**Lab Guide: Using Ansible Vault for Securing Sensitive Data**

**Objective:**

Learn how to use Ansible Vault to encrypt, decrypt, view, edit, and rekey files containing sensitive data, such as passwords and other secrets.

**Prerequisites:**

* Basic understanding of Ansible.
* Ansible installed on your control node.

**Step 1: Introduction to Ansible Vault**

Ansible Vault allows you to keep sensitive data, such as passwords or keys, in encrypted files, rather than as plaintext in your playbooks or roles.

**1.1. Ansible Vault Help**

To explore the available commands and options with Ansible Vault, use:

bash

Copy code

ansible-vault -h

**Step 2: Encrypting a File with Ansible Vault**

**2.1. Encrypt a YAML File**

You can encrypt a YAML file, such as first.yml, using the following command:

bash

Copy code

ansible-vault encrypt first.yml

After running this command, you will be prompted to enter a password. This password will be required to view or use the encrypted file.

**2.2. Verify Encryption**

After encryption, if you try to view the contents of first.yml, you'll see that it is now encrypted:

bash

Copy code

cat first.yml

**Step 3: Running Playbooks with Encrypted Files**

**3.1. Run an Encrypted Playbook**

To run a playbook that has been encrypted, you need to provide the vault password:

bash

Copy code

ansible-playbook first.yml --ask-vault-pass

This command will prompt you to enter the vault password to decrypt and run the playbook.

**Step 4: Decrypting a File with Ansible Vault**

**4.1. Decrypt a YAML File**

If you need to decrypt a file that was encrypted with Ansible Vault, use:

bash

Copy code

ansible-vault decrypt first.yml

You will be prompted to enter the vault password used during encryption. Once decrypted, the file will be in plain text.

**Step 5: Viewing Encrypted Files without Decrypting**

**5.1. View an Encrypted File**

You can view the contents of an encrypted file without decrypting it permanently by using:

bash

Copy code

ansible-vault view sec.yml

This will show the decrypted contents of sec.yml temporarily.

**Step 6: Editing Encrypted Files**

**6.1. Edit an Encrypted File**

To edit an encrypted file, use the following command:

bash

Copy code

ansible-vault edit sec.yml

This command will allow you to make changes to the file without decrypting it permanently.

**Step 7: Rekeying an Encrypted File**

**7.1. Rekey an Encrypted File**

If you need to change the vault password for an encrypted file, you can rekey it with:

bash

Copy code

ansible-vault rekey sec.yml

This command will prompt you to enter the old password and then enter a new password twice.

**Conclusion:**

In this lab, you learned how to use Ansible Vault to encrypt and decrypt files, view and edit encrypted files, and change the vault password for existing encrypted files. This is essential for maintaining the security of sensitive information in your Ansible workflows.

**Lab Guide: Configuring Cisco Devices Using with\_dict in Ansible**

**Objective:**

Learn how to use the with\_dict loop in Ansible to iterate over a dictionary of interface facts from Cisco devices and apply configurations based on the line protocol status.

**Prerequisites:**

* Basic understanding of Ansible and Cisco IOS.
* Ansible installed on your control node.
* Cisco devices accessible from your Ansible control node.

**Step 1: Prepare the Ansible Playbook**

**1.1. Create the Playbook**

Create a playbook named test2.yml that performs the following tasks:

yaml

Copy code

---

- name: Configure Cisco Devices

hosts: devices

gather\_facts: no

tasks:

- name: Gather facts about interfaces

ios\_facts:

gather\_subset: interfaces

register: raman\_ios\_facts

- name: Output all info

debug:

var: raman\_ios\_facts

- name: Print line protocol status for all interfaces

debug:

msg: "Interface {{ item.key }} has lineprotocol: {{ item.value.lineprotocol }}"

with\_dict: "{{ raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces }}"

- name: Configure interfaces if line protocol is up

ios\_config:

lines:

- description Configured by Khanna

- no shutdown

parents:

- "interface {{ item.key }}"

ignore\_errors: true

with\_dict: "{{ raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces }}"

when: item.value.lineprotocol == 'up'

**Step 2: Run the Playbook**

**2.1. Execute the Playbook**

Run the playbook using the following command:

bash

Copy code

ansible-playbook test2.yml

You will need to ensure that the Cisco devices specified in your inventory file are accessible and configured to allow Ansible to connect.

**Step 3: Understanding the Playbook Tasks**

**3.1. Gather Facts**

* **Task**: Gather facts about interfaces
* **Module**: ios\_facts
* **Purpose**: Collect detailed information about network interfaces on the Cisco devices and register the result as raman\_ios\_facts.

**3.2. Output All Info**

* **Task**: Output all info
* **Module**: debug
* **Purpose**: Print the gathered facts to help with debugging.

**3.3. Print Line Protocol Status**

* **Task**: Print line protocol status for all interfaces
* **Module**: debug
* **Purpose**: Iterate over the interfaces using with\_dict to display the line protocol status of each interface.

**3.4. Configure Interfaces**

* **Task**: Configure interfaces if line protocol is up
* **Module**: ios\_config
* **Purpose**: Apply configuration changes to interfaces where the line protocol is 'up'. This task uses with\_dict to iterate over interfaces and apply the configuration only when the condition item.value.lineprotocol == 'up' is met.

**Step 4: Debugging and Verification**

**4.1. Check Output**

Review the output from the debug tasks to ensure that the facts are gathered correctly and the conditions for configuration are accurate.

**4.2. Verify Configuration**

After running the playbook, verify the changes on your Cisco devices to ensure the configuration has been applied as expected.

**Conclusion:**

In this lab, you used with\_dict in Ansible to iterate over a dictionary of interface facts and apply configurations conditionally based on the line protocol status. This method is useful for dynamically managing device configurations based on their current state.

**Lab Guide: Managing Multiple Vault Passwords with Ansible Vault**

**Objective:**

Learn how to use multiple vault passwords with Ansible Vault for encrypting and decrypting files and running playbooks with different vault IDs.

**Prerequisites:**

* Basic understanding of Ansible and Ansible Vault.
* Ansible installed on your control node.
* Basic knowledge of working with encrypted files.

**Step 1: Encrypt Files with Multiple Vault Passwords**

**1.1. Encrypt a File with One Vault ID**

You can encrypt a file with a single vault password using:

bash

Copy code

ansible-vault encrypt --vault-id invpassword@prompt inv

This command will prompt you to enter the password for the vault ID invpassword and encrypt the file inv.

**1.2. Encrypt a File with Multiple Vault IDs**

To encrypt a file with multiple vault passwords, use:

bash

Copy code

ansible-vault encrypt --vault-id ymlpassword@prompt sec.yml

This command will prompt you for passwords associated with ymlpassword and encrypt the file sec.yml.

**Step 2: Running Playbooks with Multiple Vault Passwords**

**2.1. Run a Playbook with Multiple Vault IDs**

When running a playbook that uses files encrypted with different vault passwords, you need to provide all the vault IDs. For example:

bash

Copy code

ansible-playbook sec.yml -i inv --vault-id invpassword@prompt --vault-id ymlpassword@prompt

This command specifies both invpassword and ymlpassword to decrypt the necessary files and execute the playbook sec.yml.

**2.2. Run a Playbook with Multiple Vault Passwords for Another File**

If you have another file encrypted with different vault passwords, you can include those in your playbook run:

bash

Copy code

ansible-playbook third.yml -i inv --vault-id invpassword@prompt --vault-id ymlpassword2@prompt

In this example, third.yml is encrypted with ymlpassword2 and invpassword and is run with the corresponding vault IDs.

**Step 3: Additional Vault Operations**

**3.1. Encrypt a File with Different Vault Passwords**

You can encrypt a file with a different vault password using:

bash

Copy code

ansible-vault encrypt --vault-id ymlpassword2@prompt third.yml

If you want to use a second password, you can specify it with the --vault-id option.

**3.2. Verify Encrypted Files**

To ensure that files are encrypted correctly, you can use:

bash

Copy code

ansible-vault view sec.yml

ansible-vault view third.yml

These commands will prompt for passwords and display the decrypted content.

**Conclusion:**

In this lab, you practiced encrypting and decrypting files using multiple vault passwords with Ansible Vault. You also learned how to run playbooks that involve multiple vault IDs. This approach enhances security by allowing different sets of passwords for various files or environments.

**Lab Guide: Managing Encrypted Passwords with Ansible Vault and YAML Inventory**

**Objective:**

Learn how to use Ansible Vault to encrypt sensitive information, convert an inventory file to YAML format, and run a playbook using the encrypted data.

**Step 1: Gathering Initial Information**

1. **Show Current Users on the Cisco Device:**

Run the following command to see the current users configured on the Cisco device:

bash

Copy code

ansible devices -m ios\_command -a "commands='sh run | i user'"

1. **Create a New User Manually (Optional):**

To manually create a user, enter configuration mode and add the user:

bash

Copy code

Cat8000V#config

Cat8000V(config)#username testuser privilege 15 secret testpassword

Cat8000V(config)#end

Cat8000V#write memory

Confirm that the user has been added:

bash

Copy code

Cat8000V#sh run | i user

**Step 2: Encrypt a String with Ansible Vault**

1. **Encrypt the Password:**

Encrypt the password using Ansible Vault:

bash

Copy code

ansible-vault encrypt\_string --vault-id r1\_userid@prompt 'testpassword' --name 'ansible\_password'

You'll receive an output similar to this:

yaml

Copy code

ansible\_password: !vault |

$ANSIBLE\_VAULT;1.2;AES256;r1\_userid

33396430386434303066356534303266663636343130333733366639323438313463643430636263

3730633765313661646163663935653265353537636338620a643936663166643062313761636663

39326334636266623331613435356664626633366330623763646231393830333037636534633339

6664383135353766300a623934336461663864343238623931663038613430326366636339306231

6336

1. **Prepare Your YAML Inventory File:**

Convert your inventory file from INI format to YAML format and replace the plain text passwords with the encrypted strings.

**Step 3: Converting Inventory File to YAML Format**

1. **Original INI Inventory File:**

ini

Copy code

[dev01]

r1 ansible\_host=devnetsandboxiosxe.cisco.com ansible\_network\_os=ios ansible\_connection=network\_cli

[dev02]

r2 ansible\_host=15.228.223.216 ansible\_ssh\_private\_key\_file="/root/cisco.pem" ansible\_user=ec2-user ansible\_network\_os=ios ansible\_connection=network\_cli

[dev01:vars]

ansible\_user=testuser

ansible\_password=testpassword

[demo]

s1 ansible\_host=172.31.16.54

s2 ansible\_host=172.31.19.132

[demo:vars]

ansible\_user=root

ansible\_password=raman

1. **Converted YAML Inventory File:**

yaml

Copy code

all:

children:

dev01:

hosts:

r1:

ansible\_host: devnetsandboxiosxe.cisco.com

ansible\_network\_os: ios

ansible\_connection: network\_cli

vars:

ansible\_user: testuser

ansible\_password: !vault |

$ANSIBLE\_VAULT;1.2;AES256;r1\_userid

33396430386434303066356534303266663636343130333733366639323438313463643430636263

3730633765313661646163663935653265353537636338620a643936663166643062313761636663

39326334636266623331613435356664626633366330623763646231393830333037636534633339

6664383135353766300a623934336461663864343238623931663038613430326366636339306231

6336

dev02:

hosts:

r2:

ansible\_host: 15.228.223.216

ansible\_ssh\_private\_key\_file: "/root/cisco.pem"

ansible\_user: ec2-user

ansible\_network\_os: ios

ansible\_connection: network\_cli

demo:

hosts:

s1:

ansible\_host: 172.31.16.54

s2:

ansible\_host: 172.31.19.132

vars:

ansible\_user: root

ansible\_password: raman

**Step 4: Running the Playbook with the Encrypted Inventory**

1. **Run the Playbook:**

Now, run your playbook using the encrypted inventory file and provide the vault ID:

bash

Copy code

ansible-playbook -i hosts fifth.yml --vault-id r1\_userid@prompt

This command will prompt you for the password associated with r1\_userid to decrypt the encrypted data and execute the playbook.

**Conclusion:**

In this lab, you learned how to:

* Encrypt sensitive strings using Ansible Vault.
* Convert an inventory file from INI format to YAML format.
* Replace plain text passwords with encrypted strings in a YAML inventory.
* Run an Ansible playbook using an inventory with encrypted data.

This process ensures that sensitive information such as passwords is securely managed within your Ansible workflows.

**Lab Guide: Encrypting Strings with Ansible Vault Without Replacing the Whole String in Inventory File**

**Objective:**

Learn how to securely manage sensitive information in Ansible playbooks by encrypting strings using Ansible Vault and referencing them in your inventory files.

**Step 1: Encrypt a String with Ansible Vault**

1. **Encrypt the Password and Save It to a File:**

Use the ansible-vault encrypt\_string command to encrypt a password and save it to a separate YAML file (vault\_secrets.yml):

bash

Copy code

ansible-vault encrypt\_string --vault-id r1\_userid@prompt 'testpassword' --name 'enc\_password' > vault\_secrets.yml

The content of the vault\_secrets.yml file will look something like this:

yaml

Copy code

enc\_password: !vault |

$ANSIBLE\_VAULT;1.2;AES256;r1\_userid

62366163636165343066363733616362616335336565383538313066333935613435613332323434

3833653439626531323830633038333261386439343866610a376532373465383634376637623763

34373061646161643531643535666533353665363963626635623830643961323337646134363333

3261306331646130650a336531343462393064343330646533383438326631643032386138656135

6338

**Step 2: Modify the Inventory File**

1. **YAML Inventory File:**

Modify the inventory file to reference the encrypted password from vault\_secrets.yml instead of storing the password directly:

yaml

Copy code

all:

children:

dev01:

hosts:

r1:

ansible\_host: devnetsandboxiosxe.cisco.com

ansible\_network\_os: ios

ansible\_connection: network\_cli

vars:

ansible\_user: testuser

ansible\_password: "{{ enc\_password }}"

dev02:

hosts:

r2:

ansible\_host: 15.228.223.216

ansible\_ssh\_private\_key\_file: "/root/cisco.pem"

ansible\_user: ec2-user

ansible\_network\_os: ios

ansible\_connection: network\_cli

demo:

hosts:

s1:

ansible\_host: 172.31.16.54

s2:

ansible\_host: 172.31.19.132

vars:

ansible\_user: root

ansible\_password: raman

**Step 3: Create the Playbook**

1. **Playbook to Display Configuration:**

Create a playbook named fifth.yml that uses the encrypted password to retrieve and display configuration details from a Cisco device:

yaml

Copy code

- name: Network Getting Started First Playbook Extended

connection: network\_cli

gather\_facts: false

hosts: dev01

vars\_files:

- vault\_secrets.yml

tasks:

- name: Get config for IOS devices

cisco.ios.ios\_facts:

gather\_subset: all

- name: Display the config

debug:

msg: "The hostname is {{ ansible\_net\_hostname }} and the OS is {{ ansible\_net\_version }}"

**Step 4: Run the Playbook**

1. **Execute the Playbook:**

Run the playbook using the inventory file and the encrypted password:

bash

Copy code

ansible-playbook -i hosts fifth.yml --vault-id r1\_userid@prompt

When prompted, enter the vault password (r1\_userid). The playbook will then use the encrypted password stored in vault\_secrets.yml to authenticate with the Cisco device.

**Step 5: Review the Output**

1. **Playbook Output:**

After running the playbook, you should see output similar to the following, indicating that the playbook successfully retrieved and displayed configuration details:

plaintext

Copy code

PLAY [Network Getting Started First Playbook Extended] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Get config for IOS devices] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[WARNING]: ansible-pylibssh not installed, falling back to paramiko

ok: [r1]

TASK [Display the config] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [r1] => {

"msg": "The hostname is Cat8000V and the OS is 17.12.02"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

r1 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

**Conclusion:**

In this lab, you learned how to:

* Encrypt a sensitive string using Ansible Vault without needing to replace the entire string in the inventory file.
* Reference the encrypted string in the inventory file.
* Create and execute a playbook that uses the encrypted password to interact with network devices.

This approach allows you to manage sensitive data securely and efficiently within your Ansible projects.

**Lab Guide: Asynchronous IOS Image Upgrade and Device Management Using Ansible**

**Objective:**

This lab guide will help you perform an asynchronous IOS image upgrade on a Cisco device using Ansible. Additionally, you will learn to manage a Cisco IOS device by applying configurations, backing up the current state, and verifying the new IOS image.

**Part 1: Asynchronous IOS Image Upgrade**

**Step 1: Prepare the Playbook for Asynchronous Upgrade**

1. **Create the Playbook:** Create a new Ansible playbook named async\_upgrade.yml with the following content:

yaml

Copy code

---

- name: Asynchronously Upgrade Cisco IOS Image

hosts: cisco\_ios

gather\_facts: no

tasks:

- name: Start IOS image upgrade

ios\_command:

commands:

- copy tftp://192.168.1.100/ios\_image.bin flash:

- reload

async: 1800 # Timeout in seconds (30 minutes)

poll: 15 # Check task status every 15 seconds

register: async\_result

- name: Check the status of the asynchronous upgrade

async\_status:

jid: "{{ async\_result.ansible\_job\_id }}"

register: job\_result

until: job\_result.finished

retries: 30

delay: 30 # Check every 30 seconds

- name: Verify the new IOS image is present

ios\_command:

commands:

- dir flash:

register: dir\_output

- name: Display the IOS image directory

debug:

var: dir\_output.stdout\_lines

**Step 2: Run the Asynchronous Playbook**

1. **Execute the Playbook:** Run the playbook to initiate the IOS upgrade:

bash

Copy code

ansible-playbook -i hosts async\_upgrade.yml

1. **Monitor the Progress:**
   * The playbook will start the upgrade and reload the device asynchronously.
   * It will poll the status every 15 seconds and continue checking until the upgrade process finishes.
   * Once complete, it will verify the presence of the new IOS image in the device's flash memory.

**Part 2: Managing Cisco IOS Device Configuration**

**Step 3: Prepare the Device Management Playbook**

1. **Create the Playbook:** Create another playbook named 13th.yml for managing the Cisco IOS device:

yaml

Copy code

---

- name: Manage Cisco IOS device

hosts: devices

gather\_facts: no

connection: network\_cli

tasks:

- name: Ensure the device is reachable

ping:

tags:

- connectivity

- name: Backup the current configuration

ios\_config:

backup: yes

backup\_options:

filename: backup.cfg

tags:

- backup

- name: Configure hostname

ios\_config:

lines:

- hostname Router01

tags:

- config

- hostname

- name: Apply interface configuration

ios\_config:

lines:

- description Connected to LAN

parents: interface GigabitEthernet1

tags:

- config

- interface

- name: Save running configuration to startup

ios\_command:

commands:

- write memory

tags:

- save

**Step 4: Run the Device Management Playbook**

1. **Execute the Playbook:** Run the playbook to manage the device:

bash

Copy code

ansible-playbook -i hosts 13th.yml

1. **Understand the Playbook Tasks:**
   * **Ensure Connectivity:** This task pings the device to ensure it is reachable.
   * **Backup Configuration:** This task backs up the current configuration of the device to a file named backup.cfg.
   * **Configure Hostname:** This task changes the device's hostname to Router01.
   * **Apply Interface Configuration:** This task configures an interface with a description.
   * **Save Configuration:** This task saves the running configuration to startup-config.

**Conclusion:**

In this lab, you:

* Performed an asynchronous IOS image upgrade on a Cisco device using Ansible.
* Managed various configurations on a Cisco IOS device, including hostname changes, interface configuration, and backing up device settings.

By the end of this lab, you should be comfortable with handling both asynchronous actions and typical device management tasks using Ansible in a network automation context.

**Lab Guide: Creating, Uploading, and Testing a Custom Ansible Role**

**Objective:**

This guide will walk you through the steps to create, upload, and test a custom Ansible role, including setting up a GitHub repository, uploading the role to Ansible Galaxy, and downloading and testing the role.

**Part 1: Creating and Uploading a Custom Role**

**Step 1: Create the Custom Role**

1. **Prepare the Custom Role:**
   * If you already have a custom role, ensure it is organized according to the standard Ansible role structure.
2. **Move the Role to a New GitHub Repository:**
   * Create a new GitHub repository. For example, let's call it lumen\_role\_26-30th.
   * Clone the repository to your local machine:

bash

Copy code

git clone https://github.com/ramannkhanna2/lumen\_role\_26-30th.git

* + Move your custom role into the cloned directory:

bash

Copy code

mv raman\_role/ lumen\_role\_26-30th/

1. **Commit and Push the Role to GitHub:**
   * Navigate to the directory:

bash

Copy code

cd lumen\_role\_26-30th/

* + Add the files to the repository:

bash

Copy code

git add .

* + Commit the changes:

bash

Copy code

git commit -m "files added by raman"

* + Set your Git configuration if not already set:

bash

Copy code

git config --global user.email "ramanmail.com"

git config --global user.name "raman"

* + Push the changes to the GitHub repository:

bash

Copy code

git push origin main

**Step 2: Upload the Role to Ansible Galaxy**

1. **Import the Role to Ansible Galaxy:**
   * Go to Ansible Galaxy.
   * Navigate to **My Content** and choose **Import Role**.
   * Select the GitHub repository you created, and import the role.

**Part 2: Downloading and Testing the Role**

**Step 3: Download a Role from Ansible Galaxy**

1. **Search and Install a Role:**
   * Search for a role on Ansible Galaxy (e.g., cisco):

bash

Copy code

ansible-galaxy search cisco

* + Install a role from Ansible Galaxy (e.g., geerlingguy.apache):

bash

Copy code

ansible-galaxy role install geerlingguy.apache

* + The role will be downloaded to the default roles directory, typically under ~/.ansible/roles/.

1. **Verify the Installation:**
   * Navigate to the roles directory:

bash

Copy code

cd ~/.ansible/roles/geerlingguy.apache/

* + Check the contents, including defaults, tasks, and other directories.

**Part 3: Testing the Role**

**Step 4: Create a Playbook to Test the Role**

1. **Prepare an Inventory File:**
   * Create an inventory file (e.g., inv) with the following content:

ini

Copy code

[demo]

s1 ansible\_host=172.31.16.54

1. **Create a Playbook:**
   * Create a playbook named playbook.yml to use the downloaded role:

yaml

Copy code

---

- hosts: demo

roles:

- geerlingguy.apache

1. **Run the Playbook:**
   * Execute the playbook to apply the role to the specified hosts:

bash

Copy code

ansible-playbook -i inv playbook.yml

* + This playbook will apply the configurations defined in the geerlingguy.apache role.

**Part 4: Additional Testing**

**Step 5: Modify and Reapply the Role**

1. **Modify Role Variables:**
   * Navigate to the role’s default variables file:

bash

Copy code

cd ~/.ansible/roles/geerlingguy.apache/defaults/

vi main.yml

* + Modify any variables as necessary.

1. **Reapply the Role:**
   * Run the playbook again to apply the modified role:

bash

Copy code

ansible-playbook -i inv playbook.yml

1. **Clean Up:**
   * Remove any unnecessary packages (e.g., remove Apache from demo servers):

bash

Copy code

ansible demo -a "apt remove apache2" -i inv

**Conclusion:**

In this lab, you:

* Created a custom Ansible role and uploaded it to a GitHub repository.
* Imported the role to Ansible Galaxy.
* Downloaded and tested an existing role from Ansible Galaxy on a target system.

By completing this lab, you should have a good understanding of how to manage, distribute, and test Ansible roles using both GitHub and Ansible Galaxy.

4o