**Lab Guide: Ad-Hoc Commands in Ansible**

This guide walks through the use of Ansible's ad-hoc commands to perform operations on managed nodes. It includes details about configuration, executing commands, and manipulating remote systems.

**Prerequisites**

1. **Ansible Installation:** Ensure Ansible is installed on the control node.
   * Install using:

bash

Copy code

sudo apt update && sudo apt install ansible -y

1. **SSH Access:** Configure passwordless SSH access to managed nodes.
2. **Inventory Configuration:** Update the /etc/ansible/hosts file with target node details.

**Step-by-Step Guide**

**1. Verify Ansible Setup**

* Check Ansible is installed and working:

bash

Copy code

ansible --version

**2. Define Inventory**

* Open the hosts file:

bash

Copy code

vi /etc/ansible/hosts

* Add managed hosts under the [demo] group:

csharp

Copy code

[demo]

m1 ansible\_host=172.31.25.29

m2 ansible\_host=172.31.24.46

**3. Ping All Hosts**

* Verify connectivity to all hosts:

bash

Copy code

ansible all -m ping

* Expected output:

javascript

Copy code

m1 | SUCCESS => ...

m2 | SUCCESS => ...

**4. List Hosts in Inventory**

* List all hosts in the demo group:

bash

Copy code

ansible demo --list-hosts

* List all hosts in the all group:

bash

Copy code

ansible all --list-hosts

* Target specific hosts using indices:
  + First host:

bash

Copy code

ansible demo[0] --list-hosts

* + Last host:

bash

Copy code

ansible demo[-1] --list-hosts

**5. Execute Basic Commands**

* Run a command on all hosts in the demo group:

bash

Copy code

ansible demo -a "ls"

* List files with details:

bash

Copy code

ansible demo -a "ls -ltra"

**6. Check Directory on Specific Host**

* Run a command on the first host in the group:

bash

Copy code

ansible demo[0] -a "ls /home/ubuntu/"

**7. Install Packages**

* Update package lists on a specific host:

bash

Copy code

ansible demo[1] -a "apt update -y"

* Install the tree package:

bash

Copy code

ansible demo[1] -a "apt install tree -y"

**8. Verify Package Installation**

* Check if the tree command is available:

bash

Copy code

ansible demo -a "which tree"

**9. Create and Manage Files**

* Create a file on all hosts:

bash

Copy code

ansible demo -a "touch /home/ubuntu/ramanfile"

* List files to confirm creation:

bash

Copy code

ansible demo -a "ls -l /home/ubuntu/"

**10. Get System Information**

* Display the kernel and system details:

bash

Copy code

ansible demo -a "uname -a"

**Extra Tips**

1. **Check Ansible Help:**
   * Explore more ad-hoc command options:

bash

Copy code

ansible -h

1. **Combine Commands:**
   * Run multiple commands with &&:

bash

Copy code

ansible m1 -a "apt update -y && apt install tree -y"

1. **Fine-Grained Targeting:**
   * Use indices and ranges to target subsets of hosts:

bash

Copy code

ansible demo[0:2] -a "ls"

**Cleanup**

* Remove the test file:

bash

Copy code

ansible demo -a "rm /home/ubuntu/ramanfile"

* Confirm deletion:

bash

Copy code

ansible demo -a "ls -l /home/ubuntu/"

**Summary**

This lab demonstrated the use of Ansible ad-hoc commands for basic tasks like file management, package installation, and system information retrieval. You can extend this to automate more complex tasks efficiently.

**Lab Guide for Ad-Hoc Ansible Commands and Modules**

This lab guide focuses on using Ansible's ad-hoc commands and modules to perform tasks on a remote host. The tasks covered in this guide are essential for managing configurations, installing packages, managing services, and retrieving system information.

**Objective:**

* Use Ansible's ad-hoc commands to manage remote hosts.
* Utilize different Ansible modules for common system administration tasks.
* Learn how to configure the ansible.cfg file and hosts inventory for specific host configurations.

**Pre-requisites:**

* Ansible installed on the local machine.
* Access to remote machines with SSH enabled.
* Proper configuration in /etc/ansible/ansible.cfg and /etc/ansible/hosts.

**Lab 1: Configure Inventory with Custom Host Variables**

1. **Editing the hosts file**:
   * Open the Ansible inventory file located at /etc/ansible/hosts.
   * Add the following entries for your demo hosts:

ini

Copy code

[demo]

m1 ansible\_host=172.31.25.29

m2 ansible\_host=172.31.24.46

* + This configuration assigns m1 and m2 with specific IP addresses (ansible\_host).

1. **Verifying the Hosts Configuration**:
   * Run the following command to check the hosts:

bash

Copy code

ansible demo --list-hosts

* + - This will list the hosts in the demo group.

**Lab 2: Running Ad-Hoc Commands with Ansible**

1. **Basic Command (Ping)**:
   * Ping all hosts in the demo group to ensure they are reachable:

bash

Copy code

ansible demo -m ping

1. **Executing Commands**:
   * List the contents of the /home/ubuntu/ directory on all hosts:

bash

Copy code

ansible demo -a "ls /home/ubuntu/"

* + Execute a more detailed listing with ls -ltra:

bash

Copy code

ansible demo -a "ls -ltra"

1. **Installing Packages**:
   * Install the tree package on the second host (m2):

bash

Copy code

ansible demo[1] -a "apt install tree -y"

* + Alternatively, use Ansible’s package module for better control:

bash

Copy code

ansible demo -m apt -a "name=tree state=present"

1. **Checking if tree is installed**:
   * Check if tree is installed:

bash

Copy code

ansible demo -a "which tree"

1. **Creating Files**:
   * Create a file called ramanfile in /home/ubuntu/:

bash

Copy code

ansible demo -a "touch /home/ubuntu/ramanfile"

1. **Listing Files**:
   * List the contents of /home/ubuntu/ to verify ramanfile exists:

bash

Copy code

ansible demo -a "ls -l /home/ubuntu/"

**Lab 3: Using Ansible Modules for System Administration**

1. **Package Management**:
   * Remove the tree package:

bash

Copy code

ansible demo -a "apt remove tree -y"

* + Alternatively, use the package module to ensure the package is absent:

bash

Copy code

ansible demo -m apt -a "name=tree state=absent"

1. **Managing Users**:
   * Create a user khanna with specific attributes:

bash

Copy code

ansible demo -m user -a "name=khanna comment='created via ansible cli' group=root shell=/bin/bash"

* + Remove the user khanna:

bash

Copy code

ansible demo -m user -a "name=khanna state=absent"

1. **Managing Services**:
   * Disable and stop the nginx service:

bash

Copy code

ansible m1 -m service -a "name=nginx enabled=false state=stopped"

* + Enable and start the nginx service:

bash

Copy code

ansible m1 -m service -a "name=nginx enabled=true state=started"

1. **Running Commands**:
   * Execute a simple uptime command:

bash

Copy code

ansible demo -m command -a "uptime"

**Lab 4: Working with Ansible Documentation**

1. **List Available Modules**:
   * To see all available modules, run:

bash

Copy code

ansible-doc -l

1. **Search for a Specific Module**:
   * To search for the ping module, use the following:

bash

Copy code

ansible-doc -l | grep ping

1. **Get Detailed Information on a Module**:
   * To get detailed information on the ping module:

bash

Copy code

ansible-doc -v ping

1. **Using Ansible with Documentation for Custom Commands**:
   * Explore other modules, such as the apt or package modules, using ansible-doc:

bash

Copy code

ansible-doc -l | grep apt

**Lab 5: Managing Remote Hosts Efficiently**

1. **Check and Update Software**:
   * Update and install tree on m1:

bash

Copy code

ansible m1 -a "apt update -y && apt install tree -y"

1. **Remove a Package**:
   * Remove the nginx package from the demo group:

bash

Copy code

ansible demo -m package -a "name=nginx state=absent"

**Lab 6: Troubleshooting and Debugging**

1. **Check Host Connectivity**:
   * Use ping to verify all hosts are reachable:

bash

Copy code

ansible all -m ping

1. **Verify Package Installation**:
   * Check if tree is installed across all hosts:

bash

Copy code

ansible all -a "which tree"

**Conclusion**

By following the steps in this lab guide, you’ve gained hands-on experience with Ad-Hoc commands in Ansible, such as managing packages, services, users, and executing commands remotely. You also explored how to use Ansible's documentation to search for modules and get detailed information to enhance your automation skills.

**Ansible Playbook Lab Guide**

**Objective:**

In this lab, you will create and run an Ansible playbook that installs Nginx and creates directories on remote servers using Ansible. This lab involves multiple examples with different configurations to help understand various scenarios and tasks within Ansible playbooks.

**Prerequisites:**

1. **Ansible Installed:** Ensure that Ansible is installed on your local machine (controller node). You can verify this by running:

bash

Copy code

ansible --version

1. **Remote Servers:** You should have two or more remote servers (e.g., m1 and m2) configured and accessible via SSH.
2. **Inventory File:** Ansible needs to know the hosts (servers) to target. Typically, this is set in the /etc/ansible/hosts file or through a custom inventory file.

Example inventory (/etc/ansible/hosts or inventory.ini):

csharp

Copy code

[demo]

m1 ansible\_host=192.168.1.101

m2 ansible\_host=192.168.1.102

1. **SSH Access:** Ensure you have SSH access to the remote servers. You may need to configure the SSH keys for passwordless login.

**Part 1: Simple Playbook Example**

**1.1 Create the Playbook File:**

Create a playbook named first.yml on the controller node:

bash

Copy code

nano first.yml

**1.2 Add the Following Content:**

yaml

Copy code

- hosts: m1,m2

user: root

connection: ssh

gather\_facts: true

tasks:

- name: Install Nginx on both servers

apt:

name: nginx

state: present

- hosts: m1

tasks:

- name: Create a directory on m1

file:

state: directory

path: /home/ubuntu/ramantestdir

**1.3 Breakdown:**

* **Hosts (m1,m2)**: This part specifies which hosts (servers) the tasks will target. In this case, m1 and m2 from the inventory file are targeted.
* **User**: Specifies the user Ansible will use to connect to the remote hosts. Here, root is used for privilege escalation.
* **Connection**: Defines the connection method, ssh in this case.
* **Gather Facts**: Set to true, Ansible will collect basic facts about the remote servers, like operating system details, network interfaces, etc.
* **Tasks**:
  + **Install Nginx**: Installs the Nginx package on both m1 and m2 servers using the apt module (for Ubuntu/Debian-based systems).
  + **Create Directory**: On the m1 server, creates a directory /home/ubuntu/ramantestdir using the file module.

**1.4 Run the Playbook:**

Execute the playbook to install Nginx and create the directory:

bash

Copy code

ansible-playbook first.yml

**Part 2: Additional Playbook with Specific Tasks for Each Host**

**2.1 Modify Playbook for Specific Hosts:**

In the following example, a directory is created only on m1, while Nginx is installed on both m1 and m2.

yaml

Copy code

- hosts: m1,m2

user: root

connection: ssh

gather\_facts: true

tasks:

- name: Install Nginx on both servers

apt:

name: nginx

state: present

- hosts: m1

tasks:

- name: Create a directory on m1

file:

state: directory

path: /home/ubuntu/khannadir

**2.2 Explanation:**

* **Nginx Installation**: This part is the same as before, and the Nginx package is installed on both servers (m1 and m2).
* **Directory Creation**: A new directory /home/ubuntu/khannadir is created only on m1.

**2.3 Run the Modified Playbook:**

bash

Copy code

ansible-playbook first.yml

**Part 3: Using become for Privilege Escalation**

This example demonstrates how to use the become option to execute commands with elevated privileges (root) on remote servers.

**3.1 Create Playbook File:**

Create or modify a playbook first.yml to include become and become\_user options:

yaml

Copy code

- hosts: demo

user: ubuntu

become: yes

become\_user: root

connection: ssh

gather\_facts: true

tasks:

- name: Install Nginx on both servers

apt:

name: nginx

state: present

- name: Create a directory as root on both servers

file:

state: directory

path: /root/ramantestdir

**3.2 Explanation:**

* **Become (become: yes)**: This option allows Ansible to escalate privileges to root on the remote servers.
* **Become User (become\_user: root)**: Specifies that the root user will be used for the tasks requiring elevated privileges.

**3.3 Run the Playbook:**

bash

Copy code

ansible-playbook first.yml

**Part 4: Using Different Users and Custom Paths**

**4.1 Example Playbook with Different Users:**

yaml

Copy code

- hosts: demo

user: ubuntu

become: yes

become\_user: root

connection: ssh

gather\_facts: true

tasks:

- name: Install Nginx on both servers

apt:

name: nginx

state: present

- name: Create a directory on both servers

file:

state: directory

path: /root/ramantestdir

**4.2 Explanation:**

This example is very similar to **Part 3**, but in this case, the directory /root/ramantestdir is created on both servers (demo group).

**Troubleshooting and Useful Commands**

* **Check the Syntax of the Playbook**:

bash

Copy code

ansible-playbook --syntax-check first.yml

* **View Available Modules**: To get information on all available modules for your tasks, use the ansible-doc command:

bash

Copy code

ansible-doc -l

* **Check the Facts Collected by Ansible**: You can check the gathered facts using the setup module:

bash

Copy code

ansible m1 -m setup

**Conclusion:**

In this lab, you learned how to create and execute Ansible playbooks with tasks like installing packages (Nginx) and creating directories on remote servers. You also explored various configurations, including privilege escalation and targeting specific hosts for specific tasks.

**Ansible Privilege Escalation and User Management Lab Guide**

**Objective:**

In this lab, you will learn how to manage privilege escalation in Ansible, work with different user accounts, and configure user permissions on target machines. You will create a user, assign privileges, and use various privilege escalation options to run tasks as different users.

**Prerequisites:**

1. **Ansible Installed**: Ensure that Ansible is installed on the controller node.

Check the installation:

bash

Copy code

ansible --version

1. **Managed Nodes**: Ensure you have a few remote servers (m1, m2, etc.) where Ansible can perform tasks. These servers must have SSH access from the controller node.
2. **Root and Standard User Access**: You need to have root access to the managed nodes to configure user accounts and sudo permissions.

**Part 1: Privilege Escalation Basics**

**1.1 Privilege Escalation Options in Ansible**

Ansible provides several options for privilege escalation when performing tasks that require higher privileges (e.g., root user). The options include:

* **--become-method**: Specifies the method for privilege escalation. The default method is sudo, but others like su, pbrun, etc., are possible. You can list available methods using:

bash

Copy code

ansible-doc -t become -l

* **--become-user**: Specifies which user to become when running the tasks. The default user is root. Example:

bash

Copy code

ansible-playbook first.yml --become-user raman

* **-b, --become**: This flag tells Ansible to use privilege escalation. You don’t need to specify a password with --become if you configure passwordless sudo for the user.

**1.2 Example: Run a Task with Privilege Escalation**

In the following example, you will run a task with the raman user, and escalate privileges to the root user.

bash

Copy code

ansible demo -m apt -a "mkdir /etc/configapp" -u ubuntu --become-user raman -b

* **-u ubuntu**: Specifies the remote user (ubuntu).
* **--become-user raman**: Runs the task as the raman user after using sudo to escalate privileges.
* **-b**: This flag tells Ansible to perform the task with privilege escalation (i.e., sudo).

**Part 2: Creating and Managing Users**

**2.1 Create a New User on All Nodes**

To create a new user (raman) on all managed nodes (m1, m2), log into the managed nodes as root and use the following commands:

bash

Copy code

useradd raman

passwd raman

1. **Create the User (raman)**: This will add the user to the system.
2. **Set Password**: Set a password for the raman user.

**2.2 Add the User to the Sudoers File**

In order to allow raman to perform administrative tasks without a password prompt, add the user to the /etc/sudoers file. On each managed node, run:

bash

Copy code

vi /etc/sudoers

Add the following line for the raman user:

bash

Copy code

raman ALL=(ALL) NOPASSWD: ALL

This line allows the raman user to run all commands as root without needing to enter a password.

**Part 3: Ansible Playbook Examples with Different User Configurations**

**3.1 Playbook to Install Nginx Using the ubuntu User with Privilege Escalation**

Create the playbook first.yml on the controller node:

bash

Copy code

nano first.yml

Add the following content:

yaml

Copy code

- hosts: demo

user: ubuntu

become: yes

become\_user: root

connection: ssh

gather\_facts: true

tasks:

- name: Install nginx on both servers

apt:

name: nginx

state: present

- name: Create a directory as root on both servers

file:

state: directory

path: /root/ramantestdir

**Explanation:**

* **user: ubuntu**: Ansible will use the ubuntu user to connect to the managed nodes.
* **become: yes**: This flag tells Ansible to use privilege escalation.
* **become\_user: root**: The tasks will be run as the root user after privilege escalation.
* **apt**: Installs the Nginx package on both servers.
* **file**: Creates a directory (/root/ramantestdir) on the remote server.

**3.2 Run the Playbook:**

To execute the playbook, you need to supply the SSH password for the ubuntu user if necessary (since it's a non-passwordless setup).

bash

Copy code

ansible-playbook first.yml -k

The -k flag prompts for the SSH password of the ubuntu user.

**Part 4: Running Playbook as the raman User**

**4.1 Playbook to Install Nginx as the raman User**

Now, let’s configure Ansible to run the tasks as the raman user with privilege escalation. Update the playbook to look like this:

yaml

Copy code

- hosts: demo

user: raman

connection: ssh

gather\_facts: true

tasks:

- name: Install nginx on both servers

apt:

name: nginx

state: absent

**Explanation:**

* **user: raman**: The playbook will now run as the raman user.
* **apt**: The nginx package will be removed (state: absent) from both servers.

**4.2 Add raman to the Sudoers File**

Ensure that the raman user has sufficient privileges by adding the user to the sudoers file on each managed node.

bash

Copy code

vi /etc/sudoers

Add the following line:

bash

Copy code

raman ALL=(ALL) NOPASSWD: ALL

This line ensures that raman can run all commands as root without needing a password.

**4.3 Run the Playbook:**

To execute the playbook with the raman user:

bash

Copy code

ansible-playbook first.yml -k

You will be prompted to enter the SSH password for the raman user, after which Ansible will execute the playbook.

**Troubleshooting and Useful Commands**

* **Check Playbook Syntax**: Always check the syntax of your playbook before running it:

bash

Copy code

ansible-playbook --syntax-check first.yml

* **View Available Become Methods**: You can list all available privilege escalation methods by using:

bash

Copy code

ansible-doc -t become -l

* **View Collected Facts**: To view the facts collected from a node, use the setup module:

bash

Copy code

ansible demo -m setup

**Conclusion:**

In this lab, you have learned how to manage user accounts and configure privilege escalation using Ansible. You have created users, modified sudoers settings, and used different options to run tasks as various users with privilege escalation. You also learned how to run playbooks with specific users and perform administrative tasks securely using Ansible.

4o mini

**Ansible Lab Guide: Syntax Check, Dry Run, Passwordless Authentication for Standard User raman**

**Objective:**

In this lab, you will:

1. Perform syntax checks and dry runs on Ansible playbooks.
2. Set up passwordless SSH authentication for a standard user (raman).
3. Run an Ansible playbook to install software, manage files, and manage groups on remote servers.

**Prerequisites:**

1. **Ansible Installed**: Ensure that Ansible is installed on the controller node.

Check the installation:

bash

Copy code

ansible --version

1. **Managed Nodes**: Ensure you have one or more remote servers (e.g., 172.31.25.29, 172.31.24.46) where you want to perform tasks using Ansible.
2. **SSH Access**: Ensure that the raman user has SSH access to the managed nodes. This is required for running tasks via Ansible.

**Part 1: Syntax Check and Dry Run with Ansible Playbook**

**1.1 Syntax Check of Playbook**

Before running a playbook, it is a good practice to check its syntax to ensure there are no errors. To perform a syntax check on the playbook, use the --syntax-check flag:

bash

Copy code

ansible-playbook --syntax-check first.yml

If there are any syntax issues, Ansible will report them. If the syntax is correct, the command will return with no output.

**1.2 Dry Run of Playbook (Check Mode)**

The --check flag allows you to perform a dry run of the playbook. This simulates the execution of tasks without actually making any changes on the remote hosts:

bash

Copy code

ansible-playbook --check first.yml

This will show you which tasks would be executed without actually applying them.

**1.3 Running the Playbook**

To run the playbook and execute the tasks, use the following command:

bash

Copy code

ansible-playbook first.yml

This will execute the tasks specified in the first.yml playbook on the managed nodes.

**1.4 Step-by-Step Execution**

If you want to run the playbook one task at a time and review each step before proceeding, you can use the --step flag:

bash

Copy code

ansible-playbook first.yml --step

This will prompt you for each task, allowing you to decide whether to proceed with the execution of that task.

**Part 2: Setting Up Passwordless SSH Authentication for Standard User raman**

**2.1 Creating the raman User**

On the controller node, create a new user (raman) to be used for managing tasks. Use the adduser command for a more interactive setup or useradd for a quick addition.

bash

Copy code

sudo adduser raman

Set a password for the raman user (you can skip this step if using passwordless authentication directly).

**2.2 Generate SSH Key Pair**

To enable passwordless SSH login, generate an SSH key pair on the controller node using the ssh-keygen command:

bash

Copy code

ssh-keygen -t rsa

Press Enter to accept the default file location for the SSH key pair. You will be prompted to enter a passphrase, but you can leave it empty for passwordless authentication.

**2.3 Copy the Public Key to the Remote Hosts**

Use ssh-copy-id to copy the SSH public key to the raman user on the remote nodes (172.31.25.29 and 172.31.24.46 in this case):

bash

Copy code

ssh-copy-id raman@172.31.25.29

ssh-copy-id raman@172.31.24.46

This will add the public key to the remote raman user’s ~/.ssh/authorized\_keys file, allowing passwordless SSH login.

**2.4 Test SSH Connection**

Test the SSH connection to ensure passwordless authentication is working:

bash

Copy code

ssh raman@172.31.25.29

ssh raman@172.31.24.46

You should be able to log in without entering a password.

**Part 3: Running the Ansible Playbook**

**3.1 Example Playbook: First.yml**

Now, let’s create the Ansible playbook (first.yml) to install Nginx, create directories, and manage groups on the remote servers.

Create the first.yml file on the controller node:

bash

Copy code

nano first.yml

Add the following content to the playbook:

yaml

Copy code

- hosts: demo

user: raman

become: yes

connection: ssh

tasks:

- name: Install nginx on server

action: apt name=nginx state=present

- name: Add file directory

action: file path=/opt state=directory

- name: Group Creation

group:

name: grp1

gid: 5555

state: present

- name: Another Group Creation

group:

name: grp2

gid: 5656

state: present

**Explanation:**

* **user: raman**: This specifies that the tasks should be run as the raman user.
* **become: yes**: Enables privilege escalation (to root by default).
* **apt**: The task to install the nginx package on the remote server.
* **file**: The task to create a directory /opt on the remote server.
* **group**: These tasks create groups (grp1 and grp2) with specified GIDs (5555 and 5656).

**3.2 Run the Playbook**

After creating the playbook, run it using the following command:

bash

Copy code

ansible-playbook first.yml

This will execute the tasks and apply the changes on the managed nodes.

**3.3 Step-by-Step Execution (Optional)**

If you want to execute the playbook step by step, use the --step flag:

bash

Copy code

ansible-playbook first.yml --step

This allows you to approve each task before it is executed.

**Troubleshooting**

* **Syntax Errors**: If the syntax check (--syntax-check) fails, it will provide details about the errors. Correct the syntax in the playbook and try again.
* **Permissions Issues**: If raman does not have sufficient privileges, ensure that the user is added to the sudoers file with the appropriate permissions (e.g., raman ALL=(ALL) NOPASSWD: ALL).
* **SSH Connection Issues**: If passwordless SSH is not working, ensure the public key is copied correctly and that the ~/.ssh/authorized\_keys file has the correct permissions on the remote nodes.

**Conclusion:**

In this lab, you have:

1. Performed syntax checks and dry runs on an Ansible playbook to ensure correctness before execution.
2. Configured passwordless SSH authentication for the standard user raman to allow for seamless automation.
3. Ran an Ansible playbook that installs software, manages files, and creates groups on remote servers.

This workflow helps ensure that your automation tasks are executed correctly, securely, and efficiently.