**Topics**

1. **What is Ansible?**
   * Ansible is an open-source automation tool used for configuration management, application deployment, and task automation.
   * It uses simple YAML files called playbooks to automate tasks on remote machines.
   * Ansible is agentless, meaning it doesn't require any software to be installed on the remote systems.
2. **Ansible Automation with Python**:
   * Python can be used to trigger Ansible playbooks, pass variables, and manage inventory.
   * The ansible Python library (ansible-runner) is used for interacting with Ansible.
   * ansible-playbook command can be invoked using Python's subprocess module to automate execution.
3. **Common Use Cases for Ansible Automation**:
   * Automating server setup and configuration.
   * Deploying applications to multiple environments.
   * Managing cloud resources like AWS, Azure, or GCP.

**Practical Exercises**

**1. Install Ansible and ansible-runner**

* First, you need to install Ansible and the Python ansible-runner library.

bash

Copy code

pip install ansible-runner

sudo apt update

sudo apt install ansible

**2. Run a Simple Ansible Playbook with Python**

* **Create a sample Ansible Playbook**: install\_nginx.yml

yaml

Copy code

---

- name: Install Nginx on remote server

hosts: all

become: yes

tasks:

- name: Install Nginx

apt:

name: nginx

state: present

* **Run the Playbook using Python**:

python

Copy code

import subprocess

playbook = "install\_nginx.yml"

command = ["ansible-playbook", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

if result.returncode == 0:

print("Playbook executed successfully.")

print(result.stdout)

else:

print(f"Error running playbook: {result.stderr}")

**3. Automate Playbook Execution with Dynamic Inventory**

* **Create a dynamic inventory**: This allows you to pass in dynamic host groups or IPs.

python

Copy code

import json

import subprocess

# Example of dynamic inventory

inventory = {

"all": {

"hosts": ["192.168.1.1", "192.168.1.2"],

"vars": {"ansible\_user": "ubuntu"}

}

}

# Save dynamic inventory to file

with open("inventory.json", "w") as f:

json.dump(inventory, f)

# Run the playbook with dynamic inventory

playbook = "install\_nginx.yml"

command = ["ansible-playbook", "-i", "inventory.json", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

if result.returncode == 0:

print("Playbook executed successfully.")

print(result.stdout)

else:

print(f"Error running playbook: {result.stderr}")

**4. Pass Variables to Playbook via Python**

* Objective: Pass variables dynamically to an Ansible playbook.
* **Modify your Ansible playbook** to accept variables:

yaml

Copy code

---

- name: Install Nginx on remote server

hosts: all

become: yes

vars:

nginx\_package: "{{ nginx\_package\_name }}"

tasks:

- name: Install Nginx

apt:

name: "{{ nginx\_package }}"

state: present

* **Pass variables via Python**:

python

Copy code

import subprocess

playbook = "install\_nginx.yml"

variables = {"nginx\_package\_name": "nginx"}

command = ["ansible-playbook", "-e", f"nginx\_package\_name={variables['nginx\_package\_name']}", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

if result.returncode == 0:

print("Playbook executed successfully with custom variables.")

print(result.stdout)

else:

print(f"Error running playbook: {result.stderr}")

**5. Check the Status of a Remote Server**

* Objective: Use Ansible to check the status of a service like nginx on a remote server.
* **Ansible Playbook (check\_nginx.yml)**:

yaml

Copy code

---

- name: Check if Nginx is running on remote server

hosts: all

become: yes

tasks:

- name: Check service status

service:

name: nginx

state: started

register: service\_status

- name: Display status

debug:

msg: "Nginx is {{ service\_status.state }}"

* **Run the Playbook**:

python

Copy code

import subprocess

playbook = "check\_nginx.yml"

command = ["ansible-playbook", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

if result.returncode == 0:

print("Playbook executed successfully.")

print(result.stdout)

else:

print(f"Error running playbook: {result.stderr}")

**6. Automate Ansible with Cron Jobs**

* Objective: Schedule playbook execution on a regular basis using cron.
* **Example Cron Job** to run the playbook daily at midnight:

bash

Copy code

0 0 \* \* \* /usr/bin/python3 /path/to/your/playbook\_script.py

**7. Challenge: Automatically Deploy a Web Application with Ansible**

* **Create a simple playbook (deploy\_app.yml)** to deploy a web application to remote servers:

yaml

Copy code

---

- name: Deploy web application

hosts: all

become: yes

tasks:

- name: Clone the repository

git:

repo: "https://github.com/yourusername/yourrepo.git"

dest: "/var/www/html"

version: master

- name: Install dependencies

command: "pip install -r /var/www/html/requirements.txt"

- name: Restart web server

service:

name: nginx

state: restarted

* **Run the deployment script** using Python:

python

Copy code

import subprocess

playbook = "deploy\_app.yml"

command = ["ansible-playbook", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

if result.returncode == 0:

print("Web application deployed successfully.")

print(result.stdout)

else:

print(f"Error deploying app: {result.stderr}")

**8. Bonus: Handling Ansible Output and Logs with Python**

* Objective: Process and store the output from Ansible runs into a log file.
* **Script**:

python

Copy code

import subprocess

playbook = "deploy\_app.yml"

command = ["ansible-playbook", playbook]

result = subprocess.run(command, capture\_output=True, text=True)

with open("ansible\_output.log", "w") as log\_file:

log\_file.write(result.stdout)

if result.returncode == 0:

print("Playbook executed successfully, output saved to ansible\_output.log.")

else:

print(f"Error running playbook, check ansible\_output.log for details.")

**Summary**

* **Day 8** focused on integrating **Ansible** with Python for automating various tasks like server setup, configuration, application deployment, and service management.
* By using Python’s subprocess module, we can execute and control Ansible playbooks, pass variables dynamically, and automate tasks like web application deployment and service monitoring.