**Lab Guide for Creating and Managing Users, Groups, Files, and Directories with Ansible**

**Overview**

This lab guide demonstrates how to create an Ansible playbook to manage users, groups, files, and directories on a target system. The tasks include creating a user and group, setting up a directory with specific permissions, and creating a file with content and ownership settings.

**Prerequisites**

1. An Ansible control node (e.g., controller) with Ansible installed.
2. A target node or group of nodes (inventory defined as demo in this example).
3. SSH access from the control node to the target node(s) with appropriate permissions.
4. Sudo privileges on the target node(s) to create users, groups, files, and directories.

**Step-by-Step Lab Instructions**

**Step 1: Prepare the Environment**

1. **On the Control Node:**
   * Verify Ansible is installed by running:

bash

Copy code

ansible --version

* + Ensure you have a proper inventory file defining the demo group. For example:

bash

Copy code

cat /etc/ansible/hosts

[demo]

target-node ansible\_host=<target\_ip> ansible\_user=<username>

1. **On the Target Node(s):**
   * Verify SSH access from the control node:

bash

Copy code

ssh <username>@<target\_ip>

* + Ensure sudo is available to the user.

**Step 2: Create the Ansible Playbook**

1. Create a directory for the lab files on the control node:

bash

Copy code

mkdir -p ~/ansible-labs/user-group-management

cd ~/ansible-labs/user-group-management

1. Create the playbook file:

bash

Copy code

nano first.yml

1. Add the following content to first.yml:

yaml

Copy code

- hosts: demo

become: yes # Elevate privileges for tasks requiring root access

tasks:

- name: Group Creation

group:

name: grouptest

gid: 7777

state: present

- name: User Creation

user:

name: usertest

shell: /bin/bash

uid: 6666

comment: "my first user"

state: present

- name: Create a directory

file:

path: /tmp/testdir

state: directory

owner: root

group: root

mode: 0755

- name: Create a file and copy content

copy:

content: |

This is some content of practice.

dest: /tmp/testdir/test

owner: usertest

group: grouptest

mode: 0644

**Step 3: Run the Playbook**

1. Test the connection to the target node:

bash

Copy code

ansible -m ping demo

1. Run the playbook:

bash

Copy code

ansible-playbook first.yml

1. Verify the output:
   * Ensure all tasks complete successfully. You should see a summary with "changed" or "ok" statuses for each task.

**Step 4: Validate the Results**

1. **Check the Group:**
   * Log in to the target node and verify the group grouptest is created:

bash

Copy code

grep grouptest /etc/group

1. **Check the User:**
   * Verify the user usertest is created with the correct UID and shell:

bash

Copy code

id usertest

getent passwd usertest

1. **Check the Directory:**
   * Ensure the directory /tmp/testdir exists with the correct ownership and permissions:

bash

Copy code

ls -ld /tmp/testdir

1. **Check the File:**
   * Verify the file /tmp/testdir/test exists, contains the expected content, and has correct ownership:

bash

Copy code

ls -l /tmp/testdir/test

cat /tmp/testdir/test

**Step 5: Clean Up (Optional)**

If needed, remove the created resources from the target node:

1. Delete the user:

bash

Copy code

sudo userdel usertest

1. Delete the group:

bash

Copy code

sudo groupdel grouptest

1. Delete the directory and file:

bash

Copy code

sudo rm -rf /tmp/testdir

**Additional Notes**

* **Customization:** You can modify the uid, gid, directory path, file content, or permissions as needed for your use case.
* **Scalability:** This playbook can be expanded to manage multiple users, groups, directories, and files.

Enjoy automating your infrastructure with Ansible!

**Lab Guide for Using Ansible Handlers to Configure and Manage NTP Services**

**Overview**

This lab demonstrates how to use **Ansible handlers** for efficient service management. Handlers are triggered only when their associated task reports a change, improving the idempotency and efficiency of Ansible playbooks. In this lab, you will configure the NTP service, ensuring it is installed, its configuration file is updated, and the service is restarted only when necessary.

**Step-by-Step Lab Instructions**

**Step 1: Prepare the Environment**

1. **On the Control Node:**
   * Ensure Ansible is installed:

bash

Copy code

ansible --version

* + Define your inventory to include the target node under the group demo:

bash

Copy code

cat /etc/ansible/hosts

[demo]

target-node ansible\_host=<target\_ip> ansible\_user=<username>

1. **On the Target Node:**
   * Ensure you have sudo privileges to manage packages and services.

**Step 2: Create the Directory and Playbook**

1. **Set Up the Directory:**
   * Create a new directory to store the playbook and files:

bash

Copy code

mkdir -p ~/ansible-labs/ntp-setup

cd ~/ansible-labs/ntp-setup

1. **Create a Custom NTP Configuration File:**
   * Save a sample configuration in ntp.conf:

bash

Copy code

nano ntp.conf

* + Example content:

csharp

Copy code

# Sample NTP configuration

driftfile /var/lib/ntp/ntp.drift

server 0.centos.pool.ntp.org

server 1.centos.pool.ntp.org

server 2.centos.pool.ntp.org

restrict default kod nomodify notrap nopeer noquery

1. **Create the Ansible Playbook:**
   * Open first.yml for editing:

bash

Copy code

nano first.yml

* + Add the following content:

yaml

Copy code

- hosts: demo

become: yes # Elevate privileges for tasks requiring root access

tasks:

- name: Install the NTP package

package:

name: ntp

state: present

- name: Deploy the NTP configuration file

copy:

src: ntp.conf

dest: /etc/ntp.conf

notify:

- restart ntp service

handlers:

- name: restart ntp service

service:

name: ntp

state: restarted

enabled: yes

**Step 3: Run the Playbook**

1. Test the connection to the target node:

bash

Copy code

ansible -m ping demo

1. Run the playbook:

bash

Copy code

ansible-playbook first.yml

1. Observe the Output:
   * The playbook installs the ntp package, copies the configuration file, and restarts the NTP service only if the file is updated.

**Step 4: Validate the Results**

1. **Check the Package:**
   * On the target node, verify that the ntp package is installed:

bash

Copy code

rpm -qa | grep ntp # For RPM-based systems

dpkg -l | grep ntp # For Debian-based systems

1. **Check the Configuration File:**
   * Ensure the configuration file is deployed correctly:

bash

Copy code

cat /etc/ntp.conf

1. **Check the Service:**
   * Verify the NTP service is running and enabled:

bash

Copy code

systemctl status ntp

systemctl is-enabled ntp

**Step 5: Understanding the Playbook**

* **Tasks:**
  + Install the ntp package if not already installed.
  + Copy the ntp.conf configuration file from the control node to /etc/ntp.conf on the target node.
  + Notify the handler to restart the NTP service only if the file changes.
* **Handler:**
  + Defined under handlers.
  + Triggers the restart ntp service handler to restart the service if the configuration file is modified.

**Step 6: Clean Up (Optional)**

If needed, you can remove the configuration and uninstall the service:

1. Stop the NTP service:

bash

Copy code

sudo systemctl stop ntp

1. Uninstall the NTP package:

bash

Copy code

sudo yum remove ntp -y # For RPM-based systems

sudo apt remove ntp -y # For Debian-based systems

1. Delete the configuration file:

bash

Copy code

sudo rm /etc/ntp.conf

**Step 7: Key Takeaways**

* **Handlers in Ansible:**
  + Handlers only execute when notified, ensuring efficient task execution.
  + Commonly used for restarting services or performing actions that depend on changes in the environment.
* **Idempotency:**
  + Ansible ensures the playbook does not make redundant changes.

This lab showcases the best practices in task automation using Ansible, focusing on using handlers to manage services effectively.

**Lab Guide for Advanced Ansible Playbooks with Variables, Handlers, and Host/Group Vars**

**Overview**

This lab demonstrates advanced Ansible concepts, including the use of variables, handlers, vars\_files, host\_vars, and group\_vars to manage configurations dynamically. You will deploy and configure an Apache web server on target hosts (m1 and m2), modify its behavior, and utilize templating to customize web content.

**Lab Prerequisites**

1. **Control Node Requirements:**
   * Ensure Ansible is installed:

bash

Copy code

ansible --version

1. **Target Nodes (m1 and m2):**
   * Ensure SSH access is set up for the control node.
   * Install Python if not already available.
2. **Inventory File Configuration:**
   * Define your inventory (e.g., /etc/ansible/hosts):

ini

Copy code

[all]

m1 ansible\_host=<m1\_ip>

m2 ansible\_host=<m2\_ip>

1. **Directory Structure:**
   * Verify the following directory and file structure:

python

Copy code

.

├── group\_vars/

│ └── demo

├── host\_vars/

│ ├── m1

│ └── m2

├── vars.yml

├── index.html

├── third.yml

**Lab Steps**

**Step 1: Configure host\_vars and group\_vars**

1. **Group Vars for demo:**
   * Create a file for group variables:

bash

Copy code

mkdir -p group\_vars

nano group\_vars/demo

* + Add the following:

yaml

Copy code

pkgname: apache2

web\_content: |

<h1>This is a demo for Apache Webserver!</h1>

<p>Hostname: {{ ansible\_hostname }}</p>

<p>IP Address: {{ ansible\_default\_ipv4.address }}</p>

rk1: present

1. **Host Vars for m1 and m2:**
   * Create a directory for host variables:

bash

Copy code

mkdir -p host\_vars

* + Define variables for m1:

bash

Copy code

nano host\_vars/m1

* + Add:

yaml

Copy code

ansible\_user: root

* + Define variables for m2:

bash

Copy code

nano host\_vars/m2

* + Add:

yaml

Copy code

ansible\_user: raman

**Step 2: Define Variables File (vars.yml)**

* Create the vars.yml file:

bash

Copy code

nano vars.yml

* Add:

yaml

Copy code

pkgname: apache2

rk1: present

**Step 3: Prepare the Template File (index.html)**

* Create a dynamic web page template:

bash

Copy code

nano index.html

* Add:

html

Copy code

{{ web\_content }}

**Step 4: Create the Playbook (third.yml)**

* Create the main playbook:

bash

Copy code

nano third.yml

* Add the following:

yaml

Copy code

- name: "Deploy {{ pkgname }} webserver"

hosts: all

become: yes

vars\_files:

- vars.yml

tasks:

- name: Update apt package index

ansible.builtin.apt:

update\_cache: yes

- name: Installing {{ pkgname }} packages

ansible.builtin.apt:

name: "{{ pkgname }}"

state: "{{ rk1 }}"

- name: Changing Listening port to 81

lineinfile:

path: /etc/apache2/ports.conf

regexp: "^Listen 80"

line: Listen 81

notify:

- restart\_apache

- name: Changing default site to listen on port 81

ansible.builtin.lineinfile:

path: /etc/apache2/sites-available/000-default.conf

regexp: '^<VirtualHost \\*:80>'

line: "<VirtualHost \*:81>"

notify:

- restart\_apache

- name: Configuring Apache configuration files

template:

src: index.html

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

notify:

- restart\_apache

handlers:

- name: restart\_apache

service:

name: apache2

state: restarted

enabled: yes

**Step 5: Run the Playbook**

1. **Test Connectivity:**
   * Ping the nodes:

bash

Copy code

ansible all -m ping

1. **Deploy the Web Server:**
   * Run the playbook:

bash

Copy code

ansible-playbook third.yml

1. **Validate the Web Server:**
   * Access the web page from a browser:

arduino

Copy code

http://<m1\_ip>:81

http://<m2\_ip>:81

* + You should see:

less

Copy code

<h1>This is a demo for Apache Webserver!</h1>

<p>Hostname: <host\_name></p>

<p>IP Address: <host\_ip></p>

**Step 6: Modify and Test Variables on the Fly**

1. **Override Variables Using -e:**
   * Change the state of apache2 to absent:

bash

Copy code

ansible-playbook third.yml -e rk1="absent"

* + Modify the user1 dynamically:

bash

Copy code

ansible-playbook sec.yaml -e user1="khanna1"

1. **Simulate the Playbook Run (--check):**
   * Dry-run the playbook:

bash

Copy code

ansible-playbook sec.yaml -e test="present" --check

**Step 7: Verify Results**

1. **Check the Package Status:**
   * Confirm the installation/uninstallation of apache2:

bash

Copy code

dpkg -l | grep apache2

1. **Check the Service:**
   * Ensure the service is running on port 81:

bash

Copy code

sudo systemctl status apache2

curl http://<host\_ip>:81

1. **Verify Dynamic Content:**
   * Ensure the index.html file contains host-specific details:

bash

Copy code

cat /var/www/html/index.html

**Step 8: Clean Up**

* Remove the Apache installation:

bash

Copy code

ansible-playbook third.yml -e rk1="absent"

**Key Takeaways**

* **Dynamic Variables:** Use vars\_files, host\_vars, and group\_vars for flexibility and reuse.
* **On-the-Fly Variable Overrides:** Use the -e flag to override variables dynamically during playbook execution.
* **Handlers:** Efficiently manage services by triggering actions only when changes occur.
* **Templating:** Customize configurations and content dynamically with templates.

This lab helps you master Ansible's dynamic capabilities for robust and flexible automation.

This guide covers Ansible playbooks with loops, variables, and error handling techniques. Each lab includes explanations and step-by-step instructions.

**Lab 1: Install Multiple Packages Using Loops**

**Objective**

Learn how to install multiple packages using Ansible loops.

**Playbook**

yaml

Copy code

- name: Install packages on Ubuntu

hosts: all

become: true

gather\_facts: true

tasks:

- name: Install packages

apt:

name: "{{ item }}"

state: present

update\_cache: yes

loop:

- php

- gcc

- talk

- vim

- apache2

when: ansible\_distribution == "Ubuntu" and ansible\_distribution\_major\_version == "24"

**Steps**

1. Create a file loops2.yml in the control node.
2. Add the above playbook content.
3. Execute the playbook:

bash

Copy code

ansible-playbook loops2.yml

1. Verify the packages are installed:

bash

Copy code

dpkg -l | grep -E "php|gcc|talk|vim|apache2"

**Lab 2: User Creation with Loops**

**Objective**

Understand how to create multiple users using loop constructs.

**Playbook**

yaml

Copy code

---

- hosts: m1

tasks:

- name: Create multiple users with loop

user:

name: "{{ item }}"

state: present

shell: /bin/bash

loop:

- gagan1

- gagan2

- gagan3

- name: Second task for loop with conditions

command: echo {{ item }}

loop: [1, 3, 5, 7, 8, 9]

when: item < 5

**Steps**

1. Create a file loop1.yml.
2. Populate the file with the playbook above.
3. Run the playbook:

bash

Copy code

ansible-playbook loop1.yml

1. Verify user creation:

bash

Copy code

getent passwd gagan1 gagan2 gagan3

**Lab 3: Variables with Loops**

**Objective**

Use variables with loops to manage both packages and user creation dynamically.

**Playbook**

yaml

Copy code

- name: Install packages and create users

hosts: all

become: true

gather\_facts: true

vars:

rk1: present

users:

- name: ramank

uid: 9876

- name: rakeshk

uid: 8796

- name: ravik

uid: 9877

pkgs:

- php

- gcc

- talk

- vim

- apache2

tasks:

- name: Create users

user:

name: "{{ item.name }}"

uid: "{{ item.uid }}"

state: "{{ rk1 }}"

loop: "{{ users }}"

- name: Install packages

apt:

name: "{{ item }}"

state: "{{ rk1 }}"

update\_cache: yes

loop: "{{ pkgs }}"

when: ansible\_distribution == "Ubuntu" and ansible\_distribution\_major\_version == "24"

**Steps**

1. Save the file as loops2.yml.
2. Execute the playbook:

bash

Copy code

ansible-playbook loops2.yml

1. Verify:
   * User creation:

bash

Copy code

getent passwd ramank rakeshk ravik

* + Package installation:

bash

Copy code

dpkg -l | grep -E "php|gcc|talk|vim|apache2"

**Lab 4: Ignoring Errors**

**Objective**

Understand how to use ignore\_errors to bypass errors during execution.

**Playbook**

yaml

Copy code

---

- name: Ignore Error Playbook

hosts: all

tasks:

- name: This will not be counted as failure

command: /bin/date12

ignore\_errors: yes

- name: Second task

service:

name: ntpd

state: restarted

**Steps**

1. Create the file ignore\_error.yaml.
2. Add the playbook content.
3. Run the playbook:

bash

Copy code

ansible-playbook ignore\_error.yaml

1. Observe that the playbook continues despite the first task failing:
   * Logs will indicate the failure of /bin/date12, but subsequent tasks will execute.

**Best Practices and Tips**

* Use gather\_facts: true to ensure all required variables (like ansible\_distribution) are available.
* Use ignore\_errors sparingly, only when failures should not halt the execution.
* Always verify results after playbook execution to ensure tasks achieved the desired state.

This lab guide equips you to manage dynamic configurations and error handling in Ansible effectively.