**Lab Guide: Configuring Cisco Device Interfaces with Ansible**

**Objective:**

Learn how to create and execute an Ansible playbook that checks the status of a Cisco device's interface and configures it based on the status of the line protocol.

**Prerequisites:**

* Basic understanding of Ansible.
* Ansible installed on your control node (e.g., your local machine).
* Cisco device(s) accessible from your control node.
* SSH access to the Cisco device(s).
* Python packages paramiko or netmiko and ansible are installed.
* Cisco devices are configured to allow Ansible to interact via SSH.

**Lab Setup:**

1. **Create Inventory File:**

You need an inventory file to define the Cisco device(s) that Ansible will manage.

Create a file named inv and add the following content:

ini

Copy code

[devices]

your\_cisco\_device\_ip ansible\_user=your\_username ansible\_password=your\_password ansible\_network\_os=ios ansible\_connection=network\_cli

Replace your\_cisco\_device\_ip, your\_username, and your\_password with your device's actual IP, username, and password.

1. **Create the Ansible Playbook:**

Create a file named sixth.yml with the following content:

yaml

Copy code

---

- name: Configure Cisco Device

hosts: devices

gather\_facts: no

tasks:

- name: Gather the facts about interfaces

ios\_facts:

gather\_subset: interfaces

register: raman\_ios\_facts

- name: Print ios\_facts for debugging

debug:

var: raman\_ios\_facts

- name: Configure interface1 if line protocol is down

ios\_config:

lines:

- description Configured by Ansible

- no shutdown

ignore\_errors: true

when: >

raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces is defined and

raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces['GigabitEthernet1'] is defined and

raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces['GigabitEthernet1'].lineprotocol == 'down'

- name: Print line protocol status for debugging

debug:

msg: "Line protocol: {{ raman\_ios\_facts.ansible\_facts.ansible\_net\_interfaces['GigabitEthernet1'].lineprotocol }} "

1. **Verify Ansible Setup:**

Before running the playbook, verify that your Ansible environment is set up correctly and can communicate with your Cisco device.

bash

Copy code

ansible -m ping -i inv

If the setup is correct, you should see a "pong" response from the device.

1. **Run the Playbook in Check Mode:**

Run the playbook in check mode to see what changes would be made without actually applying them.

bash

Copy code

ansible-playbook sixth.yml --check

Review the output to ensure the playbook will work as expected.

1. **Execute the Playbook:**

After verifying the playbook in check mode, run it to apply the changes to the device.

bash

Copy code

ansible-playbook sixth.yml

1. **Debugging with Verbose Output:**

If you encounter any issues or want more detailed output, you can increase the verbosity level.

bash

Copy code

ansible-playbook sixth.yml -vvv

1. **Review the Results:**
   * Check the output for the status of the GigabitEthernet1 interface.
   * Verify on the Cisco device whether the interface configuration was applied as expected.
2. **Cleanup:**
   * Remove any temporary files created during the lab.
   * Optionally, revert any changes made to the Cisco device's configuration.

**Conclusion:**

By completing this lab, you have learned how to create and execute an Ansible playbook that checks the status of a Cisco device's interface and configures it accordingly. This is a practical example of using conditional statements in Ansible to make decisions based on device status.

**Lab Guide: Configuring Cisco IOS Interfaces and Managing Users with Ansible**

**Objective:**

Learn how to use Ansible to configure multiple interfaces and manage user accounts on Cisco IOS devices by utilizing loops and conditional statements in playbooks.

**Prerequisites:**

* Basic understanding of Ansible and Cisco IOS.
* Ansible installed on your control node.
* Cisco devices accessible via SSH.
* Python packages paramiko or netmiko and ansible installed.
* Cisco devices are configured to allow Ansible to interact via SSH.

**Part 1: Configuring Multiple Interfaces**

**1. Create an Inventory File**

Create a file named inv with the following content:

ini

Copy code

[devices]

your\_cisco\_device\_ip ansible\_user=your\_username ansible\_password=your\_password ansible\_network\_os=ios ansible\_connection=network\_cli

Replace your\_cisco\_device\_ip, your\_username, and your\_password with your device's actual IP, username, and password.

**2. Create the Ansible Playbook for Interface Configuration**

Create a file named interface\_config.yml with the following content:

yaml

Copy code

---

- name: Configure multiple interfaces on Cisco IOS devices

hosts: devices

gather\_facts: no

connection: network\_cli

vars:

interfaces:

- name: GigabitEthernet1

description: Configured by RamanKhanna

- name: GigabitEthernet2

description: Configured by RamanKhanna

- name: GigabitEthernet3

description: Configured by Ansible

- name: GigabitEthernet4

description: Configured by Ansible

tasks:

- name: Gather facts about interfaces

cisco.ios.ios\_facts:

gather\_subset: interfaces

register: ios\_facts

- name: Print ios\_facts for debugging

debug:

var: ios\_facts

- name: Configure interfaces

cisco.ios.ios\_config:

lines:

- description {{ item.description }}

parents: "interface {{ item.name }}"

loop: "{{ interfaces }}"

when: >

ios\_facts.ansible\_facts.ansible\_net\_interfaces is defined and

ios\_facts.ansible\_facts.ansible\_net\_interfaces[item.name] is defined and

ios\_facts.ansible\_facts.ansible\_net\_interfaces[item.name].operstatus == 'up'

**3. Verify Ansible Connectivity**

Ensure that Ansible can connect to your Cisco device by running:

bash

Copy code

ansible -m ping -i inv

**4. Run the Playbook in Check Mode**

Test the playbook to ensure it will perform as expected:

bash

Copy code

ansible-playbook interface\_config.yml --check

**5. Execute the Playbook**

Run the playbook to configure the interfaces:

bash

Copy code

ansible-playbook interface\_config.yml

**6. Verify Configuration**

After execution, check the interface configurations on your Cisco device.

**Part 2: Managing Users on Cisco IOS Devices**

**1. Create the Ansible Playbook for User Management**

Create a file named user\_management.yml with the following content:

yaml

Copy code

---

- name: Create multiple users on Cisco IOS devices

hosts: devices

gather\_facts: no

connection: network\_cli

vars:

users:

- name: admin1

password: admin1password

privilege: 15

- name: admin2

password: admin2password

privilege: 15

- name: operator

password: operatorpassword

privilege: 5

tasks:

- name: Create users

cisco.ios.ios\_user:

name: "{{ item.name }}"

configured\_password: "{{ item.password }}"

privilege: "{{ item.privilege }}"

state: present

update\_password: always

loop: "{{ users }}"

**2. Run the Playbook in Check Mode**

Test the playbook to verify what changes would be made:

bash

Copy code

ansible-playbook user\_management.yml --check

**3. Execute the Playbook**

Run the playbook to create the users:

bash

Copy code

ansible-playbook user\_management.yml

**4. Verify User Creation**

Check the Cisco device to ensure the users were created with the correct privileges.

**Conclusion:**

By following this lab guide, you've learned how to configure multiple interfaces and manage user accounts on Cisco IOS devices using Ansible playbooks with loops and conditionals. This approach enhances automation, making network management more efficient and error-free.

**Lab Guide: Using Handlers in Ansible to Manage Cisco IOS Devices and Apache Services**

**Objective:**

Learn how to use handlers in Ansible to trigger specific actions based on changes made during playbook execution. This guide covers two scenarios: configuring Cisco IOS interfaces and managing an Apache service on a Linux server.

**Prerequisites:**

* Basic understanding of Ansible.
* Ansible installed on your control node.
* Cisco device(s) and Linux server(s) accessible via SSH.
* Python packages paramiko or netmiko and ansible installed.
* Cisco devices and Linux servers are configured to allow Ansible to interact via SSH.

**Part 1: Configuring Cisco IOS Devices and Using Handlers**

**1. Create an Inventory File**

Create a file named inv with the following content:

ini

Copy code

[devices]

your\_cisco\_device\_ip ansible\_user=your\_username ansible\_password=your\_password ansible\_network\_os=ios ansible\_connection=network\_cli

[demo]

your\_linux\_server\_ip ansible\_user=your\_username ansible\_password=your\_password ansible\_connection=ssh

Replace your\_cisco\_device\_ip, your\_linux\_server\_ip, your\_username, and your\_password with your actual IPs, username, and password.

**2. Create the Cisco IOS Configuration Playbook**

Create a file named 11th.yml with the following content:

yaml

Copy code

---

- name: Configure Cisco IOS Devices

hosts: devices

gather\_facts: no

connection: network\_cli

tasks:

- name: Check if GigabitEthernet1 is present

cisco.ios.ios\_command:

commands: show ip interface brief

register: interface\_status

- name: Output interface status

debug:

var: interface\_status

- name: Configure interface GigabitEthernet1

cisco.ios.ios\_interfaces:

config:

- name: GigabitEthernet1

description: "Configured by RamanK"

enabled: yes

state: merged

when: "'GigabitEthernet1' in interface\_status.stdout[0]"

notify: Restart network

- name: Configure interface GigabitEthernet2 if present

cisco.ios.ios\_interfaces:

config:

- name: GigabitEthernet2

description: "Configured by Ansible"

enabled: yes

state: merged

when: "'GigabitEthernet2' in interface\_status.stdout[0]"

notify: Restart network

- name: Get interface descriptions

cisco.ios.ios\_command:

commands:

- show interfaces description

register: interface\_descriptions

- name: Display interface descriptions

debug:

msg: "{{ interface\_descriptions.stdout[0] }}"

handlers:

- name: Restart network

cisco.ios.ios\_command:

commands:

- "write memory"

- "reload"

timeout: 300 # Increase timeout to 300 seconds (5 minutes)

**3. Run the Cisco IOS Playbook**

Run the playbook to configure the interfaces and trigger the handler:

bash

Copy code

ansible-playbook 11th.yml -i inv

After running, the playbook will configure the interfaces if they are present and, if changes are made, it will restart the network on the Cisco device.

**Part 2: Managing Apache Services on a Linux Server**

**1. Create the Apache Configuration Playbook**

Create a file named 12th.yml with the following content:

yaml

Copy code

---

- hosts: demo

gather\_facts: true

vars:

package: apache2

tasks:

- name: Update apt package index

apt:

update\_cache: yes

- name: Install apache2 package

package:

name: "{{ package }}"

state: present

- name: Configure hostname and IP in webpage

template:

src: index.html.j2

dest: /var/www/html/index.html

notify: Restart apache2 service

handlers:

- name: Restart apache2 service

service:

name: apache2

state: restarted

**2. Create the Jinja2 Template File**

Create a template file named index.html.j2 in the same directory as the playbook:

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Server Information</title>

</head>

<body>

<h1>Server Information</h1>

<p>Hostname: {{ ansible\_facts['ansible\_nodename'] | default('Unknown') }}</p>

<p>IP Address: {{ ansible\_facts['default\_ipv4']['address'] | default('Unknown') }}</p>

</body>

</html>

**3. Run the Apache Playbook**

Run the playbook to install Apache, configure the webpage, and trigger the handler if the configuration changes:

bash

Copy code

ansible-playbook 12th.yml -i inv

After running, the playbook will install Apache, update the webpage with the server's hostname and IP address, and restart Apache if the webpage configuration changes.

**Conclusion:**

In this lab, you have learned how to use handlers in Ansible to trigger specific actions based on changes in Cisco IOS device configurations and Apache services on Linux servers. Handlers are a powerful feature in Ansible, allowing for efficient automation and management of network and server infrastructure.

**Lab Guide: Using Ansible Roles to Manage Cisco IOS Devices**

**Objective:**

Learn how to create and use Ansible roles to manage Cisco IOS devices. This guide will walk you through creating a role, using variable substitution, implementing handlers, and executing the playbook with command-line variables.

**Prerequisites:**

* Basic understanding of Ansible.
* Ansible installed on your control node.
* Cisco device(s) accessible via SSH.
* Python packages paramiko or netmiko and ansible installed.
* Cisco devices are configured to allow Ansible to interact via SSH.

**Step 1: Create the Ansible Role**

**1.1. Create the Role Structure**

Use the following command to create a new Ansible role named ios\_config:

bash

Copy code

ansible-galaxy init ios\_config

This command generates the directory structure for the role:

plaintext

Copy code

ios\_config/

├── README.md

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── tasks

│ └── main.yml

├── templates

├── tests

│ ├── inventory

│ └── test.yml

└── vars

└── main.yml

**Step 2: Define Role Tasks**

**2.1. Edit the Role Tasks File**

Edit the tasks/main.yml file to define tasks for configuring Cisco interfaces:

yaml

Copy code

---

# tasks file for ios\_config

- name: "Check if {{ interface1 }} is present"

cisco.ios.ios\_command:

commands: show ip interface brief

register: interface\_status

- name: "Configure interface {{ interface1 }}"

cisco.ios.ios\_interfaces:

config:

- name: "{{ interface1 }}"

description: "Configured by {{ ansible\_user }} "

enabled: yes

state: merged

when: "'GigabitEthernet1' in interface\_status.stdout[0]"

notify: Restart network

- name: "Configure interface {{ interface2 }} if present"

cisco.ios.ios\_interfaces:

config:

- name: "{{ interface2 }}"

description: "Configured by Ansible"

enabled: yes

state: merged

when: "'GigabitEthernet2' in interface\_status.stdout[0]"

notify: Restart network

- name: Get interface descriptions

cisco.ios.ios\_command:

commands:

- show interfaces description

register: interface\_descriptions

- name: Display interface descriptions

debug:

msg: "{{ interface\_descriptions.stdout[0] }}"

**Step 3: Configure Pre-Tasks and Post-Tasks**

**3.1. Create the Playbook**

Create a playbook playbook.yml that uses the role ios\_config and includes pre-tasks and post-tasks:

yaml

Copy code

---

- hosts: ios\_devices

gather\_facts: no

pre\_tasks:

- name: Pre-task example

debug:

msg: "Starting configuration of Cisco devices"

roles:

- ios\_config

post\_tasks:

- name: Post-task example

debug:

msg: "Configuration complete"

**Step 4: Define Role Variables**

**4.1. Edit Role Variables**

Edit the vars/main.yml file to define role-specific variables:

yaml

Copy code

# vars file for ios\_config

---

interface1: GigabitEthernet1

interface2: GigabitEthernet2

Edit the defaults/main.yml file to define default variables:

yaml

Copy code

# defaults file for ios\_config

---

ansible\_user: "default\_user"

**Step 5: Implement Handlers**

**5.1. Edit the Handlers File**

Edit the handlers/main.yml file to define a handler that restarts the network:

yaml

Copy code

---

# handlers file for ios\_config

- name: Restart network

cisco.ios.ios\_command:

commands:

- "write memory"

- "reload"

timeout: 300 # Increase timeout to 300 seconds (5 minutes)

**Step 6: Passing Variables from the Command Line**

**6.1. Running the Playbook with Variables**

You can pass variables from the command line when running the playbook:

bash

Copy code

ansible-playbook -i inventory playbook.yml -e "interface1=GigabitEthernet3 ansible\_user=admin\_user"

This command overrides the interface1 and ansible\_user variables.

**Step 7: Execute an Ansible Role in a Different Playbook**

**7.1. Create a Secondary Playbook**

Create a playbook sec.yml to include and execute the ios\_config role:

yaml

Copy code

---

- name: starting

hosts: devices

gather\_facts: false

tasks:

- name: get config for ios dev

ios\_facts:

gather\_subset: interfaces

- name: display the conf

debug:

msg: "all ipv4 addresses: {{ ansible\_net\_all\_ipv4\_addresses }}"

- name: Execute ios\_config role

include\_role:

name: ios\_config

ignore\_errors: true

register: role\_result

- name: output of roles execution

debug:

var: role\_result

You can run this playbook with:

bash

Copy code

ansible-playbook -i inventory sec.yml

**Conclusion:**

In this lab, you have learned how to create and use Ansible roles to manage Cisco IOS devices. You have also learned how to pass variables from the command line, include roles in different playbooks, and utilize handlers to manage device configurations effectively.