RHCE: Red Hat Ansible Automation (RH294)

Quick Guide

Version 25.08

Ahmed Abdelwahed
ahmed@abdelwahed.me
LinkedIn

Contents

Ansible Architecture 4
Ansible Setup 5
1. Ansible Server Setup (`controller`) 5
Ansible Configuration 6
A. Configure ansible.cfg 6
B. Configure the Inventory (hosts file) 6
ad-hoc commands in Ansible 6
Shell Module 8
Raw Module 8
Execute a command using file module. 8
Execute a command using dnf module. 9
Execute a command using Service module. 9
Playbooks - Transitioning from Ansible Ad-Hoc Commands to Playbooks 10
Understanding YAML 10
Run Playbook 10
Validate Playbook 10
Working with facts 10
Variables 12
Using Conditional Tasks in Ansible Playbooks 14
Using Registers in Ansible for Effective Output Capture 15
using tags with playbooks 16
Using Blocks 17
Ansible Loops 18
Ansible Vault 18
Handlers 19
Jinja2 Templates 20
Ansible Plugins 21
Ansible Roles 21
Ansible Galaxy for Pre-Built Roles 23
File and Directory Management Automation with Ansible 24
Managing Packages with Ansible 26
Managing Users and Authentication 28
Scheduling with the at Module 30
Managing Services with the systemd Module 32
Managing Services with the service Module 32

Playbook Using the reboot Module 33
Managing Storage 33
Managing Network 35
Managing Windows 36

Ansible Architecture

1. Control Node

- The machine where Ansible is installed and from which all tasks and playbooks are executed.
- It contains the necessary configurations, modules, and plugins required to manage the target nodes.

2. Managed Nodes

- These are the target machines (servers, network devices, etc.) that Ansible manages.
- Managed nodes are usually not required to have Ansible installed; they only need SSH access (or WinRM for Windows systems).

3. Inventory

- The inventory is a file (by default located at `/etc/ansible/hosts`) that lists all the managed nodes (hosts) and groups.
- Can be a simple text file with IP addresses or hostnames, or a dynamic inventory script for more complex environments.

4. Facts

- System information automatically gathered by Ansible setup module from managed nodes at runtime. They provide detailed information about managed nodes, such as OS type, IP address, and hardware details.
- Useful for conditionals and managing different configurations based on system properties.

5. Modules

- Small programs that Ansible pushes out from the control node to the managed nodes.
- Ansible comes with numerous built-in modules for tasks such as installing packages, managing services, and handling files.

6. Playbooks

- Written in YAML, playbooks define the tasks and the desired end state of the managed nodes.
- They are a series of plays, where each play applies a group of tasks to a group of hosts.

7. Variables

- Variables are used to store values that can be reused throughout playbooks and roles.
- Can be defined in various places like playbooks, inventory, roles, or from external sources.

8. Templates

- Files that use **Jinja2** templating engine to generate dynamic content.
- Templates are rendered on the control node before being pushed to managed nodes.

9. Handlers

- Handlers are similar to tasks but are triggered by other tasks. They are used to perform actions like restarting a service only when a change is made. like restarting services or performing other actions that should only happen when changes occur.

10. Roles

- A way to organize playbooks and other files in a structured format.
- Roles can include tasks, handlers, variables, templates, and files, making them reusable and modular.

Ansible Setup

1. Ansible Server Setup (controller)

OS and Basic Configuration

- **Operating System**: Rockey Linux release 9.3
- **Server Name**: Execute `hostnamectl set-hostname controller` to set the server name to "controller".
- **IP Configuration**: Use the `nmcli` tool to configure the network settings.
    - Assign IP address, gateway, DNS, and set the connection method to manual:

```
nmcli connection modify "ens160" ipv4.addresses 192.168.244.100/24
ipv4.gateway 192.168.244.2 ipv4.dns 1.1.1.1 ipv4.method manual autoconnect yes
```

    - Bring the connection up: `nmcli connection up "ens160"`

- **Hosts File**: Add entries for node1 and node2 to `/etc/hosts`

```
10.0.0.4        controller
10.0.0.5        node1
10.0.0.6        node2
10.0.0.7        c22 #Windows Server 2022
```

User Creation and Configuration (on all nodes and controller)

Creating a User: `useradd -m ansibleadmin` and set the password using `passwd ahmed`.
Sudoers Configuration: Correctly edit the sudoers file to give ahmed sudo privileges without a password. Use visudo or `echo " ansibleadmin ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers` (but be very careful with direct echoes to /etc/sudoers as it can corrupt the file if done incorrectly).

SSH Configuration

- Generate SSH keys and copy them to node1 and node2 to enable passwordless SSH login.

```
su - ahmed
ssh-keygen -t rsa -b 2048 -N ""
ssh-copy-id ansibleadmin@node1
ssh-copy-id ansibleadmin@node2
```

Ansible Installation

- **Enable the Ansible repository** and install Ansible:

```
dnf repolist all | grep ansible
dnf config-manager --enable ansible-automation-platform-2.4-for-rhel-9-x86_64-
rpms (applicable for Redhat users)
dnf repolist enabled
dnf install -y ansible*
ansible –version
dnf install pip
sudo pip install --upgrade ansible
```

Node	Operating System	Server Name Command	IP Configuration & User Creation
Node1	Rocky Linux release 9.3	hostnamectl set-hostname node1	Follow similar steps as for the Ansible server, adjusting IP to **10.0.0.5**
Node2	Rocky Linux release 9.3	hostnamectl set-hostname node2	Similar to Node1, but with IP address set to **10.0.0.6**

Ansible Configuration

To complete the Ansible lab setup after having prepared the Ansible server and the nodes as you've described, the next steps involve configuring the `ansible.cfg` file and the inventory (`hosts`) file to ensure proper communication and management of your nodes from the Ansible controller. Let's outline these steps:

A. Configure ansible.cfg

The `ansible.cfg` file is the main configuration file for Ansible. It tells Ansible where to find the inventory, what user to run as, and various other settings. If there isn't an `ansible.cfg` in your working directory, Ansible will use the default one at `/etc/ansible/ansible.cfg`. To customize Ansible for your environment, you can create an `ansible.cfg` in your working directory or edit the global `/etc/ansible/ansible.cfg`.

```
sudo vim /etc/ansible/ansible.cfg
```

```
[defaults]
inventory = /etc/ansible/hosts      #point to your inventory file
remote_user = ahmed                 #Ansible uses this user for SSH connections
host_key_checking = False           #disable SSH host key checking in Ansible
```

Ansible follows a config file priority order:
$ANSIBLE_CONFIG → ./ansible.cfg → ~/.ansible.cfg → /etc/ansible/ansible.cfg.
So /etc/ansible/ansible.cfg is only used if none of the higher-priority files exist.

B. Configure the Inventory (hosts file)

The inventory file tells Ansible about the nodes it can manage. Since you already have node IP addresses and names, you'll add them to the inventory.

1. **Edit Inventory File**:
   `sudo cp -a /etc/ansible/hosts /etc/ansible/hosts_bk`
   Open the inventory file with a text editor:
   `sudo vim /etc/ansible/hosts`
2. **Add Nodes to Inventory**:
   Add your nodes under a group. For simplicity, let's call the group `nodes`. You can also specify ansible user (`ansible_user`) if it's different from the one Ansible runs under.

```
[nodes]
node1 ansible_host=10.0.0.5 ansible_user=ahmed
node2 ansible_host=10.0.0.6 ansible_user=ahmed
```

Ansible inventory file is resolved in this priority order:
1 -i option in the command → 2 inventory= in ansible.cfg → 3 /etc/ansible/hosts (default).
So /etc/ansible/hosts is only used if no other inventory is explicitly defined.

ad-hoc commands in Ansible

Ad-hoc commands in Ansible are one-liners used to perform quick tasks on managed nodes without writing a playbook. They leverage modules (pre-built tools) to execute actions like package management, file operations, or service control. and more. Here's how to execute ad-hoc commands in Ansible:

Syntax: **ansible [pattern] -i [inventory] -m [module] -a "[module arguments]" -u [user] -b**

- `[pattern]` specifies the hosts or group of hosts from your inventory on which you want to run the command.
- `-i [inventory]` (Optional) Specify a custom inventory file
- `-m [module]` it's like a tool that Ansible uses to perform a specific action on a target system.
- `-a "[module arguments]"` specifies the arguments to pass to the module.

- `-u [user]` specifies the user account to use for the connection.
- `-b` (Optional) Run the command with sudo privileges

List all available modules in Ansible

```
ansible-doc -l
ansible –help
pip install --upgrade ansible
ansible-galaxy collection install community.general ansible.posix --force
```

Module Naming: FQCN vs. Short Name

Ansible modules can be referenced using either their Fully Qualified Collection Name (FQCN) or their short name. For example, the `copy` module can be used as `ansible.builtin.copy` (FQCN) or simply copy (short name).

While both methods are valid, using the FQCN is generally recommended for writing playbooks to ensure clarity and avoid potential conflicts. For ad-hoc commands and quick tasks, the short name can be used for simplicity.

Ansible Configuration File Precedence

Ansible checks for ansible.cfg in the following order, using the first valid file it finds:

1. **Environment Variable**:

`ANSIBLE_CONFIG=/path/to/ansible.cfg`

- Set temporarily with export
- For a user: add to `~/.bashrc`
- For all users: add to `/etc/profile`

2. **Current Directory**: Looks for `./ansible.cfg` where the command is run.
3. **User Home**: Looks for `~/.ansible.cfg.`
4. **System-Wide**: Defaults to `/etc/ansible/ansible.cfg.`

If a higher-priority file is missing or invalid, Ansible moves to the next.

Ansible: Desired vs. Current State

- **Desired State**: The configuration you define in tasks.
- **Current State**: The actual state of the system.
- **How Ansible Works**: It compares both and changes only what's needed.
- **Idempotency**: Modules make changes **only once** — re-running has no effect if the system is already correct.
- **Applies To**: Both **ad-hoc commands** and **playbooks**, ensuring consistent, efficient automation.

Execute a command using command module **(For general command execution)**

```
ansible-doc command | grep -A 20 -i EXAMPLES
ansible node1 -m command -a "your_command_here"
ansible all -m ping
ansible all -m setup
ansible nodes -m reboot --become
ansible node1 -m command -a "uptime"
ansible node1 -m command -a "lastb" -b
ansible node1 -m command -a "df -h"
ansible node1 -m command -a "ls /tmp"
ansible node1 -m command -a "hostname"
ansible node1 -m command -a "getenforce"
ansible node1 -m command -a "uname -a"
ansible node1 -m command -a "ss -tuln"
```

Shell Module

Runs commands through the shell (/bin/sh) on the remote host — supports full shell features.

✅ Supports variables, pipes, redirection, &&, ||, etc.

⚠️ Less safe than command — risk of shell injection

Extracting Examples from ansible-doc shell

```
ansible-doc shell | grep -A 20 -i EXAMPLES
ansible all -m raw -a "python3 --version"
```

1. **List installed packages** `ansible all -m shell -a "dnf list installed| grep vim"`
2. **Run a Command with a Pipeline:**
   `ansible all -m shell -a "ps aux | grep httpd"`
3. **Redirect Output to a File:**
   `ansible all -m shell -a "ls -l /var/log > /tmp/logfiles.txt"`
4. **AND and OR logic used in the Linux shell,:**
   `ansible all -m shell -a "rpm -q nginx && echo 'Nginx is installed' || echo 'Nginx is NOT installed'"`
5. **Memory Usage Query Across All Hosts**
   `ansible all -a "free -m"`
6. **Extract Memory Information from** nodes
   `ansible nodes -m shell -a "cat /proc/meminfo | head -2`

Raw Module

- **Run low-level commands over SSH on remote hosts without requiring Python.**
- **Key Points**:
  - ✅ Ideal for bootstrapping minimal systems (no Python)
  - 🚫 Skips Ansible features (no vars, loops, or conditionals)
  - ⚠️ Limited output/error handling
  - 🧰 Executes commands as-is, like plain SSH

1. **Extracting Examples from ansible-doc raw**
   `ansible-doc raw | grep -A 20 -i EXAMPLES`
2. **Run a Simple Command:** `ansible all -m raw -a "echo Hello, World!"`
3. **Install Python on Red Hat-based Systems:**
   `ansible all -m raw -a "yum install -y python3"`
4. **Reboot a System:** `ansible all -m raw -a "reboot"`

The **command**, **shell** and **raw** modules in Ansible are **not idempotent** by default, meaning they execute commands without checking the current state of the system. To achieve idempotency with these modules, you must manually incorporate conditional logic within the commands. For inherently idempotent operations, it's recommended to use specific modules like file or service that ensure the desired state automatically.

Execute a command using file module.

```
ansible-doc file | grep -A 20 -i EXAMPLES
```

- **Create a Directory:** `ansible nodes -m file -a "path=/tmp/mydir state=directory"`
- **Create a file** `ansible node1 -m file -a "path=/tmp/emptyfile.txt state=touch"`
- **Check file property** `ansible node1 -m file -a "path=/tmp/emptyfile4.txt state=file" -v`
- **Delete a File:**`ansible nodes -m file -a "path=/tmp/myfile state=absent"`
- **Change Permissions:**
  `ansible nodes -m file -a " "path=/usr/local/bin/myscript.sh mode=0750 owner=admin group=devs""`
- **Create a Symbolic Link:**
  `ansible nodes -m file -a "src=/tmp/myfile dest=/tmp/mylink state=link"`

Execute a command using dnf module.

```
ansible-doc dnf | grep -A 20 -i EXAMPLES
```

1. **Install the httpd package:**
   ```
   ansible nodes -m dnf -a "name=httpd state=present"
   ```
2. **install multiple packages**
   ```
   ansible all -m dnf -a "name=vim,wget,curl,htop state=present" -b
   ```
3. **Remove the nginx package:**
   ```
   ansible nodes -m dnf -a "name=nginx state=absent"
   ```
4. **Package Installation or Update on nodes:**
   ```
   ansible nodes -m dnf -a "name=curl state=latest"
   ```
5. **Install a specific version of the git package:**
   ```
   ansible nodes -m dnf -a "name=git-2.18.1-3.el8 state=present"
   ```
6. **Allow Downgrade**
   ```
   ansible all -m dnf -a "name=vim-8.0.1500 state=present allow_downgrade=yes"
   ```
7. **Update all packages on the system:**
   ```
   ansible nodes -m dnf -a "name=* state=latest"
   ```

Execute a command using Service module.

```
ansible-doc service | grep -A 20 -i EXAMPLES
```

1. **Start the httpd service:** `ansible all -m service -a "name=httpd state=started"`
2. **Stop the nginx service:**
   ```
   ansible all -m service -a "name=nginx state=stopped"
   ```
3. **Restart the mysql service:**
   ```
   ansible all -m service -a "name=mysql state=restarted"
   ```
4. **Reload the sshd service:**
   ```
   ansible all -m service -a "name=sshd state=reloaded"
   ```
5. **Enable the httpd service to start on boot:**
   ```
   ansible all -m service -a "name=httpd enabled=yes"
   ```
6. **Disable the httpd service from starting on boot:**
   ```
   ansible all -m service -a "name=httpd enabled=no"
   ```

Playbooks - Transitioning from Ansible Ad-Hoc Commands to Playbooks

You've mastered using ad-hoc commands for single, non-repetitive tasks like checking server uptime or fetching system details. This chapter moves you into automating multiple tasks through Ansible playbooks, designed for complex operations and sequencing multiple actions across your infrastructure efficiently.

Understanding YAML

```
---                          # --- marks the beginning of a YAML document
# Block 1: Start of the YAML and Play Definition
# This defines the beginning of the playbook and what it will do
- name: Install the latest version of Apache     # Play name (high-level description of what this play does)
  hosts: n1                                       # Target host/group from inventory
  become: yes                                     # Use privilege escalation (e.g., sudo)
  become_user: root                               # The user to become (optional if root is default)
  gather_facts: yes                               # Collect system information (e.g., OS, IPs, memory)
  # Block 2: Tasks to execute on the target system
  tasks:                                          # Start of the tasks list
# Task 1: Install or update Apache
    - name: Ensure the latest version of Apache is installed   # Description of this specific task
      ansible.builtin.dnf:                        # Module name used to manage DNF packages on RHEL
        name: httpd                               # Parameter 'name' – package to install
        state: latest                             # Parameter 'state' – ensure the latest version is present
```

Run Playbook

To run the playbook `ansible-playbook gather_info.yml`

Validate Playbook

Validate YAML - Online YAML Tools

`ansible-playbook --check playbook1.yml` a dry run of your playbook before actually running the playbook.

`ansible-playbook --list-tasks playbook1.yml` list the tasks of each play in your playbook

Working with facts

Gathering facts (Collecting Facts)

Ansible automatically collects system information (facts) each time you execute a playbook, providing your playbook with the latest node information. This information can include system architecture, network interfaces, operating system, disk space, and more. Facts are useful for making decisions based on the specific characteristics of a host. You can modify this default setting to prevent facts from being gathered each run or to expedite playbook execution.

Turning Off Automatic Fact Collection `gather_facts: no`

display facts using ad-hoc `ansible node1 -m setup`

display facts using playbook:

```
---
- name: Gather and display facts
  hosts: all
  tasks:
    - name: Gather facts
      ansible.builtin.setup:

    - name: Display all facts
      ansible.builtin.debug:
        var: ansible_facts
```

OR

```
---
- name: Gather info about nodes
  hosts: node1
  gather_facts: yes
  tasks:
    - name: Display Gathered Info
      debug:
        msg:
          - "ServerName: {{ ansible_hostname }}"
          - "IPv4: {{ ansible_all_ipv4_addresses }}"
          - "Memory: {{ ansible_memtotal_mb }} MB"
          - "FreeSwap: {{ ansible_swapfree_mb }} MB"
          - "OS: {{ ansible_os_family }} {{ ansible_distribution_version }}"
          - "CPU: {{ ansible_processor }} MHz"
          - "Kernel: {{ ansible_kernel }}"
```

Handling Task Failures

- **Host-Specific Failure**: When a task fails on a specific host, Ansible stops executing the remaining tasks for that particular host.
- **Continued Execution on Other Hosts**: Ansible continues executing the remaining tasks on other hosts that did not encounter the failure.

Undoing Modifications:

- **No Easy Reversion**: There is no straightforward way to undo modifications made by a playbook due to complex dependencies.
- **Reverse Playbook**: To undo changes, write a playbook that performs the opposite actions in reverse order.

```
---
- name: install apache
  hosts: n1
  gather_facts: yes
  become: yes
  ignore_errors: yes
  tasks:
    - name: for testing
      ansible.builtin.dnf:
        name: httpiid
        state: latest
    - name: Display Info
      debug:
        msg: Installation is done
OR for specific task:
      ---
      - name: Test ignore_errors option
        hosts: node1
        gather_facts: no
        tasks:
          - name: This task will intentionally fail
            command: /bin/false
            ignore_errors: yes

          - name: This task will still run even after the failure
            debug:
              msg: "Previous task failed, but this one runs because we ignored the
      error."
```

Variables

Ansible variables allow you to manage dynamic values in your playbooks. They enable you to customize and control the behavior of your tasks, plays, and roles by providing a way to pass data into your automation.

- **Defining Variables in a Playbook**

```
---
- name: Install and start Apache HTTP server
  hosts: nodes
  become: yes

  vars:
    pkg_name: httpd
    pkg_state: present
    srv_name: httpd
    srv_state: started

  tasks:
    - name: Install Apache HTTP server
      ansible.builtin.dnf:
        name: "{{ pkg_name }}"
        state: "{{ pkg_state }}"

    - name: Start and enable Apache HTTP server
      ansible.builtin.service:
        name: "{{ srv_name }}"
        state: "{{ srv_state }}"
        enabled: yes
```

- **Using Variables from a Variable File**

Variables can be defined in a separate YAML file and included in the playbook using the `vars_files` keyword.

`vars.yml`

```
pkg_name: httpd
pkg_state: present
srv_name: httpd
srv_state: started
```

Playbooks with Variable File

```
---
- name: Install and start Apache HTTP server
  hosts: nodes
  become: yes

  vars_files:
    - vars.yml

  tasks:
    - name: Install Apache HTTP server
      ansible.builtin.dnf:
        name: "{{ pkg_name }}"
        state: "{{ pkg_state }}"

    - name: Start and enable Apache HTTP server
      ansible.builtin.service:
        name: "{{ srv_name }}"
        state: "{{ srv_state }}"
        enabled: yes
```

- **Passing Variables via Command Line**

You can also pass variables directly via the command line when running the playbook using the --extra-vars option.

```
ansible-playbook -i hosts install_httpd.yml --extra-vars "package_name=httpd service_name=httpd"
```

- **Playbook3: Example of Using Variables in Tasks:**

vim var-task.yml

```
- name: Install packages
  yum:
     name: "{{ packages }}"
     state: present
  vars:
   packages:
     - httpd
     - nginx
- name: Start nginx Service
  service:
    name=nginx
    state=started
    enabled=yes
```

Now import var-task.yml in 1st.yml playbook

```
---
- name: Create directory and file with specified permissions and owner
  hosts: all
  become: yes
  tasks:
    - name: Ensure directory /tmp/dir1 exists
      ansible.builtin.file:
        path: /tmp/dir1
        state: directory
        mode: '0777'
        owner: ahmed

    - name: Ensure file /tmp/dir1/file1 exists
      ansible.builtin.file:
        path: /tmp/dir1/file1
        state: touch
        mode: '0666'
        owner: ahmed
    - name: Import tasks from var-task.yml
      import_tasks: var-task.yml
```

Using Conditional Tasks in Ansible Playbooks

In Ansible, conditionals allow you to control the execution of tasks based on specific criteria, adding logic to your playbooks. This enables tasks to run only when certain conditions are met. Below are examples demonstrating how to use conditional tasks in Ansible.

Conditional Task Based on OS (Using Variables in Conditionals)

```
---
- name: Conditional package installation based on OS
  hosts: all
  tasks:
    - name: Install nginx on Debian/Ubuntu
      ansible.builtin.apt:
        name: nginx
        state: present
      when: ansible_os_family == "Debian"
    - name: Install nginx on RHEL/CentOS
      ansible.builtin.yum:
        name: nginx
        state: present
      when: ansible_os_family == "RedHat"
```

Using ansible_facts for Multiple conditions Execution

```
---
---
- name: Conditional tasks using facts
  hosts: all
  become: yes
  gather_facts: yes  # Explicitly enable since we're using facts

  tasks:
    - name: Install htop on RedHat node1 with sufficient memory
      ansible.builtin.dnf:
        name: htop
        state: present
      when:
        - ansible_hostname == "node1"
        - ansible_os_family == "RedHat"
        - ansible_memtotal_mb > 1024

    - name: Reboot if uptime exceeds 1 day
      ansible.builtin.reboot:
      when: ansible_uptime_seconds > 86400
```

Using Registers in Ansible for Effective Output Capture

In Ansible, registers are used to capture the output of a task, which can then be used in subsequent tasks or conditions. This can be particularly useful for debugging, **making decisions based on command output**, or dynamically modifying the playbook flow.

```
---
- name: Install and Start Apache if not installed
  hosts: all
  become: yes

  tasks:
    - name: Check if Apache is installed
      command: rpm -q httpd
      register: apache_status
      ignore_errors: yes

    - name: Show Apache check result
      debug:
        var: apache_status.rc

    - name: Install Apache if not present
      dnf:
        name: httpd
        state: present
      when: apache_status.rc != 0

    - name: Ensure Apache service is running
      service:
        name: httpd
        state: started
```

The when clause checks the `rc` (return code) of the previous command. If `apache_installed.rc != 0`, it means Apache is not installed, so this task will run.

List the server if CPU usage exceeds 50%.

```
---
- name: gather info from the Nodes
  hosts: all
  gather_facts: yes  #=setup
  tasks:
     - name: check CPU Utilization
       shell: top -bn1 | grep -i "%Cpu(s)" | awk '{print $2}'
       register: high_cpu

     - name: Display High_CPU Values
       debug:
         var: high_cpu.stdout

     - name: Display Nodes with High CPU utilization
       debug:
         var: ansible_hostname
       when: high_cpu.stdout | float > 50
```

Reboot the server if CPU usage exceeds 50%.

```
---
- name: gather info from the Nodes
  hosts: all
  gather_facts: yes  #=setup
  tasks:
     - name: check CPU Utilization
       shell: top -bn1 | grep -i "%Cpu(s)" | awk '{print $2}'
       register: high_cpu
     - name: Display High_CPU Values
       debug:
         var: high_cpu.stdout
     - name: Display Nodes with High CPU utilization
       debug:
         var: ansible_hostname
       when: high_cpu.stdout | float > 5
    - name: Restart the server if CPU usage is greater than 50%
      ansible.builtin.reboot:
      when: cpu_usage.stdout | float > 50
```

using tags with playbooks

Using tags in Ansible playbooks allows you to run specific tasks or groups of tasks selectively, without executing the entire playbook.

```
---
- name: Simple tagged playbook
  hosts: all
  become: yes
  tasks:
    - name: Update all packages
      dnf:
        name: "*"
        state: latest
      tags: update
    - name: Install nano text editor
      dnf:
        name: nano
        state: present
      tags: install
    - name: Create a welcome file
      copy:
        content: "Welcome to Ansible"
        dest: /tmp/welcome.txt
      tags: file
    - name: Remove httpd Package
      ansible.builtin.yum:
        name: httpd
        state: absent
      tags:
        - remove
        - cleanup
```

Run Only the remove Tasks `ansible-playbook -i hosts multipletags.yaml --tags remove`

Run Only the create Tasks `ansible-playbook -i hosts multipletags.yaml --tags create`

Run Tasks Tagged as create and import `ansible-playbook -i hosts multipletags.yaml --tags create,import`

Skip Tasks Tagged as cleanup

`ansible-playbook -i hosts multipletags.yaml --skip-tags cleanup`

Using Blocks

Using blocks in Ansible allows you to group related tasks together and manage their execution in a more structured manner. This includes handling exceptions and ensuring that certain tasks are always run, regardless of whether an error occurred.

```
---
- name: Apache Installation with Error Handling
  hosts: all
  become: yes

  tasks:
    - name: Main Apache Setup Block
      block:  # Contains a group of tasks to be executed.
        - name: Install Apache
          yum:
            name: httpd
            state: present

        - name: Start and enable Apache
          service:
            name: httpd
            state: started
            enabled: yes

        - name: Create Welcome Page
          copy:
            dest: /var/www/html/index.html
            content: "<h1>Welcome to Apache</h1>"

      rescue:  #Contains tasks to be executed if any task within the block fails.
        - name: Install tools for troubleshooting
          yum:
            name: httpd-tools
            state: present


        - name: Log the error
          debug:
            msg: "Setup failed. Troubleshooting tools installed."

      always:  # Contains tasks to be executed regardless of whether the tasks in the block or rescue sections succeeded or failed.
        - name: Ensure Apache service is enabled
          service:
            name: httpd
            enabled: yes
        - name: Confirm block execution
          debug:
            msg: "Finished running main block and handlers."
```

Ansible Loops

Ansible Loops let you repeat a task multiple times — like creating many users, installing several packages, or copying multiple files — without duplicating the task.

Default Loop Variable (item):

- **Usage:** By default, Ansible uses the variable `item` to reference each element in a loop.

Install multiple packages using a loop

```
---
- name: Install multiple packages using loop
  hosts: all
  become: yes
  vars:
    my_packages:
      - vim
      - curl
      - htop
      - tree
  tasks:
    - name: Install common packages
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ my_packages }}"
```

Ansible Vault

Ansible Vault provides encryption for sensitive data like passwords, API keys, and other secrets in your playbooks and variables.

1. Creating Encrypted Files

```
ansible-vault create secrets.yml   # Creates new encrypted file
ansible-vault encrypt existing.yml   # Encrypts existing file
```

2. Editing Encrypted Files

```
ansible-vault edit secrets.yml   # Edit encrypted file
ansible-vault view secrets.yml   # View without editing
```

3. Managing Encryption

```
ansible-vault decrypt secrets.yml   # Remove encryption
ansible-vault rekey secrets.yml   # Change password
```

Example Workflow

1. Create an Encrypted File

```
openssl passwd -1
ansible-vault create secret.yml
```

You will be prompted to enter and confirm a password. Then you can add your sensitive data to `secret.yml`:

```
users:
  - name: user10
    password: $1$Fqmw5PXx$OxiaFO1Q1MfrFuuv2UwM71
  - name: user11
    password: $1$Fqmw5PXx$OxiaFO1Q1MfrFuuv2UwM71
  - name: user12
    password: $1$Fqmw5PXx$OxiaFO1Q1MfrFuuv2UwM71
```

2. Use the Encrypted File in a Playbook

Create a playbook that uses the encrypted variables:

create-user.yml

```
---
- name: Gathering custom facts from the nodes
  hosts: all
  become: yes
  gather_facts: no
  ignore_errors: yes
  vars_files:
    - secret.yml
  tasks:
    - name: add Multiple Users
      ansible.builtin.user:
        name: "{{ item.name }}"
        password: "{{ item.password }}"
      loop: "{{ users }}"
```

3. Run the Playbook

```
ansible-playbook --ask-vault-pass create-user.yml
echo 'vault_password' > vault-pass
ansible-playbook --vault-password-file=vault-pass create-user.yml
```

Handlers

Handlers are special tasks that run at the end of a playbook execution when notified by other tasks. They're commonly used for service restarts and configuration reloads.

```
---
- name: Update web server and firewall
  hosts: webservers
  become: yes
  handlers:
    - name: Restart Apache
      ansible.builtin.service:
        name: httpd
        state: restarted
    - name: Reload Firewall
      ansible.builtin.service:
        name: firewalld
  tasks:
    - name: Install Apache
      ansible.builtin.yum:
        name: httpd
        state: present
      notify:
        - Reload Firewall
        - Restart Apache
    - name: Open HTTP port in firewall
      ansible.builtin.firewalld:
        port: 80/tcp
        permanent: true
        state: enabled
      notify: Reload Firewall
        state: reloaded
```

Jinja2 Templates

Jinja is a templating engine used by Ansible to dynamically generate files, configuration, or content based on variables.

Generate a Simple HTML File and Install Apache

- Jinja2 Template (`index.html.j2`)

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ header }}</h1>
    <p>{{ content }}</p>
</body>
</html>
```

create_html_and_install_httpd.yml

```
---
- name: installa nd config Aapache server
  hosts: node2
  become: yes
  vars:
    title: "Ahmed Web Portal"
    header: "Welcome to Ahmed Site"
    content: "This Portal for testing"
  tasks:
    - name: Install Apache
      ansible.builtin.dnf:
        name: httpd
        state: latest
    - name: Allow http from firewall
      ansible.posix.firewalld:
        port: 80/tcp
        permanent: true
        state: enabled
    - name: reload firewall
      service:
        name: firewalld
        state: reloaded
    - name: Ahmed site data
      ansible.builtin.template:
        src: index.html.j2
        dest: /var/www/html/index.html

    - name: starting http service
      ansible.builtin.systemd:
        name: httpd
        state: started
        enabled: true
```

Ansible Plugins

Ansible plugins are components that add specific features or extend Ansible's core functionality. They act like helpers that control how Ansible behaves, runs tasks, displays output, and interacts with other systems.

callback_plugins `ansible-doc -t callback -l | grep timer`

Set environment variable temporarily: `ANSIBLE_CALLBACKS_ENABLED=timer ansible-playbook site.yml`

Permanent: edit ansible.cfg

```
stdout_callback = yaml
callbacks_enabled = timer, profile_tasks
```

The profile_tasks callback plugin is a built-in Ansible feature that provides detailed timing information about task execution during playbook runs.

Ansible Roles

Roles in Ansible provide a structured, reusable, and modular way to organize playbooks. They help break down complex automation into manageable components.

Step-by-Step Guide to Setup Ansible Role and Playbook

Step 1: Create the Directory Structure

Ensure you have the following structure in your project directory:

```
roles/apache/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── site.yml
├── tasks
│   └── main.yml
├── templates
│   └── index.html.j2
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Step 2: Define the Role

1. **Create the Role using ansible-galaxy**:

   ```
   cd /home/ahmed/ansible-playbooks
         ansible-galaxy init roles/apache
   ```

2. **Define Default Variables**:

   ```
   vim roles/apache/vars/main.yml:
   welcome_message: "Welcome to Ansible"
   ```

3. **Create a Template**:

```
vim roles/apache/templates/index.html.j2
```

```
<html>
  <head>
    <title>Welcome</title>
  </head>
  <body>
    <h1>{{ welcome_message }}</h1>
  </body>
</html>
```

Define Tasks:

```
vim roles/apache/tasks/main.yml
```

```
---
- name: Install Apache
  dnf:
    name: httpd
    state: present
- name: Deploy index.html
  template:
    src: index.html.j2
    dest: /var/www/html/index.html
  notify: Restart Apache
- name: Start and enable Apache
  service:
    name: httpd
    state: started
    enabled: yes
```

Define Handlers:

```
vim roles/apache/handlers/main.yml
```

```
---
- name: Restart Apache
  service:
    name: httpd
    state: restarted
```

Step 3: Create the Inventory File

Create the inventory file at the root level of your project directory: `vim ansible-playbook/inventory`

```
# inventory
[webservers]
node1 ansible_host=10.0.0.5
node2 ansible_host=10.0.0.6
```

Step 4: Create the Playbook File

Create the site.yml playbook at the root level of your project directory: `vim ansible-playbook/site.yml`

```
# site.yml
---
- hosts: webservers
  become: yes
  roles:
    - role: apache
      vars:
        welcome_message: "Hello, World!"
```

Step 5: Run the Playbook

Navigate to your project directory and run the playbook using the inventory file:

```
ansible-playbook -i apache/tests/inventory site.yml
```

Ansible Galaxy for Pre-Built Roles

Using Ansible Galaxy for ready-made roles is a great way to leverage existing work and save time when configuring your infrastructure. Ansible Galaxy is a hub for finding, sharing, and downloading roles developed by the Ansible community.

Step 1: Install a Role from Ansible Galaxy

1. **Search for a Role**:
    - Go to Ansible Galaxy and search for the role you need. For example, to find a role for Apache, search for "**nginx**".

```
ansible-galaxy search nginx
```

```
ansible-galaxy install geerlingguy.nginx
```

Step 2: Use the Installed Role in Your Playbook

1. **Create a Playbook**:
    - Create a playbook that uses the downloaded role. Here is an example playbook that uses the geerlingguy.apache role:

```
# site.yml
```

```
# install-nginx.yml
- name: Install and configure NGINX using Ansible Galaxy role
  hosts: nodes
  become: yes
  roles:
    - role: geerlingguy.nginx
```

Step 3: Run the Playbook

1. **Run the Playbook**:
    - Execute the playbook using the inventory file:

```
ansible-playbook site.yml
```

Note the conditions in /home/ansibleadmin/.ansible/roles/geerlingguy.nginx/tasks/main.yml

File and Directory Management Automation with Ansible

```
---
- name: Manage files and directories
  hosts: all
  become: yes

  tasks:
    - name: Create a directory
      ansible.builtin.file:
        path: /tmp/my_directory
        state: directory
        mode: '0755'

    - name: Create an empty file
      ansible.builtin.file:
        path: /tmp/my_directory/my_file.txt
        state: touch
        mode: '0644'

    - name: Copy a file from the control node to the managed node
      ansible.builtin.copy:
        src: ../files/sample_file.txt
        dest: /tmp/my_directory/sample_file.txt
        mode: '0644'

    - name: Set permissions on the directory
      ansible.builtin.file:
        path: /tmp/my_directory
        mode: '0755'
        recurse: yes

    - name: Set permissions on the file
      ansible.builtin.file:
        path: /tmp/my_directory/my_file.txt
        mode: '0644'

- name: Archive the directory without compression
      ansible.builtin.archive:
        path: /tmp/my_directory
        dest: /tmp/my_directory.tar
        format: tar

- name: Archive and compress the directory with gzip
      ansible.builtin.archive:
        path: /tmp/my_directory
        dest: /tmp/my_directory.tar.gz
        format: gz

    - name: Archive and compress the directory with bzip2
      ansible.builtin.archive:
        path: /tmp/my_directory
        dest: /tmp/my_directory.tar.bz2
        format: bz2

    - name: Archive and compress the directory with xz
      ansible.builtin.archive:
        path: /tmp/my_directory
        dest: /tmp/my_directory.tar.xz
        format: xz
```

Editing files using lineinfile

1. **Ensuring a Line is Present in a File**

```
---
- name: Ensure a line is present in a configuration file
  hosts: all
  tasks:
    - name: Ensure a specific line is present in /etc/sysctl.conf
      ansible.builtin.lineinfile:
        path: /etc/sysctl.conf
        line: "net.ipv4.ip_forward = 1"
        state: present
```

2. **Removing a Line from a File**

```
---
- name: Remove a line from a configuration file
  hosts: all
  tasks:
    - name: Remove a specific line from /etc/sysctl.conf
      ansible.builtin.lineinfile:
        path: /etc/sysctl.conf
        line: "net.ipv4.ip_forward = 1"
        state: absent
```

3. **Replacing a Line Based on a Regular Expression**

```
---
- name: Replace a line matching a regular expression
  hosts: all
  tasks:
    - name: Ensure the correct setting is present in /etc/sysctl.conf
      ansible.builtin.lineinfile:
        path: /etc/sysctl.conf
        regexp: '^net.ipv4.ip_forward'
        line: 'net.ipv4.ip_forward = 1'
```

4. **Inserting a Line After a Specific Pattern**

```
---
- name: Insert a line after a specific pattern
  hosts: all
  tasks:
    - name: Insert a line after a specific line in /etc/sysctl.conf
      ansible.builtin.lineinfile:
        path: /etc/sysctl.conf
        line: "net.ipv4.tcp_syncookies = 1"
        insertafter: "^net.ipv4.ip_forward = 1"
```

Managing Packages with Ansible

Basic Package Installation

```
- name: Install a single package
  hosts: all
  become: yes
  tasks:
    - name: Install vim
      yum:
        name: vim
        state: present
```

Installing a Specific Version of a Package

```
- name: Install a specific version of a package
  hosts: all
  become: yes
  tasks:
    - name: Install specific version of nginx
      yum:
        name: nginx-1.16.1
        state: present
```

Removing a Package

```
- name: Remove a package
  hosts: all
  become: yes
  tasks:
    - name: Remove httpd
      yum:
        name: httpd
        state: absent
```

Installing Multiple Packages

```
- name: Install multiple packages
  hosts: all
  become: yes
  tasks:
    - name: Install vim, git, and wget
      yum:
        name:
          - vim
          - git
          - wget
        state: present
```

Only Update some packages if it only installed

```
- name: Update the webserver, depending on which is installed on the system.
Do not install the other one
  ansible.builtin.dnf:
    name:
      - httpd
      - nginx
    state: latest
    update_only: yes
```

Updating All Packages

```
- name: Update all packages to the latest version
  hosts: all
  become: yes
  tasks:
    - name: Update all packages
      yum:
        name: '*'
        state: latest
```

Install nginx rpm from a local file

```
- name: Install nginx rpm from a local file
  ansible.builtin.dnf:
    name: /usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm
    state: present
```

Installing a Package from a Specific Repository

```
- name: Install a package from a specific repository
  hosts: all
  become: yes
  tasks:
    - name: Add EPEL repository
      yum_repository:
        name: epel
        description: "Extra Packages for Enterprise Linux"
        baseurl:
"https://dl.fedoraproject.org/pub/epel/$releasever/$basearch/"
        gpgcheck: yes
        enabled: yes
    - name: Install htop from EPEL repository
      yum:
        name: htop
        enablerepo: epel
        state: present
```

Ensuring Packages are the Latest Version

```
- name: Ensure specific packages are the latest version
  hosts: all
  become: yes
  vars:
    latest_packages:
      - vim
      - git
  tasks:
    - name: Ensure packages are the latest version
      yum:
        name: "{{ latest_packages }}"
        state: latest
```

Comprehensive Package Management Playbook

```
- name: Comprehensive package management
  hosts: all
  become: yes
  vars:
    install_packages:
      - vim
```

```
    - git
  remove_packages:
    - httpd
tasks:
  - name: Ensure packages are installed
    yum:
      name: "{{ install_packages }}"
      state: present
  - name: Ensure packages are removed
    yum:
      name: "{{ remove_packages }}"
      state: absent
  - name: Update all packages to the latest version
    yum:
      name: '*'
      state: latest
```

Managing Users and Authentication

Adding Users

```
---
- name: Create a user
  hosts: all
  become: yes
  tasks:
    - name: Ensure the user 'Ahmed' is present
      user:
        name: testuser
        state: present
        comment: "Ahmed Mahmoud"
        uid: 1001
        shell: /bin/bash
```

Removing Users

```
---
- name: Remove a user
  hosts: all
  become: yes
  tasks:
    - name: Ensure the user 'ahmed' is absent
      user:
        name: ahmed
        state: absent
```

Managing Groups

```
---
- name: Create a group
  hosts: all
  become: yes
  tasks:
    - name: Ensure the group 'admin' is present
      group:
        name: admin
        state: present
```

Adding Users to Groups

```
---
- name: Add a user to a group
```

```
  hosts: all
  become: yes
  tasks:
    - name: Ensure the user 'john' is added to the 'admin' group
      user:
        name: testuser
        groups: admin
        append: yes
```

Setting User Passwords

You can set user passwords by encrypting the password using OpenSSL or Python's crypt library. Here's an example using OpenSSL:

```
openssl passwd -1 "333"
```

This command will return an encrypted password string, which you can use in your Ansible playbook.

Comprehensive User Management Playbook

```
---
- name: Create user Ahmed with specific options
  hosts: all
    gather_facts: no
  tasks:
    - name: Ensure the group 'user1' exists
      group:
        name: user1
        state: present

    - name: Ensure the groups 'sales' and 'hr' exist
      group:
        name: "{{ item }}"
        state: present
      loop:
        - sales
        - hr

    - name: Create user ahmed with specific options
      user:
        name: testuser2
        comment: "test account"
        uid: 5002
        groups: "sales,hr"
        group: user1
        shell: /bin/sh
        create_home: no
        system: no
        password: "$1$DKur3LQP$m1anE4NO5FLlxhxE9ZLBf/"
```

Verify User Existence

```
ansible node2 -m command -a "id testuser2" -b
node2 -m command -a "tail -3 /etc/group" -b
ansible node2 -m command -a "tail -3 /etc/passwd" -b
ansible node2 -m command -a "tail -3 /etc/shadow" -b
```

Scheduling with the at Module

```
ansible-doc at | grep -A 20 -i EXAMPLES
ansible-doc cron | grep -A 20 -i EXAMPLES
```

Playbook with Verification Tasks

```
---
- name: Schedule a task with the at module and verify
  hosts: n1
  become: yes
  tasks:
    - name: Install the at package
      ansible.builtin.yum:
        name: at
        state: present

    - name: Ensure the atd service is running and enabled
      ansible.builtin.service:
        name: atd
        state: started
        enabled: yes
    - name: Schedule a one-time task to run a script in 5 minutes
      at:
        command: /usr/local/bin/backup.sh
        count: 5
        units: minutes
        unique: yes

    - name: Schedule a one-time task to restart a service at a specific time
      at:
        command: systemctl restart httpd
        count: 1
        units: days
        unique: yes
```

Verification and eliminate

```
ansible all -m command -a "atq" -b
ansible n1 -m shell -a "atrm" -b
```

Cron Job for Backup

```
---
- name: Schedule multiple cron jobs with commands and delete them
  hosts: n1
  become: yes
  tasks:
    - name: Ensure the cron package is installed
      ansible.builtin.yum:
        name: cronie
        state: present
    - name: Ensure the cron service is running and enabled
      ansible.builtin.service:
        name: crond
        state: started
        enabled: yes
    - name: Schedule a weekly cron job to echo a message
      ansible.builtin.cron:
```

```
            name: "weekly_backup_command"
            minute: "0"
            hour: "2"
            day: "*"
            month: "*"
            weekday: "0"  # Sunday
            job: "echo 'Weekly backup performed' > /tmp/weekly_backup.log"
            user: root
            state: present
        - name: Schedule a daily cron job to echo a message
          ansible.builtin.cron:
            name: "daily_cleanup_command"
            minute: "0"
            hour: "3"
            day: "*"
            month: "*"
            weekday: "*"
            job: "echo 'Daily cleanup performed' > /tmp/daily_cleanup.log"
            user: root
            state: present
        - name: Schedule a monthly cron job to echo a message
          ansible.builtin.cron:
            name: "monthly_report_command"
            minute: "0"
            hour: "4"
            day: "1"   # First day of the month
            month: "*"
            weekday: "*"
            job: "echo 'Monthly report generated' > /tmp/monthly_report.log"
            user: root
            state: present

        - name: Schedule a yearly cron job to echo a message
          ansible.builtin.cron:
            name: "yearly_maintenance_command"
            minute: "0"
            hour: "5"
            day: "1"   # First day of the month
            month: "1" # January
            weekday: "*"
            job: "echo 'Yearly maintenance performed' >
    /tmp/yearly_maintenance.log"
            user: root
            state: present
```

Verification and eliminate

```
ansible all -m command -a "crontab -l" -b
ansible n1 -m shell -a "crontab -r -u root" -b
```

Managing Services with the systemd Module

```
---
- name: Manage services with systemd
  hosts: all
  become: yes
  tasks:
    - name: Ensure the httpd service is started and enabled at boot
      systemd:
        name: httpd
        state: started
        enabled: yes

    - name: Ensure the firewalld service is stopped, masked and disabled at
boot
      systemd:
        name: firewalld
        state: stopped
        enabled: no
 masked: yes

    - name: Restart the nginx service
      systemd:
        name: nginx
        state: restarted

    - name: Reload the sshd service
      systemd:
        name: sshd
        state: reloaded
```

Managing Services with the service Module

```
---
- name: Manage services with the service module
  hosts: all
  become: yes
  tasks:
    - name: Ensure the httpd service is started and enabled at boot
      service:
        name: httpd
        state: started
        enabled: yes
    - name: Ensure the firewalld service is stopped and disabled at boot
      service:
        name: firewalld
        state: stopped
        enabled: no
    - name: Restart the nginx service
      service:
        name: nginx
        state: restarted

    - name: Reload the sshd service
      service:
        name: sshd
        state: reloaded
```

Playbook Using the reboot Module

Here's an example playbook that demonstrates how to use the reboot module to reboot a remote machine and wait for it to become available again.

```
---
- name: Reboot the server
  hosts: all
  become: yes
  tasks:
    - name: Ensure all updates are applied
      yum:
        name: '*'
        state: latest

    - name: Reboot the server
      reboot:
        reboot_timeout: 600
        test_command: whoami

    - name: Verify the server is back online
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 22
        delay: 10
        timeout: 300
        state: started
```

Managing Storage

```
ansible n1 -m command -a "lsblk"
```

```
---
- name: lvm lab
  hosts: n2
  become: yes
  gather_facts: no
  ignore_errors: yes
  tasks:
    - name: create sda partition
      community.general.parted:
        device: /dev/sda
        number: 1
        state: present
        part_start: 0%
        part_end: 100%
    - name: create sdb partition
      community.general.parted:
        device: /dev/sdb
        number: 1
        state: present
        part_start: 0%
        part_end: 100%
    - name:  create physical volume
      community.general.lvg:
        vg: vg1
```

```
      pvs:
        - /dev/sda1
        - /dev/sdb1
    - name: create logical volume lv1
      community.general.lvol:
        vg: vg1
        lv: lv1
        size: 100%FREE
    - name: format the lv1 xfs
      community.general.filesystem:
        fstype: xfs
        dev: /dev/vg1/lv1
    - name: create a mount point for lv1
      ansible.builtin.file:
        path: /mnt/lv1
        state: directory
    - name: mount lv1 to /mnt/lv1
      ansible.posix.mount:
        path: /mnt/lv1
        src: /dev/vg1/lv1
        fstype: xfs
        opts: defaults
        state: mounted
    - name: mount lv1 to /mnt/lv1
      ansible.posix.mount:
        path: /mnt/lv1
        src: /dev/vg1/lv1
        fstype: xfs
        opts: defaults
        state: present
```

```
ansible node2 -m shell -a "lsblk | grep sdc;tail -1 /etc/fstab;df -h /mnt/lv1disk"
```

Extend Logical Volume and Resize Filesystem by adding a new sde disk

```
---
- name: Extend Logical Volume and Resize Filesystem
  hosts: n2
  become: yes
  tasks:
    - name: Create partition sde1 with 20GB
      community.general.parted:
        device: /dev/sde
        number: 1
        state: present
        part_start: 0%
        part_end: 20GB
    - name: Create physical volume on /dev/sde1
      ansible.builtin.command: pvcreate /dev/sde1
    - name: Extend the volume group to include /dev/sde1
      ansible.builtin.command: vgextend vg1 /dev/sde1
    - name: Extend the logical volume lv1
      ansible.builtin.command: lvextend -L +20G /dev/vg1/lv1
    - name: Resize filesystem on lv1
      ansible.builtin.command: xfs_growfs /dev/vg1/lv1
```

Managing Network

```
ansible-galaxy collection list
ansible-galaxy collection install fedora.linux_system_roles
ansible-galaxy collection install ansible.network
---
- name: adding some data in hots file
  hosts: n5
  become: yes
  vars:
    new_hostname: n5
    new_ip: 192.168.112.55
    mask: 24
    gateway: 192.168.112.2
    dns: 1.1.1.1
    nic: ens160
    hosts_ip: 192.168.112.55
    hosts_domain: n5
    port: 9090/tcp

  tasks:
    - name: change net config
      community.general.nmcli:
        conn_name: "{{nic}}"
        ifname: "{{nic}}"
        type: ethernet
        ip4: "{{new_ip}}/{{mask}}"
        gw4: "{{gateway}}"
        dns4: "{{dns}}"
        state: present
        autoconnect: yes
    - name: changing the host name
      ansible.builtin.hostname:
        name: "{{new_hostname}}"

    - name: adding new record in hosts
      ansible.builtin.lineinfile:
        path: /etc/hosts
        line: "{{hosts_ip}} {{hosts_domain}}"
        state: present

    - name: allow port 9090 from filrewall
      ansible.builtin.firewalld:
        port: "{{port}}"
        permanent: yes
        state: enabled
    - name: restart the interface
      ansible.builtin.shell: nmcli conn up "{{nic}}"
    - name: reload firewall
      ansible.builtin.shell: firewall-cmd --reload
```

Managing Windows

Connect Windows Server to ansible

1. **Open PowerShell as Administrator on the Windows Server**

```
winrm quickconfig -force
winrm set winrm/config/service/auth '@{Basic="true"}'
winrm set winrm/config/service '@{AllowUnencrypted="true"}'
winrm set winrm/config/listener?Address=*+Transport=HTTP '@{Port="5985"}'
Enable-PSRemoting -Force
Restart-Service winrm
```

2. **Ansible Inventory Update**

```
[windows]
c22 ansible_host=192.168.244.194

[windows:vars]
ansible_user=Administrator
ansible_password=YourPassword
ansible_connection=winrm
ansible_port=5985
ansible_winrm_transport=basic
ansible_winrm_server_cert_validation=ignore
```

3. **Python packages and ansible windows collections**

```
pip install pywinrm
# Install specific compatible versions
ansible-galaxy collection install microsoft.ad:1.2.0
ansible-galaxy collection install ansible.windows:2.0.0
ansible-galaxy collection install community.windows:3.0.0
```

4. **Test Connectivity**

```
ansible c22 -m win_ping
ansible c22 -m setup
ansible c22 -m win_command -a "systeminfo"
```

Using ad-hoc command

```
ansible c22 -m win_command -a "systeminfo"
ansible c22 -m win_shell -a "Get-HotFix"
ansible c22 -m win_shell -a "Get-PSDrive -PSProvider FileSystem"
ansible c22 -m win_shell -a "Get-WmiObject -Class Win32_Product"
ansible c22 -m win_shell -a "Get-ADUser -Filter * -Property Name,
EmailAddress"
```

Useful Modules

```
win_command
win_shell
win_service
win_feature
win_copy
win_reboot
win_user
microsoft.ad.user
microsoft.ad.domain
ansible-doc microsoft.ad.membership | grep -iA 25 example
```

Managing users, groups and services

```
---
- name: Manage Users and Computers on Windows Server 2019
  hosts: c22
  tasks:
    - name: Create a new local user
      ansible.windows.win_user:
        name: ahmed
        password: "Password123!"
        groups:
          - Administrators
        state: present
        password_never_expires: yes
        update_password: always

    - name: Ensure the user directory exists
      ansible.windows.win_file:
        path: C:\Users\ahmed
        state: directory
    - name: Create a directory for the user
      ansible.windows.win_file:
        path: C:\Users\ahmed\Documents\Ansible
        state: directory
    - name: Add user to Remote Desktop Users group
      ansible.windows.win_group_membership:
        name: Remote Desktop Users
        members:
          - ahmed
        state: present
    - name: Start and enable a service (e.g., Spooler)
      ansible.windows.win_service:
        name: Spooler
        start_mode: disabled
        state: stopped
    - name: Add entry to hosts file
      ansible.windows.win_shell:
        cmd: "Add-Content -Path 'C:\\Windows\\System32\\drivers\\etc\\hosts' -
Value '11.22.33.44 www.abdelwahed.me'"
        args:
          executable: powershell.exe
```

Install and configure Active Directory and DNS

```
ansible-galaxy collection install microsoft.ad
```

```
---
- name: Install and Configure Active Directory Domain
  hosts: c22
  tasks:
    - name: Install AD Domain Services feature
      ansible.windows.win_feature:
        name: AD-Domain-Services
        state: present
    - name: Include AD DS Tools
```

```
    ansible.windows.win_feature:
      name: RSAT-ADDS
      state: present
  - name: Install DNS Server feature
    ansible.windows.win_feature:
      name: DNS
      state: present

  - name: Promote this server to a domain controller
    microsoft.ad.domain:
      dns_domain_name: abdelwahed.me
      domain_netbios_name: ABDELWAHED
      safe_mode_password: "Password123!"  # Use a secure password here
      domain_mode: 2012R2  # Example: Specify the domain functional level
      forest_mode: 2012R2
      log_path: "C:\\Windows\\NTDS"
      sysvol_path: "C:\\Windows\\SYSVOL"
      database_path: "C:\\Windows\\NTDS"
      install_dns: yes
      create_dns_delegation: no
      reboot: yes
    register: promote_domain

  - name: Reboot the server to complete the promotion
    ansible.windows.win_reboot:
      when: promote_domain.changed
```

Create AD User

```
---
- name: Manage AD User
  hosts: windows
  tasks:
    - name: Ensure user bob is present with address information
      microsoft.ad.user:
        name: bob
        firstname: Bob
        surname: Smith
        company: BobCo
        password: B0bP4ssw0rd
        state: present
        groups:
          set:
            - Domain Admins
```