Explore and familiarize the Components of Ansible

Step 1: Connect to the ansible controller machine and change to the directory **LABDCN-1258.**

Control Node:

This is the ansible management node where ansible will be installed. This machine holds the playbooks and inventory files necessary to drive ansible. Control machines reach out to networks devices or other target hosts on which the action takes place.

Explore the contents of the directory using **"ls"**



Step 2: Ansible Config File

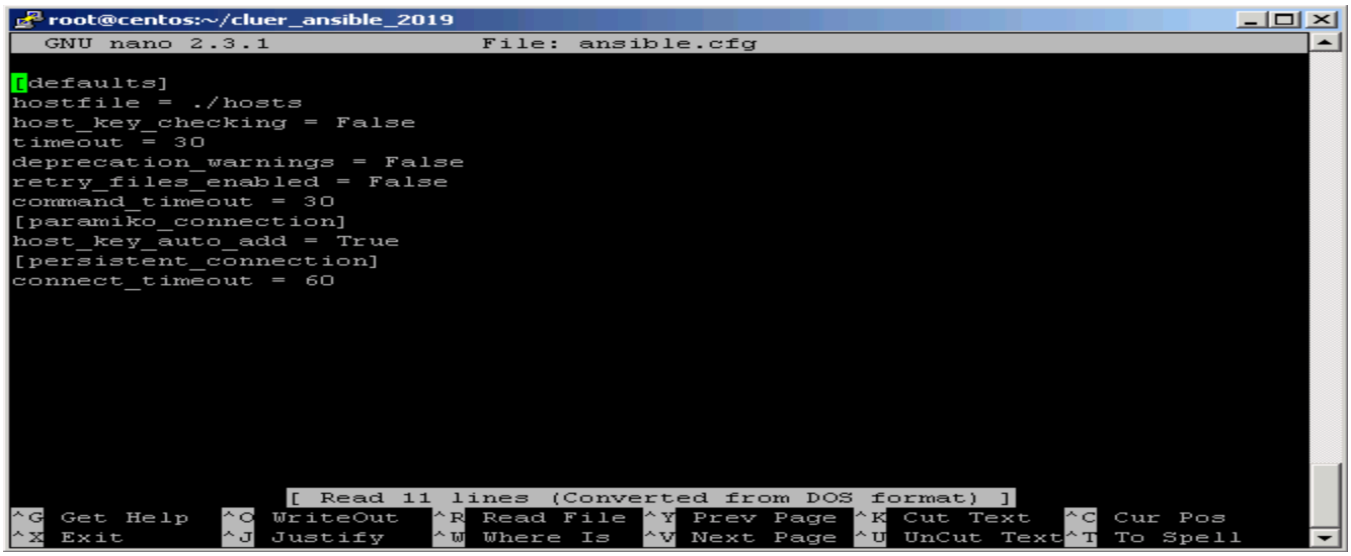
Ansible config file is the file where you adjust the default settings based on your

requirements. This file can be modified any time to customize the environment.

The key point to note here is the location of the hosts file. This is the pointer to the inventory file, which contains the list of all the hosts/network devices that we are managing using ansible.

Open the config file "ansible.cfg" using the editor nano

```
nano ansible.cfg
```



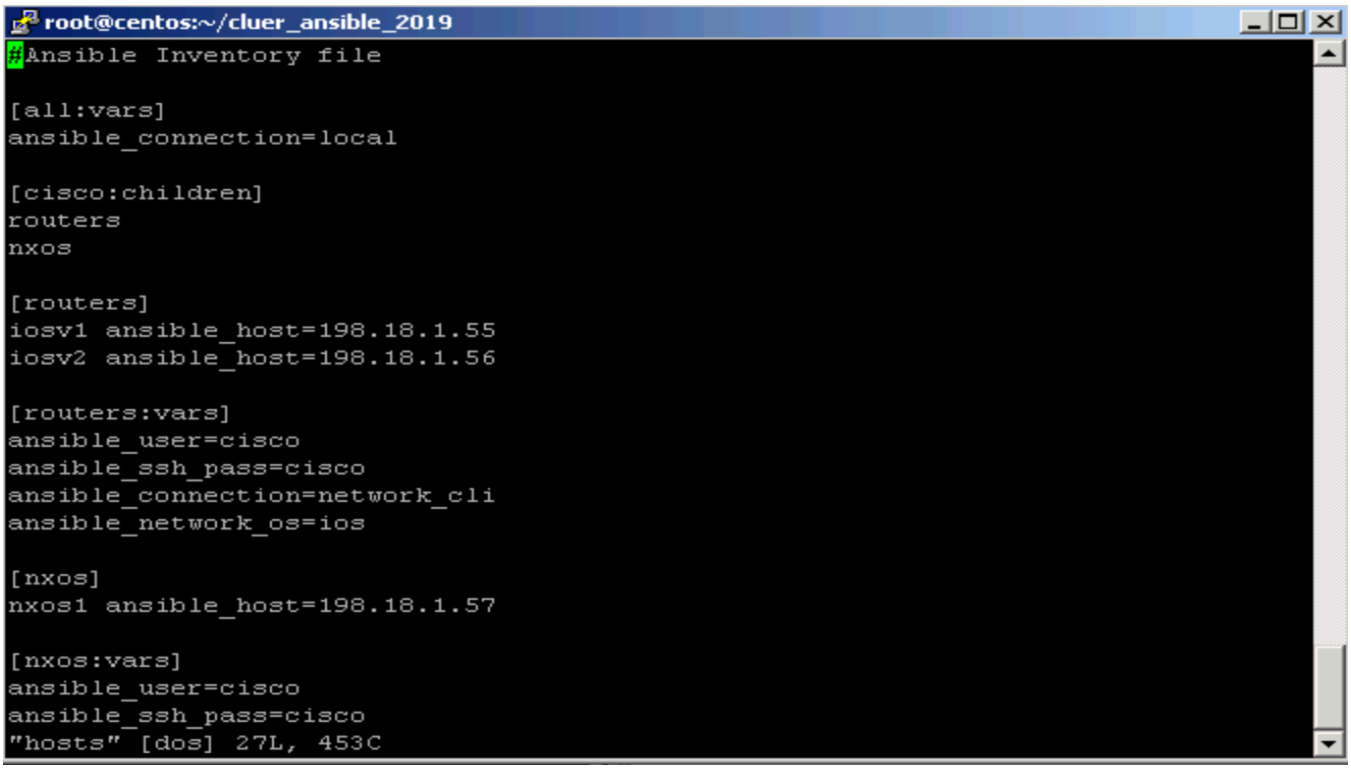
Step 3: Ansible Inventory File

Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in ansible's inventory, which defaults to being saved in the location **/etc/ansible/hosts**. For this lab convenience , we changed the inventory file to the same working directory . Inventory file can be written in INI format(which is the default) or YAML format. We have used the INI file format in this lab

Specific groups are created for "routers" and "nxos" in the inventory file. Creating the groups helps to execute the ansible playbook on a group of devices as needed.

Open the inventory file with the following command and explore the grouping and credentials provided to the IOS and NxOS devices in the lab

```
nano hosts
```



Groups In Ansible Inventory: The headings in brackets are group names, which are used in classifying systems and deciding what systems you are controlling at what times and for what purpose.

In our example file above, [routers] and [nxos] are two different groups

Groups of groups can be created by using the **:children** suffix in the ini format or **children:** entry in YAML format

In our example here, two groups **routers** and **nxos** are are added as the groups under the main group **cisco** using the snippet below

```
[cisco:children]
routers
nxos
```

The group **all** is a builtin default group, which includes every single host defined in the inventory

Variables in Ansible Inventory: similar to groups, variables can be created by using **vars:** in INI format or by using **:vars** in YAML format

Global variables: variables defined in the format **[all:vars]** is the global variable and is applicable for all hosts.

Group specific variables:variables defined in the format **[<group-name>:vars]**. In our example above **[routers:vars]** and **[nxos:vars]** are two group specific variables and are relevant only for hosts belonging to that specific group

Step 4: Ansible Playbooks

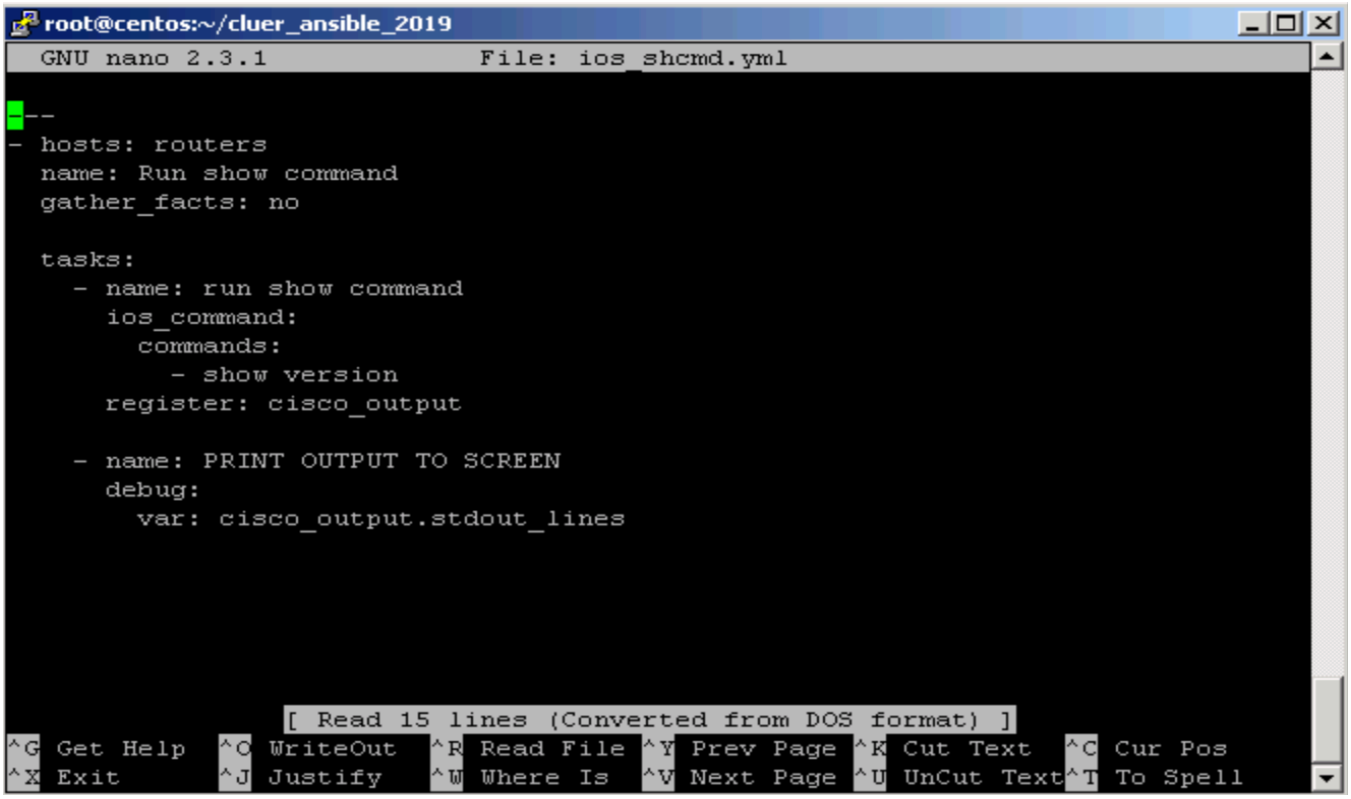
Ansible playbooks are the instruction manual for the tasks that need to be executed on devices. Playbooks are written in a simple markup language - YAML . Each playbook is composed of one or multiple plays, and the goal of a play is to map a group of hosts to well-defined roles, represented by tasks. Playbooks can orchestrate the steps of any manually ordered task, and can execute tasks at the same time or at different times.

Lab 2: Run the First Ansible Playbook

Playbooks can be executed by issuing the command **ansible-playbook <playbook-name>** on the ansible control node.

Lab Exercise 1: ios_shcmd.yml

Open the file with **nano ios_shcmd.yml**



The three hyphen at the top of the file --- marks the beginning of the playbook .

A single hyphen '-' denotes a new list in YAML file . Each list can contain multiple key value pairs. Here the first list specifies the following

hosts: hosts against which these tasks need to be executed. This script which execute the tasks against the IOS devices.

Name: name of the list

Gather_facts: This module is automatically called by playbooks to gather useful variables about remote hosts that can be used in playbooks. It can also be executed directly by /usr/bin/ansible to check what variables are available to a host. Ansible provides many *facts* about the system, automatically.

Tasks: This provide the list of tasks that need to be executed. We are running two tasks in this playbook.

1. Execute the show command on a router.
   - **ios_command:** This is the module used to execute the command on a cisco ios device. Sends arbitrary commands to an ios node and returns the results read from the device. This module includes an argument that will cause the module to wait for a specific condition before returning or timing out if the condition is not met.This module does not support running commands in configuration mode.
   - The output of the command will be stored in a system variable array called **result[]**
   - The command **register** will register the output of the above command which is stored in **result[0]** to the provided variable**.** We are saving the output of **show version** to the variable **cisco_output**

2. Display the output of the show command on to the screen

   - **Debug:** The debug module is used to prints statements during execution and can be useful for debugging variables or expressions without necessarily halting the playbook
   - Debug module takes the variable that need to be printed as the attribute. We have provided the previously saved variable **cisco_output**

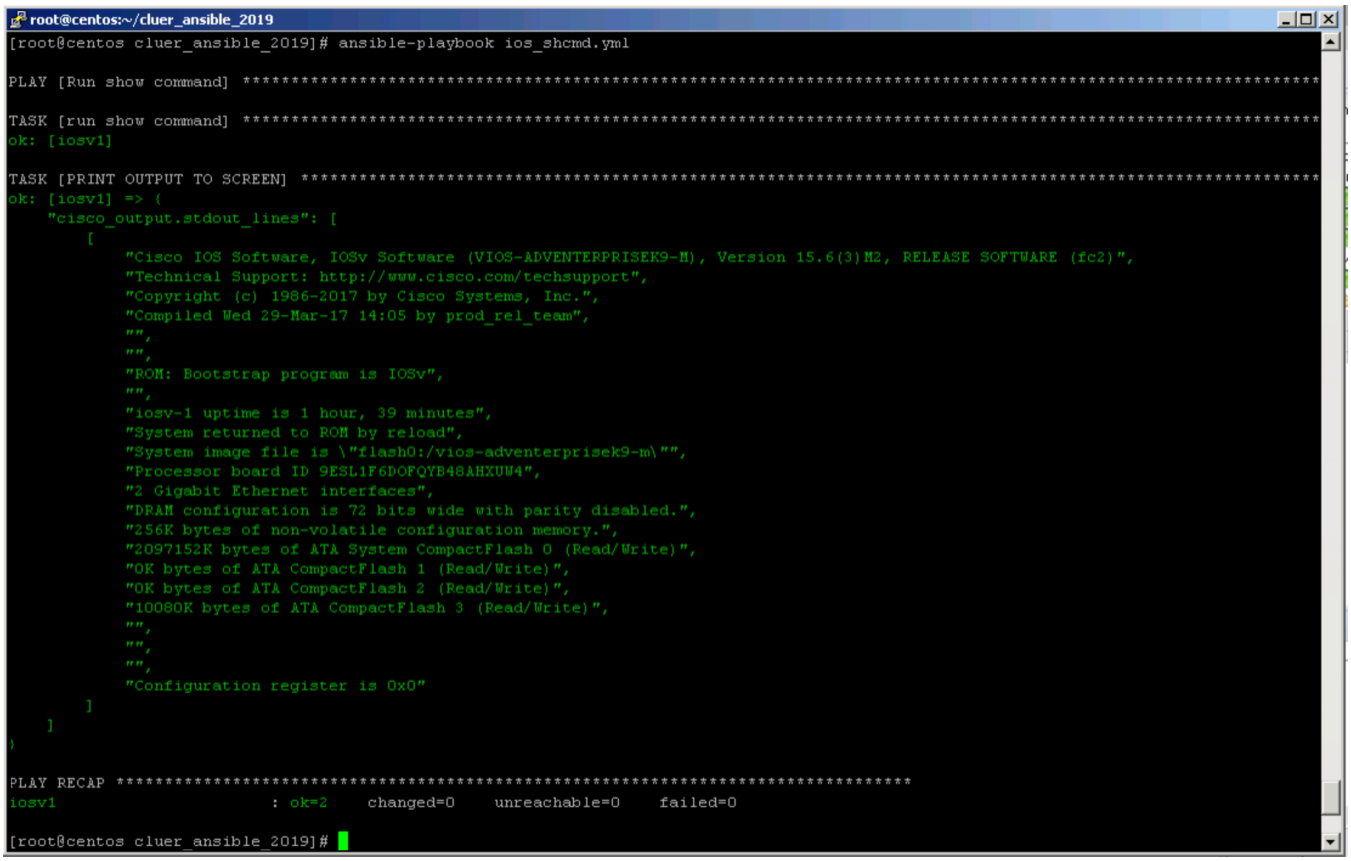
Execute the playbok with the command

```
ansible-playbook ios_shcmd.yml
```

Optional Challenge :

Execute the following multiple commands using the ios_command module .

1. Show version
2. Show interface summary



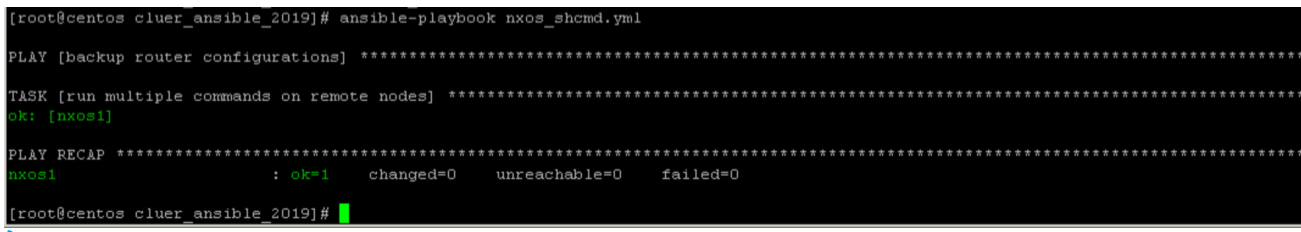
Verification: Verify that there is no error in the output and match the output with the screenshot above

Lab 3: More Ansible Playbook Use cases

<u>LAB Exercise 1:</u> Run the script nxos_shcmd.yml using the command

```
ansible-playbook nxos_shcmd.yml
```

This example is similar to the first script , but specific modules are available for NxOS devices and we have to use them as the behavior is different between NxOS and IOS. Here we used the nxos_command module .



Verification: Verify that there is no error in the output and match the output with the screenshot above

Optional Challenge: This example does not print the show command output on to the screen . Modify the script to print the output on to screen. Compare the difference in output between stdout and stdout_lines

LAB Exercise 2: Execute generic configuration commands on Devices

In this exercise , we will send configuration commands to the nxos device.

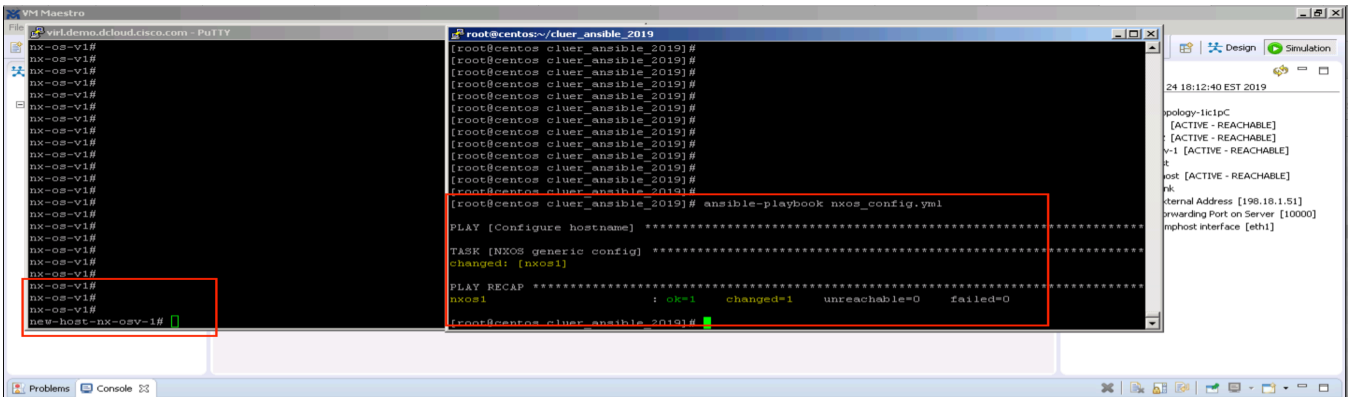
The module **nxos_config** is used to configure any config level commands.

Open the playbook with nano nxos_config.yml

The config commands specified after the lines will be configured on to the device. By default the config will not be saved after it is modified. The parameter **save_when** should be provided with the any one of the value from **"always/never/modified/changed".** We have used the value "changed" to forcefully save the configuration to nvram after the configuration changes are made

Execute the playbook nxos_config.yml with the command

ansible-playbook nxos_config.yml



Verification: Verify that there is no error in the output and match the output with the screenshot above **.**

Verify that the hostname is getting to the new value that is set in the script

Optional Challenge: Note any difference in the output from the previous execution. Modify the script to print out the output on to the screen . See if the output is printing or not ? If not Why ? Compare and evaluate the scenario with the previous exercise with the error message on variables

Use register to store the output and debug with stdout to print the output.

LAB Exercise: 3 Execute specific Config modules to configure devices

In this exercise we will configure vlans on the NxOS device by utilizing the specific module developed for configuring vlan - **nxos_vlan** . Please refer to the ansible documentation @ https://docs.ansible.com/ansible/latest/modules/ for the complete list of modules that are available.

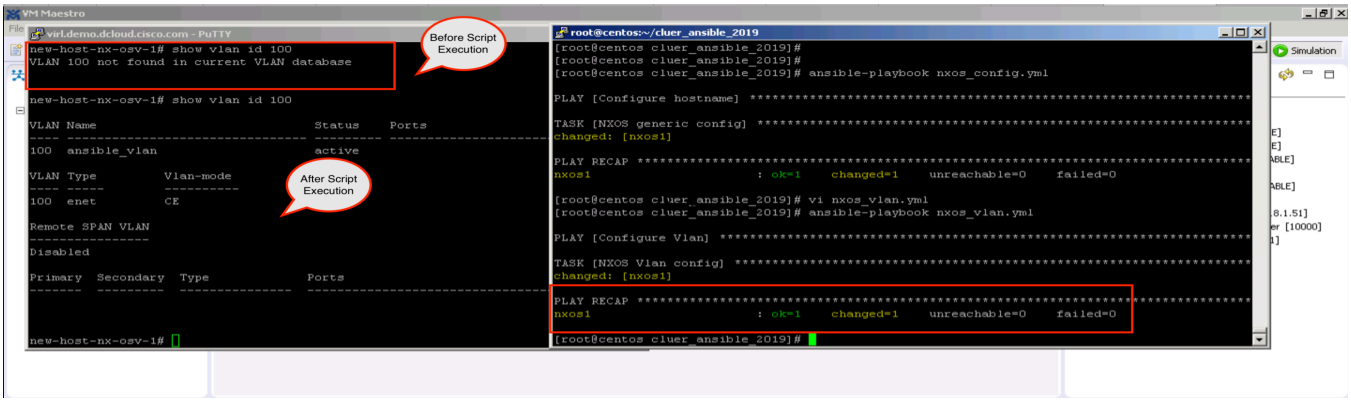
Open the file nxos_vlan.yml – nano nxos_vlan.yml

```
root@centos:~/cluer_ansible_2019
---
- hosts: nxos
  name: Configure Vlan
  gather_facts: no

  tasks:
    - name: NXOS Vlan config
      nxos_vlan:
        vlan_id: 100
        admin_state: up
        name: ansible_vlan
```

The module **nxos_vlan** takes specific attributes for vlan configuration some of the attributes are provided below.

nxos_vlan module Attributes	Values
vlan_id	<any vlan id that need to be configured>
vlan_range	<range of vlans that need to be configured>
admin_state	up / down
name	<name of the vlan>



Verification: Verify that there is no error in the output and match the output with the screenshot above

Make sure that the provided vlans are configured on to the system from the system cli.

Optional Challenge : Try to change the script with a new attribute value pair – " vlan_range"

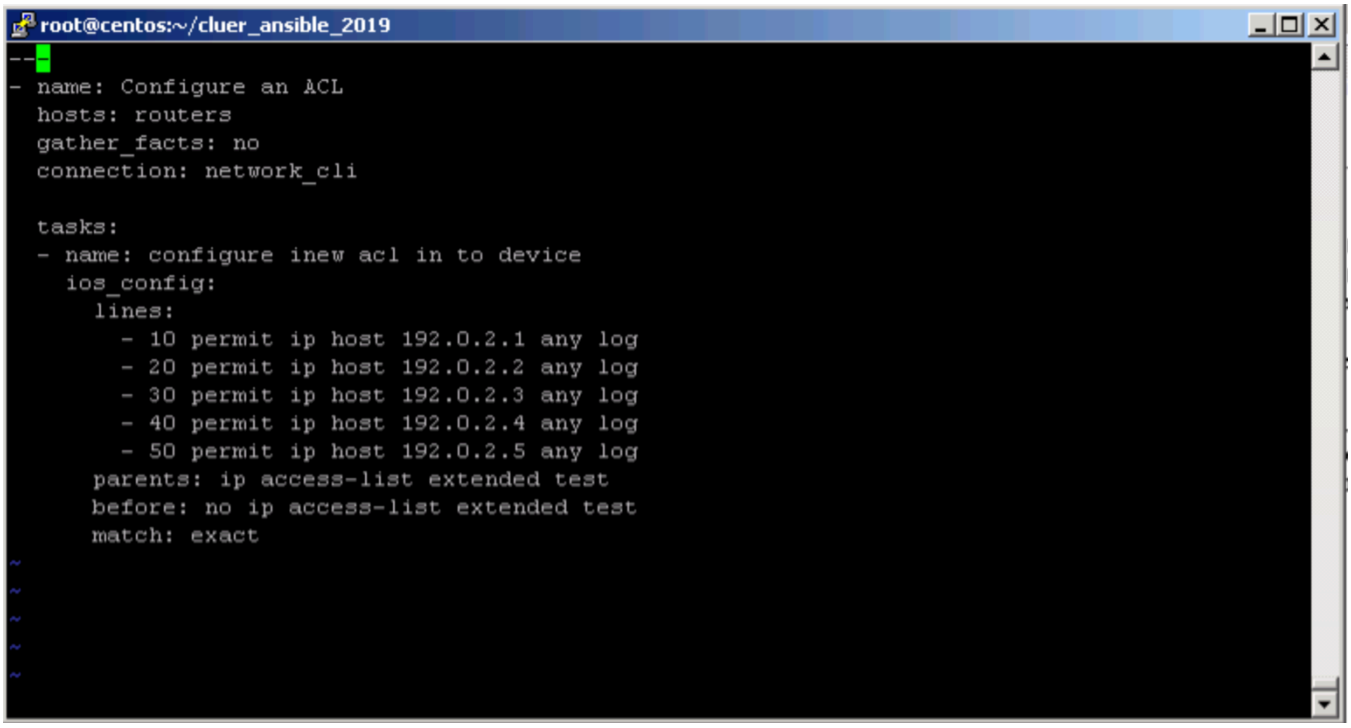
Note: Please make sure you delete the line for vlan name when using vlan-range command as both commands are mutually exclusive

LAB Exercise: 4 Execute multiline Config commands devices

In this lab exercise, we will configure multiline config commands on to the IOS device.

Open the script file ios_config.yml

nano ios_config.yml



To configure multiline commands, multiple config lines are entered under the lines statement.

New attributes in this script :

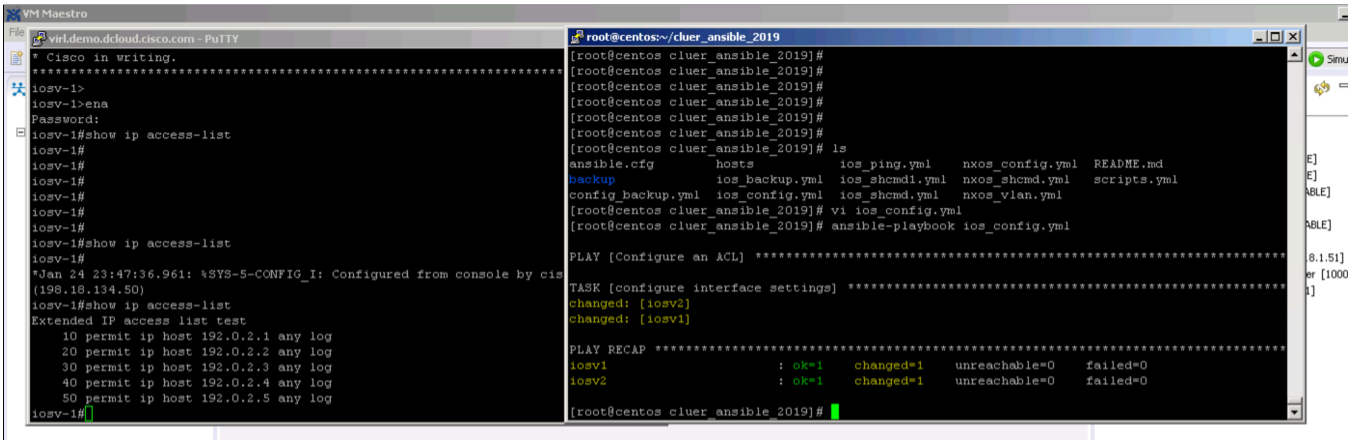
before : The ordered set of commands to push on to the command stack if a change needs to be made. This allows the playbook designer the opportunity to perform configuration commands prior to pushing any changes without affecting how the set of commands are matched against the system.

Herwe we re using this attribute to delete any access-list by the same name if it exists.

parents: The ordered set of parents that uniquely identify the section or hierarchy the commands should be checked against. If the parents argument is omitted, the commands are checked against the set of top level or global commands.

match: Instructs the module on the way to perform the matching of the set of commands against the current device config. If match is set to ***line***, commands are matched line by line. If match is set to ***strict***, command lines are matched with respect to position. If match is set to ***exact***, command lines must be an equal match. Finally, if match is set to ***none***, the module will not attempt to compare the source configuration with the running configuration on the remote device.

Execute the script **ios_config.yml**



Verification: Verify that there is no error in the output and match the output with the screenshot above

Make sure that the provided access lists are configured o to the device by using the CLI **"show ip access-list"**

Lab 4: Network Operations Use case Examples using Ansible

Lab Task 1: Configure backup of the router configurations using Ansible

Taking device backups at regular intervals is one of the most common use cases in network operations. In a non controller based network environment , ansible can be used to perform the backup configuration of network devices. In this example, the playbook config_backup.yml performs the backup configuration of IOS device and NxOS device and store them in the backup folder.

Open the file config_backup.yml

```
nano config_backup.yml
```

```
root@centos:~/cluer_ansible_2019
---
- name: Backup device config
  hosts: iosv1, nxos1
  connection: network_cli

  tasks:
    - name: Backup nxos config
      nxos_config:
        backup: yes
      when: ansible_network_os == 'nxos'

    - name: Backup IOS config
      ios_config:
        backup: yes
      when: ansible_network_os == 'ios'
~
```

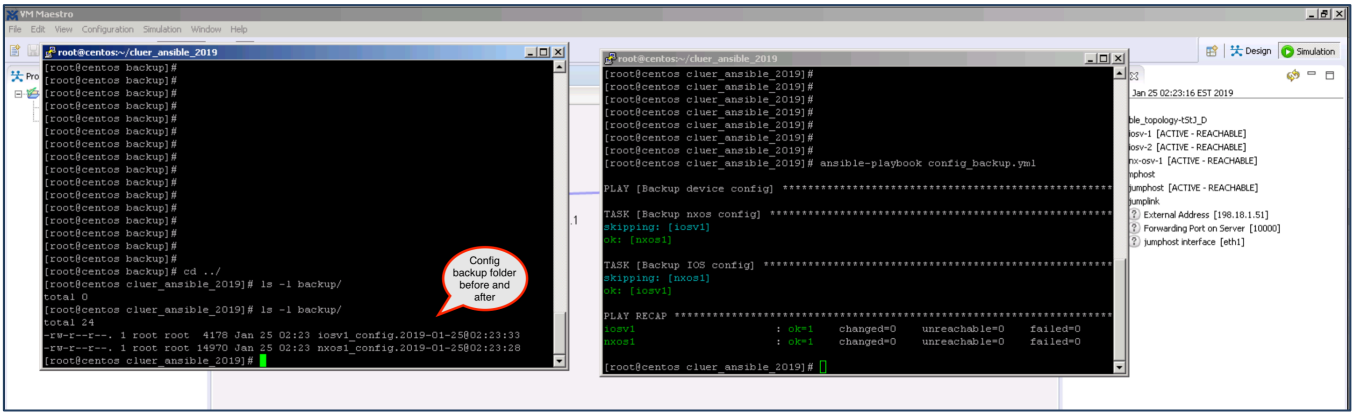
We are using the **ios_config** and **nxos_config** modules with the backup attribute set to yes.

Also , note the usage of conditional "**when:"** , which checks for the operating system using the system variable, **ansible_network_os.** The script will be executed on the device which maxes the confition and skips the device where the condition is not matched.

Backup attribute: This argument will cause the module to create a full backup of the current running-config from the remote device before any changes are made. The backup file is written to the backup folder in the playbook root directory. If the directory does not exist, it is created.

Execute the playbook using the command

```
ansible_playbook config_backup.yml
```



Verification: Verify that there is no error in the output and match the output with the screenshot above

Verify that the backup directory is created and is populated with the backup configuration from both ios and nxos devices.

Verify that the script follows the when condition , by executing for the device which match the “when “ condition and skip[s the device where the condition is not met

Lab Task 2: Reset the lab to original defaults

Execute the ansible playbook **lab_reset.yml**, to reset the lab to default values

This will reset the configuration changes made in to the default values and make the system ready for jse to the next participant.

We are utilizing the regular config modules to depete the configs that are created as part of the exercises.

To delete the linux directory , the file module is utilized with the attribute **state: absent,** which deletes the directory.

root@centos:~/cluer_ansible_2019

```
---
- name: reset lab
  hosts: iosv1, iosv2, nxos1
  connection: network_cli

  tasks:
    - name: reconfigure hostname
      nxos_config:
        lines: hostname nx-osv1
      when: ansible_network_os == 'nxos'

    - name: remove vlans from nxos
      nxos_config:
        lines: no vlan 2-1001
      when: ansible_network_os == 'nxos'

    - name: remove access-lists from ips devices
      ios_config:
        lines: no ip access-list extended test
      when: ansible_network_os == 'ios'


    - name: remove backup directory
      file:
        path: backup
        state: absent
```

Verification: make sure that there is no errors after running this script

Conclusion

You have successfully completed the lab. Please handover the WISP Lab card to the front desk. To continue your journey with ansible, please refer to the Reference links . Ansible playbooks for different uses cases are available for download and use at www.github.com .

Thank you for attending this lab, please share your valuable feedback.

Reference

Additional Reading and Reference:

https://developer.cisco.com/video/net-prog-basics/

https://www.ansible.com/resources/ebooks

https://learninglabs.cisco.com/tracks

http://shop.oreilly.com/product/0636920065500.do