**Lab Guide for Ansible Debug, Error Handling, Conditions, and Task Execution Logic**

**Overview**

This lab demonstrates the following key concepts of Ansible:

1. Using debug for troubleshooting and inspecting variables.
2. Registering outputs from tasks and using these outputs in subsequent tasks.
3. Implementing conditional task execution using when.
4. Handling errors gracefully with ignore\_errors.
5. Combining facts like ansible\_distribution and ansible\_distribution\_version for dynamic conditions.

**Pre-requisites**

1. A control node with Ansible installed (e.g., Ubuntu, CentOS).
2. Target nodes (e.g., gagan-client or a machine group specified in the inventory file).
3. Ensure SSH access is configured between the control and managed nodes.
4. Required packages (talk, ntp) should be available in the repositories of the managed nodes.

**Inventory Setup**

Create an inventory file (inventory) for the target hosts:

ini

Copy code

[gagan-client]

<target-machine-ip>

Replace <target-machine-ip> with the IP address of your managed node.

**Lab Steps**

**1. Ansible Playbook - Fourth Playbook**

**Playbook Content:**

yaml

Copy code

---

- name: This is my First Debug Play

hosts: all

tasks:

- name: Testing Ansible Facts {{ ansible\_hostname }}

debug:

msg: "Host {{ ansible\_hostname }} is having IP address {{ ansible\_all\_ipv4\_addresses }}"

- debug:

msg: First task to Print debug message

- debug:

msg: "System {{ inventory\_hostname }} has uuid {{ ansible\_product\_uuid }}"

- name: Talk package uninstallation

apt:

name: talk

state: absent

register: outp

- name: Running debugger to show the output

debug:

var: outp

- name: Task 1 - Run command "/dev/null"

command: /dev/null

register: task1\_result

ignore\_errors: yes

- name: Running debugger to show the output

debug:

var: task1\_result

- name: install ntp

apt:

name: ntp

state: present

register: ntp\_out

- name: printing complete output

debug:

var: ntp\_out.changed

- name: Second last task - Run only if Task 1 fails

debug:

msg: "This task runs only if Task 1 fails"

when: task1\_result.rc != 0

- name: Running debugger to show the output

debug:

var: task1\_result.rc

- name: Final task - Run only if Task 1 succeeds

debug:

msg: "This task runs only if Task 1 succeeds"

when: task1\_result.rc == 0

**Step-by-Step Explanation**

1. **Testing Ansible Facts:**
   * Use the debug module to inspect facts such as ansible\_hostname and ansible\_all\_ipv4\_addresses.
2. **Task Registration:**
   * Register the output of tasks like uninstallation of talk or the installation of ntp.
3. **Error Handling:**
   * Use ignore\_errors: yes to allow the playbook to continue execution even if a task fails.
4. **Conditional Execution:**
   * Use the when clause to run tasks based on the return code (task1\_result.rc) of a command.

**2. Ansible Playbook - First Playbook**

**Playbook Content:**

yaml

Copy code

---

- name: play1 for creating user on gagan-client machine

hosts: gagan-client

tasks:

- name: description for task1 for creating user

user:

name: gagandeep

state: present

when: ansible\_distribution == "CentOS" and ansible\_distribution\_version == "7.9"

register: user\_out

- name: false task

debug:

var: user\_out

ignore\_errors: true

when: user\_out.state == "present"

- name: creating a file on next machine

file:

path: /tmp/gds

state: touch

mode: 1600

when: ansible\_distribution == "Redhat"

**Step-by-Step Explanation**

1. **User Creation:**
   * Creates a user gagandeep only if the OS is CentOS and the version is 7.9.
2. **Debug with Error Handling:**
   * Attempts to print the variable user\_out if the user creation task is successful. Errors are ignored using ignore\_errors: true.
3. **File Creation:**
   * Creates a file /tmp/gds on the managed node, but only if the OS is Red Hat.

**Execution**

1. **Run the Fourth Playbook:**

bash

Copy code

ansible-playbook -i inventory fourth.yml

1. **Run the First Playbook:**

bash

Copy code

ansible-playbook -i inventory first.yaml

**Verification and Expected Output**

**Fourth Playbook**

1. Debug messages should display hostnames, IP addresses, and system UUIDs.
2. Output of tasks like talk uninstallation and ntp installation should be printed.
3. Conditional tasks will execute based on the success or failure of task1\_result.

**First Playbook**

1. The user gagandeep will be created if the conditions are met.
2. If successful, the debug output will show details of the created user.
3. A file /tmp/gds will be created on Red Hat systems.

**Additional Notes**

* Modify the inventory or playbooks to match your environment.
* Add verbosity (-vvv) when running playbooks for detailed output.
* Use ansible-doc to understand module parameters.

This concludes the detailed lab for the provided playbooks.

4o

Top of Form

Bottom of Form

**Lab Guide: Ansible Vault and Encrypted Variable Management**

This guide provides a step-by-step walkthrough to demonstrate the usage of **Ansible Vault**, its integration with playbooks, and encrypted variables for securely managing sensitive data.

**Pre-requisites**

1. **Ansible** installed on the control node.
2. A basic understanding of Ansible playbooks and inventory files.
3. At least two managed nodes with SSH connectivity.
4. A working inventory file and playbook.
5. Sudo privileges on all nodes.

**Scenario Overview**

You will:

1. Create an inventory file with encrypted passwords using Ansible Vault.
2. Write a playbook to use the encrypted variables securely.
3. Manage encryption and decryption operations.
4. Run the playbook with the encrypted inventory.

**Steps**

**1. Create an Inventory File**

Create an inventory file named inv with the following content:

ini

Copy code

[demo1]

m1 ansible\_host=172.31.25.29 ansible\_user="raman" ansible\_password="{{ encrypted\_m1\_password }}"

[demo2]

m2 ansible\_host=172.31.24.46 ansible\_user="raman2" ansible\_password="{{ encrypted\_m2\_password }}"

**2. Encrypt Sensitive Data**

**Encrypt String Values**

Use ansible-vault encrypt\_string to encrypt the passwords:

bash

Copy code

ansible-vault encrypt\_string 'raman' --name 'encrypted\_m1\_password' >> vault\_secrets.yaml

ansible-vault encrypt\_string 'khanna' --name 'encrypted\_m2\_password' >> vault\_secrets.yaml

The vault\_secrets.yaml file should look like this:

yaml

Copy code

encrypted\_m1\_password: !vault |

$ANSIBLE\_VAULT;1.1;AES256

<encrypted-string-data>

encrypted\_m2\_password: !vault |

$ANSIBLE\_VAULT;1.1;AES256

<encrypted-string-data>

**3. Verify Encrypted Variables**

View the encrypted file using:

bash

Copy code

ansible-vault view vault\_secrets.yaml

**4. Create the Playbook**

Write a playbook fourth.yml to perform the following tasks:

1. Use encrypted variables from vault\_secrets.yaml.
2. Run various tasks like package installation, debugging, and conditional execution.

**Playbook Content**

yaml

Copy code

---

- name: Debugging and Task Execution with Vault

hosts: all

vars\_files:

- vault\_secrets.yaml

tasks:

- name: Display Host Details

debug:

msg: "Host {{ ansible\_hostname }} has IP {{ ansible\_all\_ipv4\_addresses }}"

- name: Show system UUID

debug:

msg: "System {{ inventory\_hostname }} UUID: {{ ansible\_product\_uuid }}"

- name: Uninstall 'talk' package

apt:

name: talk

state: absent

register: outp

- name: Show output of 'talk' package uninstallation

debug:

var: outp

- name: Execute dummy command

command: /dev/null

register: task1\_result

ignore\_errors: yes

- name: Debug command result

debug:

var: task1\_result

- name: Install 'ntp' package

apt:

name: ntp

state: present

register: ntp\_out

- name: Display NTP installation status

debug:

var: ntp\_out.changed

- name: Run if Task 1 fails

debug:

msg: "Task 1 failed, hence this task is running."

when: task1\_result.rc != 0

- name: Run if Task 1 succeeds

debug:

msg: "Task 1 succeeded, hence this task is running."

when: task1\_result.rc == 0

- name: Show encrypted m1 password

debug:

var: encrypted\_m1\_password

- name: Show encrypted m2 password

debug:

var: encrypted\_m2\_password

**5. Encrypt the Playbook**

Encrypt the playbook fourth.yml:

bash

Copy code

ansible-vault encrypt fourth.yml

**6. Run the Playbook**

**Run with Interactive Vault Password**

Execute the playbook and provide the vault password interactively:

bash

Copy code

ansible-playbook -i inv fourth.yml --ask-vault-pass

**Run with Vault Password File**

Save the password in a file and restrict its permissions:

bash

Copy code

echo 'lumen' > .vault\_password.txt

chmod 600 .vault\_password.txt

ansible-playbook -i inv fourth.yml --vault-password-file .vault\_password.txt

**7. Decrypt Playbook**

To view or edit the playbook, decrypt it:

bash

Copy code

ansible-vault decrypt fourth.yml

**8. Test the Inventory**

Test connectivity with encrypted variables:

bash

Copy code

ansible all -m ping -i inv --ask-vault-pass --extra-vars "@vault\_secrets.yaml"

**Key Commands Summary**

1. **Encrypt a Playbook**:

bash

Copy code

ansible-vault encrypt fourth.yml

1. **Decrypt a Playbook**:

bash

Copy code

ansible-vault decrypt fourth.yml

1. **Encrypt a String**:

bash

Copy code

ansible-vault encrypt\_string 'your\_string' --name 'variable\_name'

1. **View Encrypted Data**:

bash

Copy code

ansible-vault view vault\_secrets.yaml

1. **Run Playbook with Vault**:

bash

Copy code

ansible-playbook -i inv fourth.yml --ask-vault-pass

**Conclusion**

You successfully encrypted sensitive data using Ansible Vault, integrated it into your playbook, and securely executed tasks. This practice is essential for protecting credentials and sensitive information in automation workflows.

4o

**Lab Guide: Ansible Roles and Playbook Execution with Multiple Roles**

This lab guide walks you through creating, managing, and executing Ansible roles while including them in playbooks. Follow the steps outlined below to gain hands-on experience.

**Prerequisites**

1. Ansible installed on the control node.
2. A basic understanding of Ansible playbooks and roles.
3. Inventory file (inv) with managed nodes configured.
4. Connectivity to managed nodes via SSH with necessary credentials.

**Part 1: Create an Ansible Role**

1. **Create a Role**:
   * Run the following command to create a role named myrole:

bash

Copy code

ansible-galaxy init myrole

* + This will create the following directory structure:

css

Copy code

myrole/

├── README.md

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── tasks

│ └── main.yml

├── templates

├── tests

│ ├── inventory

│ └── test.yml

└── vars

└── main.yml

1. **Define Default Variables**:
   * Edit defaults/main.yml to include:

yaml

Copy code

---

username: "defaultuser"

uid: 1001

1. **Define Tasks**:
   * Edit tasks/main.yml to perform user management:

yaml

Copy code

---

- name: Create user

user:

name: "{{ username }}"

uid: "{{ uid }}"

become: true

1. **Handlers**:
   * Define a restart handler in handlers/main.yml:

yaml

Copy code

---

- name: Restart Service

service:

name: ntpd

state: restarted

**Part 2: Create Another Role (ntp-role)**

1. **Initialize the Role**:
   * Create a role named ntp-role:

bash

Copy code

ansible-galaxy init ntp-role

1. **Add Configuration File**:
   * Place the ntp.conf file in the files/ directory.
2. **Define Default Variables**:
   * Edit defaults/main.yml to include:

yaml

Copy code

---

username: "ntpuser"

uid: 1020

1. **Define Tasks**:
   * Edit tasks/main.yml to perform the following:

yaml

Copy code

---

- name: Create user

user:

name: "{{ username }}"

uid: "{{ uid }}"

become: true

- name: Install NTP package

package:

name: ntp

state: present

- name: Copy NTP configuration file

copy:

src: ntp.conf

dest: /etc/ntp.conf

notify: Restart Service

- name: Start NTP service

service:

name: ntpd

state: started

enabled: true

**Part 3: Include Roles in a Playbook**

1. **Basic Role Inclusion**:
   * Create a playbook (pb.yml) that includes myrole and ntp-role:

yaml

Copy code

---

- name: Testing role

hosts: demo

roles:

- myrole

- ntp-role

1. **Conditional Role Execution**:
   * Create another playbook (fourth.yml) to demonstrate conditional role execution:

yaml

Copy code

---

- name: Debug and Include Roles

hosts: all

tasks:

- name: Debug Ansible facts

debug:

msg: "Host {{ ansible\_hostname }} has IP {{ ansible\_all\_ipv4\_addresses }}"

- name: Execute ntp-role

include\_role:

name: ntp-role

tasks\_from: main

register: ntp\_result

ignore\_errors: true

- name: Run myrole if ntp-role succeeds

include\_role:

name: myrole

when: ntp\_result is succeeded

**Part 4: Inventory File Configuration**

1. **Create an Inventory File (inv)**:

ini

Copy code

[demo]

m1 ansible\_host=192.168.1.10 ansible\_user=admin ansible\_password=adminpass

**Part 5: Execute Playbooks**

1. **Run Playbook with Roles**:
   * Execute the basic playbook:

bash

Copy code

ansible-playbook pb.yml -i inv

1. **Run Playbook with Conditions**:
   * Execute the playbook with conditional role execution:

bash

Copy code

ansible-playbook fourth.yml -i inv

**Part 6: Tips and Best Practices**

1. **Use roles\_path**:
   * Define the roles path in ansible.cfg:

ini

Copy code

[defaults]

roles\_path = /etc/ansible/roles

1. **Role Reusability**:
   * Keep roles modular for reusability across multiple playbooks.
2. **Testing**:
   * Use --check mode to test changes without applying them:

bash

Copy code

ansible-playbook pb.yml -i inv --check

This lab guide should provide a comprehensive approach to creating and working with Ansible roles in various scenarios.

**Lab Guide: Upload and Share an Ansible Role via GitHub and Ansible Galaxy**

This guide provides step-by-step instructions to create, upload, and share an Ansible role using GitHub and Ansible Galaxy.

**Part 1: Prepare the Role for Upload**

1. **Create and Verify the Role**:
   * Ensure your Ansible role (ntp-role) is properly structured:

bash

Copy code

tree ntp-role/

Expected structure:

css

Copy code

ntp-role/

├── defaults

│ └── main.yml

├── files

│ └── ntp.conf

├── handlers

│ └── main.yml

├── tasks

│ └── main.yml

├── vars

│ └── main.yml

└── meta

└── main.yml

**Part 2: Upload Role to GitHub**

1. **Create a New Repository**:
   * Go to GitHub and create a new repository with the name lumen\_role\_25-29th.
2. **Clone the Repository Locally**:
   * Clone the newly created repository:

bash

Copy code

git clone https://github.com/ramannkhanna2/lumen\_role\_25-29th.git

* + Navigate into the cloned repository:

bash

Copy code

cd lumen\_role\_25-29th/

1. **Copy Role Files to Repository**:
   * Copy the contents of your role (ntp-role) to the cloned repository:

bash

Copy code

cp -r ../ntp-role/\* .

1. **Initialize Git and Configure User**:
   * Check the current repository status:

bash

Copy code

git status

* + Configure Git user details (replace with your details):

bash

Copy code

git config --global user.name "raman"

git config --global user.email "raman@email"

1. **Commit and Push Changes**:
   * Stage all changes:

bash

Copy code

git add .

* + Commit the changes with a message:

bash

Copy code

git commit -m "Adding role content"

* + Push changes to the remote repository:

bash

Copy code

git push origin main

* + You may be prompted to authenticate with a personal access token (PAT).

**Part 3: Create and Use a GitHub Personal Access Token (PAT)**

1. **Generate a PAT**:
   * Go to GitHub: Settings > Developer Settings > Personal Access Tokens.
   * Generate a new token with the following scopes:
     + repo
     + write:packages
   * Copy the generated token (keep it safe).
2. **Use the Token for Authentication**:
   * When prompted for a password during git push, use the token instead.

**Part 4: Publish Role to Ansible Galaxy**

1. **Log in to Ansible Galaxy**:
   * Go to Ansible Galaxy and log in with your GitHub account.
2. **Import the Role**:
   * Navigate to **My Content > Import Role**.
   * Select the GitHub repository (lumen\_role\_25-29th) and import the role.
3. **Verify the Role on Galaxy**:
   * Search for the role:

bash

Copy code

ansible-galaxy search lumen\_role\_25-29th

**Part 5: Use the Role in Ansible**

1. **Install the Role from Ansible Galaxy**:
   * Install the published role:

bash

Copy code

ansible-galaxy role install ramannkhanna2.lumen\_role\_25-29th

1. **Verify Installation**:
   * Check if the role is installed:

bash

Copy code

ls /etc/ansible/roles/ramannkhanna2.lumen\_role\_25-29th

1. **Use the Role in a Playbook**:
   * Create a playbook (pb.yml) to include the role:

yaml

Copy code

---

- name: Test lumen\_role\_25-29th

hosts: all

roles:

- ramannkhanna2.lumen\_role\_25-29th

1. **Execute the Playbook**:
   * Run the playbook:

bash

Copy code

ansible-playbook pb.yml -i inventory\_file

**Cleanup (Optional)**

1. **Remove Local Role Files**:
   * Delete local directories:

bash

Copy code

rm -rf ntp-role/ lumen\_role\_25-29th/

1. **Uninstall Role from Ansible Galaxy**:
   * Remove the installed role:

bash

Copy code

ansible-galaxy role remove ramannkhanna2.lumen\_role\_25-29th

**Summary**

In this lab, you:

1. Created an Ansible role (ntp-role).
2. Uploaded the role to GitHub.
3. Published the role to Ansible Galaxy.
4. Installed and tested the role from Ansible Galaxy.

This approach ensures that your roles are reusable and easily shareable across teams.

4o