**Ansible Lab Exercises**

**Q1. Ansible Installation and Verification**

**Objective:**

1. Install Ansible on an Ubuntu system and verify its successful installation.

**Tasks:**

1. Update the system's package index.
2. Install the required package for managing PPAs (software-properties-common).
3. Add the official Ansible PPA (Personal Package Archive) to the system's repository.
4. Install Ansible using the package manager.
5. Verify the Ansible installation by checking its version.

**Expected Outcome:**

* Ansible is installed successfully, and the version is displayed along with relevant configuration details when the ansible --version command is executed.

**Q2. Create and Run Custom Ansible Inventory for Host Communication**

**Objective:**

Set up an Ansible inventory file to define and manage hosts for communication, and verify the setup using Ansible commands.

**Tasks:**

1. Create an inventory file named inventory.ini in your Ansible directory (e.g., /home/<user>/ansible/).
2. Add a group named [myhosts] in the inventory file and specify the IP addresses or FQDN of the host system along with their SSH user credentials.
3. Verify the inventory setup by listing the host using the Ansible command.
4. Ping the myhosts group using Ansible to ensure communication between the host VM and defined systems.

**Expected Outcome:**

* The inventory file is correctly configured, and all defined hosts are listed successfully.
* Ansible ping command confirms communication between the host VM and the specified systems in the inventory.

**Q3. Create and Run Custom Ansible Inventory for Container as a Client System**

**Objective:**

Set up an Ansible inventory file to define and manage a Docker container as a client system, and verify communication from the host system using Ansible commands.

**Tasks:**

1. **Create a Docker Container**:
   * Launch a Docker container with SSH enabled and ensure it is running.
2. **Configure Ansible Inventory**:
   * Create an inventory file named inventory.ini in your Ansible directory (e.g., /home/<user>/ansible/).
   * Add a group named [mycontainers] in the inventory file and specify the container's IP address along with the SSH user credentials.
3. **Verify Inventory Setup**:
   * Use the Ansible command to list all hosts in the mycontainers group.
4. **Test Communication**:
   * Use the Ansible ping module to verify communication between the host system and the container defined in the inventory file.

**Expected Outcome:**

* The inventory file correctly identifies the container as a client system.
* The Ansible ping command confirms successful communication from the host system to the container.

**Q4. Writing and Executing Your First Ansible Playbook**

**Objective:**

Learn how to create and execute a simple Ansible playbook to communicate with managed hosts and perform basic tasks.

**Tasks:**

1. **Create an Inventory File**:
   * Write an inventory.ini file to define the hosts. Include:
     + localhost with the Python interpreter path.
     + A remote system (e.g., 172.17.0.2) with SSH credentials.
2. **Write a Playbook**:
   * Create a playbook named playbook.yml with the following tasks:
     + Ping all hosts in the myhosts group to verify connectivity.
     + Print a message ("Hello world") on the console using the debug module.
3. **Run the Playbook**:
   * Execute the playbook using the Ansible command: ansible-playbook -i inventory.ini playbook.yml
4. **Verify the Output**:
   * Ensure the playbook:
     + Successfully pings all hosts in the inventory.
     + Displays the message "Hello world" in the output.

**Expected Outcome:**

* The ping task confirms communication with all hosts.
* The debug task displays "Hello world" as specified in the playbook.
* The play recap section shows ok=3, changed=0, and no errors.

**Q5. Creating and Managing File Types with Ansible Playbook**

**Objective:**

Learn how to create and manage different types of files and directories in a specific path using Ansible.

**Tasks:**

1. **Inventory Setup**:
   * Define an inventory file (inventory.ini) with the following:
     + Specify the localhost with the correct Python interpreter.
     + Add necessary SSH credentials for localhost.
2. **Write a Playbook**:
   * Create a playbook named playbook.yml to perform the following tasks:
     + Ensure the directory /home/raju/spm/Ansible/q5/temp exists.
     + Create a regular file named regular\_file.
     + Create a subdirectory named opt inside the specified path.
     + Create a character device.
     + Create a block device.
     + Create a symbolic link to the regular\_file.
     + Create a hard link to the regular\_file.
     + Create a FIFO file.
3. **Execution**:
   * Run the playbook using the following command: ansible-playbook -i inventory.ini playbook.yml
4. **Verification**:
   * Verify that the playbook tasks are successfully executed and the expected files and directories are created at /home/raju/spm/Ansible/q5/temp.
   * Use the ls -l command to confirm the presence of the created files and links.

**Expected Outcome:**

* The playbook creates the specified file types and directory structure under /home/raju/spm/Ansible/q5/temp.
* The play recap confirms that all tasks are completed successfully with ok=8, changed=7, and no errors.

**Q6. Container Web Server Creation with Ansible**

**Objective:** Create and configure a container-based web server using Ansible and validate its functionality through a browser.

**Steps to Complete:**

1. **Run a Container:** Start a new container to create an isolated environment for the web server.
2. **Install OpenSSH Server:** Set up an OpenSSH server within the container.
3. **Modify SSH Configuration:** Update the SSH configuration in the container to allow root login by editing the sshd\_config file and enabling PermitRootLogin yes.
4. **Start SSH Service:** Restart the SSH service and verify its status.
5. **Establish Communication:** Ensure SSH communication between the host and the container.
6. **Set Up Keys:** Generate and copy the SSH public key from the host to the container.
7. **Install Python:** Install Python in the container to enable Ansible execution.
8. **Create the Inventory File:** Write an inventory file (inventory.ini) specifying container details, including IP, SSH credentials, and Python interpreter path.
9. **Write the Ansible Playbook:** Develop a playbook (playbook.yml) to:
   * Install Apache2.
   * Add custom content to the web server.
   * Start the Apache2 service.
10. **Run the Playbook:** Execute the playbook from the host to automate the web server setup.
11. **Validate the Setup:** Access the container's IP through a web browser and verify the displayed content.

**Expected Outcome:**  
Upon accessing the container's IP, the browser should display the message:  
*"Congratulations, your webserver is working..."*

**Q7. Demonstrate Variable Types in Ansible**

**Objective:** Write an Ansible playbook to demonstrate the use of different variable types, including string, numerical, list, dictionary, and boolean variables.

**Instructions:**

1. Create a playbook named playbook.yml.
2. Define the following variable types in the playbook:
   * String Variable
   * Numerical Variable
   * List Variable
   * Dictionary Variable
   * Boolean Variable
3. Use the debug module to display the values of these variables.
4. Execute the playbook and observe the output.

**Expected Outcome:**  
The playbook should display all the variable values in the output, demonstrating the use of different variable types in Ansible.

**Q8. Using Ansible Templates**

1. Create an Ansible playbook that:
   * Generates a dynamic configuration file using a Jinja2 template.
   * The template should include placeholders for application name, environment, version, and database configurations (host, port, name, user, password).
   * Variables for these placeholders should be defined within the playbook.
2. Perform the following:
   * Create a directory named configurations in the user's home directory on the target host.
   * Use the template module to generate the configuration file in the configurations directory.
   * Display the contents of the generated configuration file.
3. Verify:
   * Ensure the configuration file is created with the correct values.
   * Confirm the playbook runs successfully without errors.

**Expected Output:** A configuration file with all placeholders replaced by the respective values defined in the playbook.

**Q9. Demonstrate Ansible Numerical Loop with Directory Creation**

Create an Ansible playbook that demonstrates the use of a numerical loop. The playbook should create five directories named demo\_dir\_1 to demo\_dir\_5 in the given directory using a loop construct.

Verify the directories are created successfully after running the playbook.

**Q10. Demonstrate Conditionals in Ansible**

1. Write an Ansible playbook to:
   * Check the OS family of the target system.
   * Install the vim package if the OS is Debian-based.
   * Display a message confirming the installation on Debian systems.
   * Print a message skipping the installation for non-Debian systems.
2. Execute the playbook on your localhost and verify the conditional behavior based on your OS.

**Q11. Exploring Ansible Facts**

**Objective:** Learn how to gather and display system information using Ansible facts.

**Task Instructions:**

1. **Create an Inventory File:**
   * Write an inventory.ini file that lists all the hosts (e.g., localhost) you want to target.
2. **Write a Playbook:**
   * Create an Ansible playbook to:
     + Gather system facts for the target hosts.
     + Display the following system details:
       - Operating System Name and Version
       - Kernel Version
       - Memory Information
       - CPU Count
       - Hostname
       - IP Address
3. **Execute the Playbook:** ansible-playbook -i inventory.ini playbook.yml
4. **Verify the Output:** Check the console output to ensure the gathered facts are displayed correctly.

**Expected Outcome:** The playbook should successfully gather and display the required system information for the target hosts.

**Q12. Disk Space Monitoring with Ansible**

Create a playbook to monitor disk usage on a local system.

1. Use a shell command to check the disk usage of a specific partition.
2. Register the output of the command to a variable.
3. Implement a conditional task to display a warning message if the disk usage exceeds 80%.

Execute the playbook and verify the warning message when the disk usage crosses the threshold.

**Q13. Disk Space Monitoring with Email Alert**

**Objective:** Create an Ansible playbook to monitor disk space on the localhost. If the disk usage exceeds 80%, send an automated email alert with the current disk usage details.

**Tasks to Perform:**

1. Write an Ansible playbook with the following functionality:
   * Check available disk space using the df command.
   * Register the output and evaluate whether the disk usage exceeds 80%.
   * If the usage exceeds the threshold, send an email alert using the mail module with:
     + SMTP server details.
     + Recipient email.
     + Warning message containing the disk usage percentage.
2. Execute the playbook and test the email alert functionality by modifying disk space to exceed 80% (e.g., by adding dummy files).

**Expected Outcome:** The playbook should detect the disk space usage. If usage is above 80%, an email alert with disk usage details should be sent to the specified recipient.

**Q14. Demonstrate ignore\_errors in Ansible**

**Objective:** Create a playbook to demonstrate the use of ignore\_errors in Ansible to allow the playbook to continue execution even after a task failure.

**Tasks:**

1. Attempt to remove a non-existent file and ensure the playbook ignores the error.
2. Display a debug message indicating the continuation of the playbook execution.
3. Create a directory named example\_directory in the /home/raju/spm/Ansible/q14 path.

**Expected Outcome:** The playbook should:

* Ignore the error when trying to remove the non-existent file.
* Display the debug message.
* Successfully create the specified directory.

**Q15. Implement Ansible Roles**

**Objective**: Demonstrate the use of Ansible roles to organize tasks and handlers for managing a service.

**Instructions**:

1. Create an Ansible role named common to perform the following:
   * Install the ntp package.
   * Ensure the ntp service is started and enabled.
   * Restart the service if required.
2. Use the provided directory structure for the role:
   * roles/common/tasks/main.yml
   * roles/common/handlers/main.yml
3. Write a playbook playbook.yml to apply the role on the target host(s).
4. Create an inventory file inventory.ini to define the host(s).
5. Execute the playbook and verify the following:
   * The ntp package is installed.
   * The ntp service is running and enabled.
   * The handler restarts the service when necessary.

**Q16. Demonstrating Vault Usage with Ansible**

**Objective:** Securely manage sensitive data using Ansible Vault and demonstrate its usage in a playbook.

**Instructions:**

1. Create an inventory file with a group named local and include localhost under it.
2. Use Ansible Vault to create a secrets.yml file containing a variable for a secret password.
3. Write a playbook that:
   * Targets the local group.
   * Uses the encrypted secrets.yml file for variables.
   * Displays the secret password using a debug task.
4. Run the playbook with Ansible Vault, ensuring no warnings appear.

**Expected Outcome:** The playbook successfully decrypts and displays the secret password without warnings.

**Mini Project: Setting Up a "Hello World" C Program Environment Using Ansible**

**Objective:** The goal of this mini-project is to set up an environment for developing, testing, and deploying a "Hello World" C program using Docker containers and Ansible. You will write an inventory file and create multiple playbooks to automate tasks for development, testing, and deployment stages.

**Steps to Follow:**

1. **Container Setup**:
   * Create a base Docker image (mini\_proj) with the necessary tools (e.g., gcc, vim, openssh-server, and python3).
   * Use the base image to create three containers for different environments:
     + dev for development
     + test for testing
     + deploy for deployment
2. **Configure the Inventory File**:
   * Write an inventory.ini file categorizing the containers into [development], [testing], and [deployment] groups.
   * Specify the IP addresses of the containers in the inventory file.
3. **Development Stage**:
   * Create a playbook to:
     + Install GCC compiler inside the dev container.
     + Copy a hello.c file from the host to the container.
     + Compile the program to generate the hello executable.
4. **Testing Stage**:
   * Create a playbook to:
     + Copy the hello executable from the dev container to the test container.
     + Run the executable inside the test container.
     + Capture and display the output of the hello program.
5. **Deployment Stage**:
   * Create a playbook to:
     + Copy the hello executable from the test container to the deploy container.
     + Run the executable inside the deploy container.
6. **Verification**:
   * Run the playbooks sequentially for each stage.
   * Verify the output at each stage (Hello, World!).
   * Ensure all tasks are automated through Ansible.

**Expected Deliverables:**

* A Docker image named mini\_proj.
* Three Docker containers (dev, test, deploy) running the Hello World program.
* An inventory.ini file categorizing the containers.
* Ansible playbooks for development, testing, and deployment stages.
* Outputs verifying the successful execution of tasks.