Ansible

Inventory   
  
An **Ansible inventory** is a file that contains information about the hosts you want to manage. In a **single-line format**, you can specify multiple hosts, their properties, and configurations.

By default, the inventory file is located at /etc/ansible/hosts.

Test the connectivity with a simple ping command:

ansible all -m ping

**Troubleshooting**

* **If you encounter any issues, ensure the ssh service is running on the managed nodes and that you have proper permissions.**
* **Use -vvvv (verbose mode) with Ansible commands to debug**

**ansible all -m ping -vvvv**

**Create the Inventory File**

**nano inventory.yml**

**Structure of the YAML Inventory File**

**all:**

**children:**

**web\_servers:**

**hosts:**

**web1.example.com:**

**ansible\_host: 192.168.1.10**

**ansible\_user: ubuntu**

**ansible\_ssh\_private\_key\_file: /home/your\_user/.ssh/id\_rsa**

**web2.example.com:**

**ansible\_host: 192.168.1.11**

**ansible\_user: ubuntu**

**ansible\_ssh\_private\_key\_file: /home/your\_user/.ssh/id\_rsa**

**db\_servers:**

**hosts:**

**db1.example.com:**

**ansible\_host: 192.168.1.20**

**ansible\_user: ubuntu**

**ansible\_port: 2222**

**ansible\_ssh\_private\_key\_file: /home/your\_user/.ssh/id\_rsa**

**vars:**

**db\_type: postgres**

**db\_port: 5432**

**vars:**

**ansible\_ssh\_private\_key\_file: /home/your\_user/.ssh/id\_rsa**

**Key Elements Explained**

1. **Group Organization:**
   * **web\_servers and db\_servers are groups of hosts.**
   * **The children section organizes these groups under the all group.**
2. **Host Definitions:**
   * **Each host can have specific variables like ansible\_host, ansible\_user, and ansible\_port.**
3. **Group Variables:**
   * **Defined under the vars section of a group (e.g., db\_type and db\_port for db\_servers).**
4. **Global Variables:**
   * **Defined under the vars section of all, applicable to all hosts and groups.**

**Notes:**

**ansible\_ssh\_private\_key\_file: Path to the private key file used for authentication.**

**Ensure the file ansable.pem has the correct permissions:**

**chmod 600 /home/ubuntu/ansabil/ansable.pem**

**Test the Inventory File: Validate the syntax of your inventory.yaml file with:**

**ansible-inventory -i inventory.yaml --list**

**Test the Inventory**

**ansible all -i inventory.yml -m ping**

**The -i option specifies the custom inventory file.**

**The -m ping module checks connectivity to the hosts.**

**Key Points in the Output**

1. **Host Authenticity Check:**
   * **When connecting to a host for the first time, SSH asks you to confirm its authenticity. You entered yes, which added the host's key to your ~/.ssh/known\_hosts file. This step won't repeat unless the host's key changes.**
2. **Python Interpreter Warning:**
   * **Ansible found Python on the remote host at /usr/bin/python3.12 and is using it.**
   * **The warning is informational and typically doesn't require action unless you plan to change the Python version on the remote host. You can suppress this warning by explicitly specifying the Python interpreter in your inventory file:**

**Solution:**

**Suppress Python Warnings (Optional): If you'd like to suppress the Python interpreter warning, add the following to your inventory.yaml file under the host definition:**

**ansible\_python\_interpreter: /usr/bin/python3**

**Ad-Hoc Command**

**An Ansible ad-hoc command allows you to execute tasks directly on remote machines without the need to write a full playbook. The command is typically run from the command line and allows you to quickly perform tasks like installing software, running commands, or gathering facts from remote machines.**

**1.Install nginx**

**Use Ansible’s apt module (for Debian/Ubuntu-based systems) to install Nginx.**

**ansible all -i inventory.yml -m apt -a "name=nginx state=present" -b**

**Explanation:**

* **all: Target all hosts in the inventory.**
* **-i inventory.yml: Specifies the inventory file.**
* **-m apt: Uses the apt module (for Ubuntu/Debian).**
* **-a "name=nginx state=present": Installs the nginx package.**
* **-b: Run the task as a sudo user (because installation requires elevated privileges).**

**2. Start Nginx Service**

**To start the Nginx service, use the service module.**

**ansible all -i inventory.yml -m service -a "name=nginx state=started" -b**

**Explanation:**

* **-m service: Uses the service module.**
* **-a "name=nginx state=started": Starts the Nginx service.**

**3.Stop Nginx Service**

**Similarly, you can stop the Nginx service using the same service module.**

**ansible all -i inventory.yml -m service -a "name=nginx state=stopped" -b**

**Explanation:**

* **-a "name=nginx state=stopped": Stops the Nginx service.**

**4. Copy a Sample index.html File**

**To copy a sample index.html file to the Nginx web root, use the copy module.**

**ansible all -i inventory.yml -m copy -a "src=/path/to/local/index.html dest=/var/www/html/index.html" -b**

**Explanation:**

* **-m copy: Uses the copy module to transfer files.**
* **-a "src=/path/to/local/index.html dest=/var/www/html/index.html":**
  + **src: Path to the local file you want to copy.**
  + **dest: Destination on the remote server.**
* **-b: Run the task as a sudo user because Nginx's web root (/var/www/html) may require elevated privileges.**

**Ansible playbook**

**An Ansible playbook is a YAML file that describes a series of tasks to be executed on remote hosts. A single plain playbook**

**Sample playbook:**

**---**

**- name: Uninstall Nginx, Install Apache, and MySQL**

**hosts: webservers**

**become: yes**

**tasks:**

**- name: Uninstall nginx**

**apt:**

**name: nginx**

**state: absent # This ensures that Nginx is uninstalled**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present # This ensures that Apache is installed**

**- name: Ensure Apache is running**

**service:**

**name: apache2**

**state: started # Ensure Apache is running**

**enabled: yes # Ensure Apache starts on boot**

**- name: Install MySQL Server**

**apt:**

**name: mysql-server**

**state: present # This ensures that MySQL Server is installed**

**- name: Ensure MySQL is running**

**service:**

**name: mysql**

**state: started # Ensure MySQL is running**

**enabled: yes # Ensure MySQL starts on boot**

### Key Components of an Ansible Playbook

1. **---: This marks the beginning of a YAML document (it is optional in playbooks but is often included).**
2. **name: (Optional) This is a descriptive name of the play, useful for logging and debugging. It doesn't affect functionality.**
3. **hosts: This specifies the target group of hosts on which to run the tasks. This could be a group defined in your inventory file (e.g., webservers), or it could be specific host names or IP addresses.**
4. **become: This specifies whether to elevate the privilege for running the tasks (similar to sudo in Linux).**
   * **become: yes means the tasks should be run with elevated privileges (like root).**
   * **become: no means no elevation (default).**
5. **vars: (Optional) This allows you to define variables that can be used within the playbook. These variables can be referenced later within tasks.**
6. **tasks: This is a list of tasks that will be executed on the target hosts. Each task typically calls a module that performs a specific action on the target system (such as installing software, copying files, or managing services).**

**Run cmd:**

**ansible-playbook -i hosts.ini file.yml**

**Ad hoc:**

**ansible all -i inventory.yml -m copy -a "src=index.html desc=/var/www/html/" -b**

**Ansible run cmds:**

**basic syntax:**

**ansible-playbook site.yml**

**Additional Options**

1. **Running a Playbook with Inventory File**

**By default, Ansible uses /etc/ansible/hosts for inventory, but you can specify a custom inventory file using the -i flag:**

**ansible-playbook -i /path/to/your/inventory.ini site.yml**

1. **Limit the Hosts**

**If you want to limit the playbook to a specific host or group from the inventory, use the --limit flag:**

**ansible-playbook -i inventory.ini site.yml --limit webservers**

1. **Running with Elevated Privileges (Sudo)**

**If the tasks in your playbook require superuser privileges, you can use the --become flag to run the playbook with sudo (or become in Ansible terms):**

**ansible-playbook site.yml --become**

**Alternatively, you can use the -K flag to prompt for the sudo password:**

**ansible-playbook site.yml -K**

1. **Check Mode (Dry Run)**

**To simulate the playbook execution without making actual changes (for testing), use the --check flag:**

**ansible-playbook site.yml --check**

**Combining --check with Other Options**

**You can combine --check with other flags like -i for specifying a custom inventory file, --limit for limiting which hosts to run the playbook on, and -v for verbosity:**

**ansible-playbook -i inventory.ini site.yml --check --limit webservers -v**

1. **Verbose Output**

**To get more detailed output (useful for debugging), you can use the -v, -vv, or -vvv flags for increasing verbosity:**

**ansible-playbook site.yml -vvv**

**-v shows basic output.**

**-vv gives more detailed output.**

**-vvv gives very detailed output, including task execution details.**

1. **Extra Variables**

**You can pass extra variables to the playbook with the --extra-vars or -e option:**

**ansible-playbook site.yml --extra-vars "var1=value1 var2=value2"**

### Example Command with All Options

**Here’s an example of running a playbook with multiple options:**

**ansible-playbook -i inventory.ini site.yml --limit webservers --become -K -vvv**

**This will:**

* **Use the inventory.ini file for host definitions.**
* **Run the playbook on the webservers group only.**
* **Execute with sudo privileges.**
* **Prompt for the sudo password.**
* **Provide verbose output.**

**Yaml file syntax check  
  
If you're asking how to check the syntax of a YAML file (commonly .yml or .yaml files) for errors or issues, especially in the context of Ansible playbooks, here are a few ways to do so.**

### 1. Using ansible-playbook with --syntax-check

**Ansible provides a built-in command option to check the syntax of a playbook without executing it. This is useful for ensuring your YAML files are correctly formatted and your Ansible playbooks are free of syntax errors.**

**ansible-playbook <playbook-file.yml> --syntax-check**

### Example:

**If you have a playbook called site.yml and want to check its syntax, you would run:**

**ansible-playbook site.yml --syntax-check**

**This will perform a syntax check, and if there are any issues with your YAML or the playbook structure, it will report them. It does not execute the playbook, it only checks for syntax errors.**

#### Example Output:

**Playbook syntax check passed**

**If there are errors in the playbook, you will see error messages pointing to the lines causing issues.**

### 2. Using YAML Lint

**If you're looking to check the syntax of a YAML file in general (not just Ansible playbooks), you can use a tool called YAML Lint. YAML Lint checks if your YAML file follows the correct structure and formatting rules for YAML syntax.**

#### Steps:

1. **Install YAML Lint:**

**On Ubuntu/Debian, you can install it via apt:  
sudo apt install yamllint**

**Alternatively, you can use pip to install YAML Lint if you have Python installed:  
pip install yamllint**

**Run YAML Lint: To check a YAML file (e.g., site.yml), run:  
yamllint site.yml**

1. **This will check the YAML syntax and provide feedback.**

#### Example Output:

**site.yml**

**1:1 warning missing document start "---" (document-start)**

**3:10 error wrong indentation: expected 2 but found 4 (indentation)**

**5:3 error too many spaces inside braces (brace)**

### Summary of Commands:

**For Ansible playbooks syntax check:  
ansible-playbook <playbook-file.yml> --syntax-check**

**For general YAML syntax check using YAML Lint:  
yamllint <file-name.yml>**

**Important modules:**

**1. System Management**

* **apt, yum, dnf: Package management.**
* **service, systemd: Service management.**
* **user, group: User and group management.**
* **cron: Manage cron jobs.**

### 2. File Management

* **file: Manage files and directories.**
* **copy: Copy files to remote hosts.**
* **template: Deploy Jinja2 templated files.**
* **unarchive: Extract archives.**
* **fetch: Fetch files from remote hosts.**
* **stat: Gather file or directory information.**

### 3. Networking

* **uri: Interact with REST APIs.**
* **firewalld: Configure firewalls.**
* **ufw: Manage uncomplicated firewalls.**
* **wait\_for: Wait for a specific condition (e.g., port availability).**

### 4. Database

* **mysql\_db: Manage MySQL databases.**
* **mysql\_user: Manage MySQL users.**
* **mysql\_query: Run SQL queries.**
* **postgresql\_db: Manage PostgreSQL databases.**
* **postgresql\_user: Manage PostgreSQL users.**

### 5. Cloud Management

* **AWS:**
  + **ec2: Manage EC2 instances.**
  + **s3: Manage S3 buckets.**
  + **cloudformation: Manage CloudFormation stacks.**
* **Azure:**
  + **azure\_rm\_virtualmachine: Manage Azure VMs.**
  + **azure\_rm\_storageaccount: Manage Azure storage accounts.**
* **GCP:**
  + **gcp\_compute\_instance: Manage GCP instances.**
  + **gcp\_storage\_bucket: Manage GCP storage buckets.**

### 6. Container Management

* **docker\_container: Manage Docker containers.**
* **docker\_image: Manage Docker images.**
* **k8s: Manage Kubernetes resources.**

### 7. Configuration Management

* **lineinfile: Manage lines in files.**
* **replace: Replace text in files.**
* **blockinfile: Insert a block of text in a file.**

### 8. Shell and Commands

* **command: Execute commands.**
* **shell: Execute shell commands.**
* **script: Run scripts on remote machines.**

### 9. Source Control

* **git: Manage Git repositories.**
* **svn: Manage SVN repositories.**

### 10. Monitoring

* **debug: Print debug messages or variables.**
* **assert: Validate conditions.**

### 11. Archive and Compression

* **archive: Create compressed files.**
* **unarchive: Extract files.**

### 12. Security

* **ansible.builtin.secrets: Manage secrets with Ansible Vault.**
* **openssl\_certificate: Generate SSL certificates.**

### 13. Virtualization

* **virt: Manage virtual machines.**
* **vmware\_guest: Manage VMware virtual machines.**

### 14. Utilities

* **set\_fact: Define custom facts.**
* **assert: Validate conditions in playbooks.**
* **include\_tasks: Include additional task files.**
* **include\_role: Include roles dynamically.**

### 15. Custom Integration

* **json\_query: Query JSON data.**
* **xml: Parse or modify XML files.**

**Ansible configuration**

### Order of Ansible Configuration

**1, Command-Line Options  
 Settings provided as command-line arguments to ansible, ansible-playbook, or other Ansible commands override all other configuration sources.  
Example:  
 ansible-playbook -i inventory.yml playbook.yml --user ubuntu**

**2, Environment Variables:  
 Environment variables can configure Ansible settings. If a setting is defined both in an environment variable and elsewhere, the environment variable takes precedence unless overridden by a command-line option.  
Example:  
  
 export ANSIBLE\_HOST\_KEY\_CHECKING=False**

**3,ansible.cfg in the Current Directory  
 Ansible looks for an ansible.cfg file in the current working directory where the command is run. This file takes precedence over system-wide or user-specific configurations.**

**4,~/.ansible.cfg in the User's Home Directory  
 A configuration file in the user's home directory (~/.ansible.cfg) applies if no ansible.cfg exists in the current directory.**

**5,System-Wide Configuration (/etc/ansible/ansible.cfg)  
 The global configuration file is used if no other configuration files are found. This is the default configuration file provided with Ansible.**

**6,Default Settings  
 If no configuration is specified in any of the above locations, Ansible falls back to its default settings.**

**Important configuration:**

### General Settings

1. **inventory  
   Specifies the default inventory file path.  
   Example: inventory = ./hosts**
2. **remote\_user  
   Default user for SSH connections.  
   Example: remote\_user = ansible\_user**
3. **forks  
   Maximum number of parallel processes.  
   Example: forks = 10**
4. **log\_path  
   Path to store logs.  
   Example: log\_path = /var/log/ansible.log**
5. **timeout  
   SSH connection timeout in seconds.  
   Example: timeout = 30**

### Privilege Escalation

1. **become  
   Enables privilege escalation.  
   Example: become = True**
2. **become\_method  
   Default method for privilege escalation (e.g., sudo, su).  
   Example: become\_method = sudo**
3. **become\_user  
   Default user for privilege escalation.  
   Example: become\_user = root**

### SSH Connection

1. **host\_key\_checking  
   Disables host key checking for new SSH hosts.  
   Example: host\_key\_checking = False**
2. **pipelining  
   Enables SSH pipelining for faster execution.  
   Example: pipelining = True**
3. **control\_path  
   Path for SSH control socket.  
   Example: control\_path = %(directory)s/%%h-%%r**

### 

### Performance

1. **strategy  
   Execution strategy (e.g., linear, free).  
   Example: strategy = free**
2. **retry\_files\_enabled  
   Enables retry file creation for failed tasks.  
   Example: retry\_files\_enabled = True**

### Error Handling

1. **action\_warnings  
   Display warnings for deprecated or unsupported actions.  
   Example: action\_warnings = False**
2. **deprecation\_warnings  
   Display deprecation warnings.  
   Example: deprecation\_warnings = False**

### Paths

1. **roles\_path  
   Directory to look for Ansible roles.  
   Example: roles\_path = ./roles:/etc/ansible/roles**
2. **library  
   Path to custom Ansible modules.  
   Example: library = ./library**
3. **callback\_plugins  
   Path for custom callback plugins.  
   Example: callback\_plugins = ./callback\_plugins**
4. **filter\_plugins  
   Path for custom filter plugins.  
   Example: filter\_plugins = ./filter\_plugins**

### Plugins

1. **stdout\_callback  
   Sets the default output format.  
   Example: stdout\_callback = yaml**

**Ansible variable**

**In Ansible, variables are used to store data and make your playbooks more flexible and reusable. You can define variables in different ways and access them throughout your playbook. Here's a breakdown of how variables work in Ansible:**

### Types of Variables in Ansible

1. **Inventory Variables**
2. **Playbook Variables**
3. **Command-line Variables**
4. **Fact Variables**
5. **Registered Variables**
6. **Default Variables**

### 1. Inventory Variables

**These variables are defined in your inventory file and are specific to each host or group of hosts.**

**Example:**

**all:**

**children:**

**web\_servers:**

**hosts:**

**web:**

**ansible\_host: 192.168.1.10**

**ansible\_user: ubuntu**

**mysql\_root\_password: 'rootpassword'**

### 2. Playbook Variables

**You can define variables in the playbook using the vars section.**

**Example:**

**---**

**- name: Example playbook**

**hosts: web\_servers**

**vars:**

**db\_name: 'my\_database'**

**db\_user: 'admin'**

**tasks:**

**- name: Print database variables**

**debug:**

**msg: "Database name is {{ db\_name }} and user is {{ db\_user }}"**

### 3. Command-line Variables

**You can pass variables when running the playbook using -e or --extra-vars:**

**Example:**

**ansible-playbook first\_playbook.yml -e "db\_name=my\_database db\_user=admin"**

**In your playbook, you can access the variables like so:**

**- name: Print database variables**

**debug:**

**msg: "Database name is {{ db\_name }} and user is {{ db\_user }}"**

### 4. Fact Variables

**Ansible collects system information (facts) about the target machine, which can be accessed using the ansible\_facts variable.**

**Example:**

**- name: Get the hostname**

**debug:**

**msg: "The hostname is {{ ansible\_hostname }}"**

### 5. Registered Variables

**You can capture the output of a task into a variable using the register keyword.**

**Example:**

**- name: Get the disk space**

**command: df -h**

**register: disk\_space**

**- name: Show disk space**

**debug:**

**msg: "{{ disk\_space.stdout }}"**

### 6. Default Variables

**You can use defaults to set default values for variables. If a variable is not defined, it will use the default value.**

**Example:**

**- name: Example with default variables**

**hosts: localhost**

**vars:**

**db\_user: "{{ db\_user | default('root') }}"**

**tasks:**

**- name: Print user**

**debug:**

**msg: "Database user is {{ db\_user }}"**

**If db\_user is not passed in, it defaults to 'root'.**

### Accessing Variables

**To access variables in your playbook, you simply reference them inside double curly braces {{ variable\_name }}.**

### Common Variable Lookup Methods:

* **vars: Custom variables defined within the playbook.**
* **facts: Automatically gathered facts about the target systems.**
* **env: Access environment variables.**

### Example Playbook with Variables

**---**

**- name: Set up MySQL**

**hosts: web\_servers**

**vars:**

**mysql\_root\_password: "password"**

**mysql\_user: "siva"**

**mysql\_db: "sample"**

**tasks:**

**- name: Ensure MySQL is installed**

**apt:**

**name: mysql-server**

**state: present**

**- name: Create MySQL database**

**mysql\_db:**

**name: "{{ mysql\_db }}"**

**state: present**

**login\_user: root**

**login\_password: "{{ mysql\_root\_password }}"**

**- name: Create MySQL user**

**mysql\_user:**

**name: "{{ mysql\_user }}"**

**password: "{{ mysql\_root\_password }}"**

**priv: "{{ mysql\_db }}.\*:ALL"**

**state: present**

**login\_user: root**

**login\_password: "{{ mysql\_root\_password }}"**

**In the above example:**

* **mysql\_root\_password, mysql\_user, and mysql\_db are variables.**
* **We use {{ variable\_name }} to substitute these variables into the tasks.**

### Best Practices for Using Variables

* **Separation of Variables: Store sensitive variables (like passwords) in separate files and use Ansible Vault to encrypt them.**
* **Avoid Hardcoding: Try not to hardcode values directly in the playbook to make it reusable and flexible.**
* **Use Default Values: Provide default values for variables when necessary to prevent errors when they are not defined.**

### Summary of Key Concepts:

* **Variable Definition: Define variables in vars or in the inventory file.**
* **Accessing Variables: Access variables in your tasks with {{ variable\_name }}.**
* **Command-line Variables: Use -e to pass variables via the command line.**
* **Registering Output: Capture output from tasks using register.**

**Ansible debug**

**In Ansible, the debug module is useful for printing information to help troubleshoot your playbooks. You can use it to print variables, output, and other useful information during the playbook execution.**

**Here are some ways to use the debug module in your playbook:**

### 1. Debugging Variables

**If you want to print the value of a variable during playbook execution, you can use the debug module as follows:**

**- name: Debugging a variable**

**debug:**

**var: db\_name**

**This will print the value of the db\_name variable.**

### 2. Custom Messages

**You can also print custom messages by using the msg argument:**

**- name: Print a custom message**

**debug:**

**msg: "The database name is {{ db\_name }}"**

**This will print: The database name is sample3 (if db\_name is sample3).**

### 3. Print the Result of a Task

**You can debug the result of a specific task by printing the output of that task.**

**For example, if you have a command that returns an output, you can capture and print the result:**

**- name: Run a command to show the MySQL version**

**command: mysql --version**

**register: mysql\_version**

**- name: Debug the command output**

**debug:**

**var: mysql\_version.stdout**

**This will run the mysql --version command, store the result in the mysql\_version variable, and then print the output (stdout).**

### 4. Conditional Debugging

**You can also use debug with conditionals to print information only when certain conditions are met:**

**- name: Debug when condition is met**

**debug:**

**msg: "Database name is {{ db\_name }}"**

**when: db\_name == "sample3"**

**This will print the message only if the db\_name variable is sample3.**

### 5. Debugging with Complex Variables

**If you want to debug a complex variable (like a dictionary or list), you can use the debug module with the var option.**

**- name: Debug a complex variable**

**debug:**

**var: my\_complex\_variable**

**This will display the entire content of the my\_complex\_variable variable, whether it's a list or dictionary.**

### Example Playbook Using Debug:

**---**

**- name: Example to debug variables and tasks**

**hosts: localhost**

**vars:**

**db\_name: "sample\_db"**

**db\_user: "root"**

**tasks:**

**- name: Debug the db\_name variable**

**debug:**

**var: db\_name**

**- name: Print a custom debug message**

**debug:**

**msg: "The database user is {{ db\_user }}"**

**- name: Run a command and debug the output**

**command: echo "Hello, world!"**

**register: command\_output**

**- name: Debug the command output**

**debug:**

**var: command\_output.stdout**

**Group & host variable**

**In Ansible, group variables and host variables are used to define specific configurations and values for hosts and groups of hosts in your inventory.**

### 1. Host Variables

**Host variables are variables that are specific to a single host in your inventory. They can be defined for each host to configure specific settings.**

**How to define host variables:**

* **Directly in the inventory file (usually with hostvars)**
* **In a separate host\_vars directory**

#### Example 1: Defining host variables in an inventory file

**all:**

**hosts:**

**web\_server:**

**ansible\_host: 192.168.1.100**

**ansible\_user: ubuntu**

**db\_name: web\_db**

**db\_password: secret\_password**

**In the example above, web\_server has two host-specific variables: db\_name and db\_password.**

**You can reference these variables in your playbook using hostvars:**

**- name: Print host variable**

**debug:**

**msg: "Database name for web\_server is {{ hostvars['web\_server']['db\_name'] }}"**

#### Example 2: Using host\_vars directory

**Ansible also allows you to store variables in separate files. The host\_vars directory is used to store host-specific variables.**

**Directory structure:**

**.**

**├── host\_vars**

**│ └── web\_server.yml**

**└── inventory.yml**

**host\_vars/web\_server.yml content:**

**db\_name: web\_db**

**db\_password: secret\_password**

**Then in your playbook, you can access these variables directly:**

**- name: Print host variable**

**debug:**

**msg: "Database name for web\_server is {{ db\_name }}"**

### 2. Group Variables

**Group variables are variables that apply to all hosts within a group. You can define group variables in the inventory file or in a separate group\_vars directory.**

#### Example 1: Defining group variables in the inventory file

**all:**

**children:**

**web\_servers:**

**hosts:**

**web1:**

**ansible\_host: 192.168.1.101**

**web2:**

**ansible\_host: 192.168.1.102**

**vars:**

**db\_name: web\_db**

**db\_user: admin**

**Here, web\_servers is a group with hosts web1 and web2. The db\_name and db\_user variables apply to both hosts.**

#### Example 2: Using group\_vars directory

**Just like host\_vars, Ansible supports defining group variables in a group\_vars directory.**

**Directory structure:**

**Copy code**

**.**

**├── group\_vars**

**│ └── web\_servers.yml**

**└── inventory.yml**

**group\_vars/web\_servers.yml content:**

**db\_name: web\_db**

**db\_user: admin**

**In your playbook, you can reference these variables like this:**

**- name: Print group variable**

**debug:**

**msg: "Database name for web\_servers group is {{ db\_name }}"**

### 3. Combining Host and Group Variables

**Ansible will prioritize variables defined for a specific host over those defined for a group. For example, if a variable is defined in both the group and the host, Ansible will use the value from the host-specific variable.**

**all:**

**children:**

**web\_servers:**

**hosts:**

**web1:**

**ansible\_host: 192.168.1.101**

**db\_name: special\_db # Host variable will override the group variable**

**web2:**

**ansible\_host: 192.168.1.102**

**vars:**

**db\_name: web\_db # Group variable**

* **For web1, the value of db\_name will be special\_db.**
* **For web2, the value of db\_name will be web\_db.**

### 4. Using Variables in Playbooks

**Once you have defined host or group variables, you can use them in your playbooks by referencing them directly.**

#### Example Playbook with Host and Group Variables:

**---**

**- name: Install and configure web server**

**hosts: web\_servers**

**vars:**

**apache\_version: "2.4" # Local variable for the playbook**

**tasks:**

**- name: Print host variable**

**debug:**

**msg: "Database name is {{ db\_name }}"**

**- name: Print group variable**

**debug:**

**msg: "Database user is {{ db\_user }}"**

**- name: Print playbook variable**

**debug:**

**msg: "Apache version is {{ apache\_version }}"**

### 5. Ansible Variable Precedence

**Ansible follows a variable precedence hierarchy. If the same variable is defined in multiple places, Ansible will use the value from the highest-precedence source. The order of precedence is:**

1. **Extra vars (ansible-playbook -e)**
2. **Playbook vars (defined in the playbook itself)**
3. **Host vars (defined in host\_vars/hostname.yml)**
4. **Group vars (defined in group\_vars/group\_name.yml)**
5. **Inventory vars (defined in the inventory file)**
6. **Facts (gathered system information)**
7. **Defaults (defined in roles or included files)**

### Summary:

* **Host variables are specific to individual hosts.**
* **Group variables apply to all hosts in a group.**
* **You can define variables in inventory files, host\_vars, and group\_vars directories.**
* **Host variables override group variables.**

**Fact variable**

**In Ansible, facts are variables that provide system information gathered from managed hosts. These facts can be gathered automatically using the setup module, and you can also define custom facts using the set\_fact module.**

**When statement**

**In Ansible, the when statement is used to add conditional logic to tasks. It allows you to run a task only if a certain condition is met. This condition can be based on variables, facts, or the outcome of previous tasks.**

**The when keyword is a powerful way to control the flow of your playbook, ensuring that specific tasks only execute when certain conditions are true.**

### Basic Syntax of when

**task\_name:**

**...**

**when: condition**

### Examples of Using when in Ansible

#### 1. Condition Based on a Variable

**You can run a task based on the value of a variable.**

**---**

**- name: Install Apache if the flag is set to true**

**hosts: all**

**become: yes**

**vars:**

**install\_apache: true**

**tasks:**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present**

**when: install\_apache**

**In this example, the task will only run if the install\_apache variable is true.**

#### 2. Condition Based on a Fact

**You can use facts gathered by Ansible, such as system information, to make decisions. For example, you might want to install a package only if the system is Ubuntu:**

**---**

**- name: Install Apache on Ubuntu**

**hosts: all**

**become: yes**

**tasks:**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present**

**when: ansible\_facts['os\_family'] == "Debian"**

**Here, the task runs only if the system's OS family is Debian (which includes Ubuntu).**

#### 3. Combining Multiple Conditions

**You can combine multiple conditions using and, or, and parentheses for more complex logic.**

**---**

**- name: Install packages based on conditions**

**hosts: all**

**become: yes**

**vars:**

**install\_apache: true**

**install\_mysql: false**

**tasks:**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present**

**when: install\_apache and not install\_mysql**

**In this example, Apache will be installed only if install\_apache is true and install\_mysql is false.**

#### 4. Using when with register

**The when statement can also be used in combination with the register keyword to execute tasks based on the result of a previous task.**

**---**

**- name: Check if Apache is installed**

**hosts: all**

**tasks:**

**- name: Check if Apache is installed**

**shell: dpkg -l | grep apache2**

**register: apache\_installed**

**ignore\_errors: yes**

**- name: Install Apache if not installed**

**apt:**

**name: apache2**

**state: present**

**when: apache\_installed.rc != 0**

**In this example:**

* **The first task checks if Apache is installed by running a shell command.**
* **The second task installs Apache if the first task shows that Apache is not installed (apache\_installed.rc != 0).**

#### 5. Using when with Lists

**You can check if a value is in a list.**

**---**

**- name: Ensure specific users are present**

**hosts: all**

**become: yes**

**vars:**

**users\_to\_create:**

**- alice**

**- bob**

**tasks:**

**- name: Create users**

**user:**

**name: "{{ item }}"**

**state: present**

**loop:**

**- alice**

**- bob**

**- charlie**

**when: item in users\_to\_create**

**Here, the task creates users only if their names are in the users\_to\_create list.**

### Complex Conditions with when

**You can also use more complex expressions with when for more control over task execution. You can check multiple conditions using logical operators (and, or, not), or even use regular expressions.**

#### Example: Complex Conditional with Regular Expression

**---**

**- name: Check and install a package based on a version string**

**hosts: all**

**become: yes**

**vars:**

**package\_version: "1.2.3"**

**tasks:**

**- name: Install specific version of a package**

**apt:**

**name: "my\_package={{ package\_version }}"**

**state: present**

**when: package\_version is match("^\d+\.\d+\.\d+$") # Ensures the version is in the correct format**

### Use Cases for when in Ansible

* **Conditional execution of tasks: Only run certain tasks under specific conditions.**
* **Environment-specific configurations: Apply different configurations based on the environment (e.g., development vs. production).**
* **Task skipping: Skip certain tasks if a condition isn't met.**
* **Checking results from previous tasks: Perform tasks only if previous ones were successful or failed.**

### Conclusion

**The when statement in Ansible is a powerful way to introduce conditional logic in your playbooks. Whether you are checking variables, facts, or task results, when enables you to control which tasks are executed based on specific conditions.**

**Loop**

**In Ansible, the loop directive is used to iterate over a list of items or dictionaries and execute a task for each item. This is useful when you need to repeat a task for multiple items, such as creating multiple users, installing packages, or processing lists of values.**

### Basic Syntax of loop

**yaml**

**Copy code**

**- name: Task to loop**

**<module>:**

**<parameter>: "{{ item }}"**

**loop:**

**- item1**

**- item2**

**- item3**

### Examples of Using loop in Ansible

#### 1. Basic Loop over a List

**This is a simple example where a task is repeated for each item in the list.**

**---**

**- name: Install multiple packages**

**hosts: all**

**become: yes**

**tasks:**

**- name: Install packages**

**apt:**

**name: "{{ item }}"**

**state: present**

**loop:**

**- nginx**

**- apache2**

**- mysql-server**

**In this example, Ansible will install nginx, apache2, and mysql-server one after the other.**

#### 2. Using loop with Variables

**You can also use a variable to loop over a list dynamically.**

**---**

**- name: Install packages dynamically**

**hosts: all**

**become: yes**

**vars:**

**packages:**

**- vim**

**- curl**

**- git**

**tasks:**

**- name: Install packages from list**

**apt:**

**name: "{{ item }}"**

**state: present**

**loop: "{{ packages }}"**

**Here, the loop uses the packages variable, so it will install the packages defined in that list.**

#### 3. Loop with Dictionary Items

**You can loop over dictionaries (key-value pairs), which is useful for tasks like managing users or configuring multiple services.**

**---**

**- name: Add multiple users**

**hosts: all**

**become: yes**

**vars:**

**users:**

**- { name: "alice", shell: "/bin/bash" }**

**- { name: "bob", shell: "/bin/zsh" }**

**tasks:**

**- name: Create users with specific shells**

**user:**

**name: "{{ item.name }}"**

**shell: "{{ item.shell }}"**

**state: present**

**loop: "{{ users }}"**

**In this case, Ansible will create the users alice and bob with their respective shell configurations.**

#### 4. Looping with a Range of Numbers

**You can use the range() function in the loop to iterate over a range of numbers.**

**yaml**

**Copy code**

**---**

**- name: Create files with numbers**

**hosts: all**

**become: yes**

**tasks:**

**- name: Create numbered files**

**file:**

**path: "/tmp/file\_{{ item }}"**

**state: touch**

**loop: "{{ range(1, 6) | list }}"**

**This will create files named /tmp/file\_1, /tmp/file\_2, /tmp/file\_3, /tmp/file\_4, and /tmp/file\_5.**

#### 5. Loop with Complex Expressions

**You can use Jinja2 filters or complex expressions in the loop to customize the loop's behavior.**

**---**

**- name: Install packages based on a condition**

**hosts: all**

**become: yes**

**vars:**

**packages:**

**- name: "nginx"**

**version: "1.18.0"**

**- name: "mysql-server"**

**version: "5.7"**

**- name: "apache2"**

**version: "2.4"**

**tasks:**

**- name: Install specific version of a package**

**apt:**

**name: "{{ item.name }}={{ item.version }}"**

**state: present**

**loop: "{{ packages }}"**

**This example installs specific versions of packages defined in a list of dictionaries.**

#### 6. Using loop with with\_items (Deprecated but still in use)

**While the loop directive is preferred, you can still use the older with\_items method for looping over a list. However, loop is more modern and recommended.**

**yaml**

**Copy code**

**---**

**- name: Install multiple packages using with\_items**

**hosts: all**

**become: yes**

**tasks:**

**- name: Install packages**

**apt:**

**name: "{{ item }}"**

**state: present**

**with\_items:**

**- nginx**

**- apache2**

**- mysql-server**

### Combining loop with Other Ansible Directives

#### 1. when with loop

**You can combine when with loop to conditionally execute tasks within the loop.**

**---**

**- name: Install packages only if not already installed**

**hosts: all**

**become: yes**

**vars:**

**packages:**

**- nginx**

**- apache2**

**- mysql-server**

**tasks:**

**- name: Install package**

**apt:**

**name: "{{ item }}"**

**state: present**

**loop: "{{ packages }}"**

**when: ansible\_facts['distribution'] == 'Ubuntu'**

**This example installs packages only on Ubuntu machines, and it will loop through the packages list.**

#### 2. Using loop to Manage Multiple Files

**You can use loop to create or manage multiple files.**

**---**

**- name: Create multiple files**

**hosts: all**

**become: yes**

**tasks:**

**- name: Create files**

**file:**

**path: "/tmp/file\_{{ item }}"**

**state: touch**

**loop:**

**- 1**

**- 2**

**- 3**

**- 4**

**This will create /tmp/file\_1, /tmp/file\_2, etc., on the target machines.**

### Conclusion

**The loop directive is a powerful feature in Ansible that allows you to perform repetitive tasks more efficiently. It can be used with simple lists, dictionaries, ranges, and even complex expressions. By leveraging loop, you can significantly reduce redundancy in your playbooks and simplify the automation of multiple tasks that follow similar patterns.**

**File,copy and template**

**In Ansible, the file, copy, and template modules are used to manage files and templates on the target system. These modules help you copy files from your local machine to the remote target, ensure specific file permissions, and manage the content of files using Jinja2 templates. Below is an explanation and examples of how each module works.**

### 1. file Module

**The file module is used to manage file properties like permissions, ownership, and whether the file or directory should exist.**

#### Key Parameters:

* **path: The path to the file or directory.**
* **state: The desired state of the file or directory. Can be file, directory, or absent.**
* **mode: Set file permissions in symbolic or octal form.**
* **owner: Set the file owner.**
* **group: Set the file group.**
* **recurse: Used to apply changes recursively to directories.**

#### Example: Creating a file with specific permissions

**---**

**- name: Ensure the file exists with correct permissions**

**hosts: all**

**become: yes**

**tasks:**

**- name: Create a file with specific owner and permissions**

**file:**

**path: "/tmp/myfile.txt"**

**state: touch # Ensure the file exists**

**owner: root**

**group: root**

**mode: '0644'**

**This example ensures that the file /tmp/myfile.txt exists, is owned by root, belongs to the root group, and has the 0644 permissions.**

#### Example: Remove a file

**---**

**- name: Remove file from the system**

**hosts: all**

**become: yes**

**tasks:**

**- name: Delete the file**

**file:**

**path: "/tmp/myfile.txt"**

**state: absent # This will remove the file**

**This will delete the file /tmp/myfile.txt if it exists.**

### 2. copy Module

**The copy module is used to copy files from the local machine (control machine) to the remote target machine.**

#### Key Parameters:

* **src: The source file or directory (local path).**
* **dest: The destination path on the remote machine.**
* **mode: Set file permissions.**
* **owner: Set file owner.**
* **group: Set file group.**
* **content: If you don't have a local file, you can directly specify content to be written to the file.**

#### Example: Copy a file to a remote server

**---**

**- name: Copy a file from local to remote**

**hosts: all**

**become: yes**

**tasks:**

**- name: Copy a configuration file**

**copy:**

**src: "/path/to/local/file.conf"**

**dest: "/etc/myconfig.conf"**

**owner: root**

**group: root**

**mode: '0644'**

**In this example, the file.conf from the local machine is copied to /etc/myconfig.conf on the target server, ensuring it has the right owner and permissions.**

#### Example: Copy content to a file

**---**

**- name: Copy content into a file**

**hosts: all**

**become: yes**

**tasks:**

**- name: Copy content directly into a file**

**copy:**

**content: |**

**Hello, this is a test file.**

**This is the second line.**

**dest: "/tmp/testfile.txt"**

**owner: root**

**group: root**

**mode: '0644'**

**In this example, the content is directly copied into /tmp/testfile.txt on the target machine.**

### 3. template Module

**The template module is used to manage files that contain Jinja2 variables or expressions. The template module processes the template file (which can include variables, loops, and conditionals) and renders it into the final content before copying it to the target location.**

#### Key Parameters:

* **src: The source template file (local path).**
* **dest: The destination path on the remote machine.**
* **owner: Set file owner.**
* **group: Set file group.**
* **mode: Set file permissions.**

#### Example: Using a template file

1. **Template file (/path/to/template.conf.j2):**

**# This is a configuration file**

**server\_name = {{ server\_name }}**

**server\_port = {{ server\_port }}**

1. **Playbook using template:**

**---**

**- name: Copy a template to the remote system**

**hosts: all**

**become: yes**

**vars:**

**server\_name: "example.com"**

**server\_port: 8080**

**tasks:**

**- name: Copy the template file to the remote server**

**template:**

**src: "/path/to/template.conf.j2"**

**dest: "/etc/myconfig.conf"**

**owner: root**

**group: root**

**mode: '0644'**

**In this example:**

* **The template.conf.j2 file contains Jinja2 syntax ({{ server\_name }} and {{ server\_port }}).**
* **The template module processes the template, substituting the values of server\_name and server\_port from the playbook.**
* **The resulting configuration file will be copied to /etc/myconfig.conf on the target machine.**

### Key Differences Between copy and template Modules:

* **copy: Copies a file directly without any modification. It is used when you have a static file or content to copy.**
* **template: Used for dynamic content. It processes Jinja2 templates and applies variables, loops, conditionals, etc., before copying the result to the destination.**

### Conclusion:

* **Use the file module for managing file properties like permissions and ownership, or to ensure files exist or are removed.**
* **Use the copy module to copy files from your local machine to the remote target machine.**
* **Use the template module when you need to process dynamic content (Jinja2 templates) and copy it to the remote target.**

**Handlers**

**In Ansible, handlers are special tasks that are only executed when they are notified by other tasks. Handlers are typically used for tasks that need to be performed only once after a change has been made, such as restarting a service after modifying its configuration file. Handlers are similar to regular tasks, but they are only triggered when notified, and they run at the end of a playbook or role.**

### Key Concepts of Handlers:

1. **Notification: A task notifies a handler when it makes a change (e.g., updates a file, installs a package).**
2. **Execution: Handlers are only executed if they are notified, and they are executed once per play, regardless of how many times they are notified.**
3. **Run at the end: Handlers are run at the end of the playbook, after all tasks have completed, to allow for multiple notifications to be handled in a single execution.**

### Syntax for Handlers:

1. **Define Handlers: Handlers are defined just like regular tasks, but they are placed under the handlers section.**
2. **Notify a Handler: To notify a handler, you use the notify directive within a task that will trigger the handler if a change occurs.**

### Example of Handlers:

#### Playbook: Restarting a Service After Configuration Change

**---**

**- name: Configure a server**

**hosts: all**

**become: yes**

**tasks:**

**- name: Copy the new configuration file**

**copy:**

**src: "/path/to/local/config.conf"**

**dest: "/etc/myservice/config.conf"**

**owner: root**

**group: root**

**mode: '0644'**

**notify: Restart myservice # Notify the handler to restart the service**

**- name: Install the latest version of the application**

**apt:**

**name: myapp**

**state: latest**

**notify: Restart myservice # Notify the handler to restart the service**

**handlers:**

**- name: Restart myservice**

**service:**

**name: myservice**

**state: restarted**

### Explanation:

1. **Tasks Section:**
   * **Copying a file: The first task copies a new configuration file to the remote machine. After this task completes, it triggers the handler Restart myservice because of the notify directive.**
   * **Installing a package: The second task installs or updates the package myapp. If the package is updated, it will also trigger the handler Restart myservice.**
2. **Handlers Section:**
   * **Restart myservice: This handler restarts the service myservice if it is notified by any task in the playbook. It will only be executed once, even if it is notified multiple times during the play.**

### Important Points:

1. **Handlers are executed once: Even if you notify the handler multiple times in your tasks, it will only be executed once at the end of the playbook.**
2. **Tasks that notify handlers must result in a change: Handlers are triggered only when a task results in a change (e.g., modifying a file, installing a package, etc.). Tasks that do not cause a change will not notify the handler.**
3. **Handlers run at the end of the playbook: Handlers are executed after all tasks in a play have been completed. This allows them to run only once, after all changes are made.**

### Example with Multiple Handlers:

**You can define multiple handlers to handle different services or tasks.**

**---**

**- name: Configure multiple services**

**hosts: all**

**become: yes**

**tasks:**

**- name: Update nginx configuration**

**copy:**

**src: "/path/to/nginx/config.conf"**

**dest: "/etc/nginx/nginx.conf"**

**notify: Restart nginx**

**- name: Update apache configuration**

**copy:**

**src: "/path/to/apache/config.conf"**

**dest: "/etc/apache2/apache2.conf"**

**notify: Restart apache**

**handlers:**

**- name: Restart nginx**

**service:**

**name: nginx**

**state: restarted**

**- name: Restart apache**

**service:**

**name: apache2**

**state: restarted**

**In this example, if either nginx or apache configuration is modified, their respective handlers will be notified and executed at the end of the playbook.**

### Common Use Cases for Handlers:

1. **Restarting Services: After updating configuration files or installing new software, you might need to restart the associated service to apply the changes.**
2. **Reloading Configuration: After changing application configuration files, a reload or restart may be necessary for the changes to take effect.**
3. **Rebuilding or Re-initializing Servers: After major changes to the system configuration (like upgrading packages or installing new dependencies), you may want to reboot or reinitialize the server.**

**Roles**

**In Ansible, roles provide a way to organize and reuse tasks, variables, handlers, templates, and other resources in a structured manner. Roles make playbooks cleaner, more modular, and easier to maintain, especially in larger projects.**

### Key Features of Roles:

1. **Modularity: Breaks down playbooks into reusable components.**
2. **Reusability: Roles can be reused across different projects or playbooks.**
3. **Structure: Provides a standardized directory structure for organizing resources.**
4. **Separation of Concerns: Roles keep tasks, variables, handlers, and templates organized and separate.**

### Role Directory Structure

**When creating a role, it follows a specific directory structure:**

**roles/**

**<role\_name>/**

**tasks/ # Contains the main task list**

**main.yml**

**handlers/ # Contains handlers triggered by tasks**

**main.yml**

**templates/ # Contains Jinja2 templates**

**files/ # Contains static files to be copied**

**vars/ # Contains variables with higher precedence**

**main.yml**

**defaults/ # Contains default variables with lower precedence**

**main.yml**

**meta/ # Contains metadata about the role**

**main.yml**

**tests/ # Contains test playbooks**

**README.md # (Optional) Documentation for the role**

### Creating and Using a Role

#### 1. Creating a Role

**Use the ansible-galaxy command to create a role:**

**ansible-galaxy init <role\_name>**

**Example:**

**ansible-galaxy init apache**

**This will create a directory structure for the apache role.**

#### 2. Defining the Role

**Populate the files in the tasks, templates, vars, or other directories as needed.**

* **tasks/main.yml: Define tasks to be executed.**

**---**

**# roles/apache/tasks/main.yml**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present**

**- name: Start and enable Apache**

**service:**

**name: apache2**

**state: started**

**enabled: true**

* **vars/main.yml: Define variables specific to the role.**

**---**

**# roles/apache/vars/main.yml**

**apache\_port: 80**

* **templates/vhost.conf.j2: Create a Jinja2 template for a virtual host configuration.**

**jinja2**

**# roles/apache/templates/vhost.conf.j2**

**<VirtualHost \*:{{ apache\_port }}>**

**DocumentRoot "/var/www/html"**

**</VirtualHost>**

#### 3. Using a Role in a Playbook

**Reference the role in your playbook:**

**---**

**- name: Configure web servers**

**hosts: web**

**roles:**

**- apache**

#### 4. Passing Variables to a Role

**You can pass variables to a role from the playbook:**

**---**

**- name: Configure web servers**

**hosts: web**

**roles:**

**- role: apache**

**vars:**

**apache\_port: 8080**

### Role Dependencies

**You can specify dependencies for a role in the meta/main.yml file. For example:**

**---**

**# roles/apache/meta/main.yml**

**dependencies:**

**- role: common**

**- role: php**

**In this case, the common and php roles will run before the apache role.**

### Role Best Practices

1. **Use Defaults for Configurable Variables:**
   * **Place variables in defaults/main.yml so they can be easily overridden.**
2. **Minimize Variables in vars/main.yml:**
   * **Use vars for non-overridable or critical variables.**
3. **Keep Tasks Focused:**
   * **Ensure each role performs a specific function (e.g., configuring Apache, deploying an app).**
4. **Use Templates for Dynamic Files:**
   * **Use Jinja2 templates to make files like configuration files dynamic.**
5. **Test Roles Independently:**
   * **Use test playbooks or tools like Molecule to test roles.**

### Example: Setting Up Apache with Roles

#### Directory Structure:

**roles/**

**apache/**

**tasks/**

**main.yml**

**templates/**

**vhost.conf.j2**

**defaults/**

**main.yml**

#### Role Files:

* **tasks/main.yml:**

**---**

**- name: Install Apache**

**apt:**

**name: apache2**

**state: present**

**- name: Copy VirtualHost template**

**template:**

**src: vhost.conf.j2**

**dest: /etc/apache2/sites-available/000-default.conf**

**- name: Start and enable Apache**

**service:**

**name: apache2**

**state: started**

**enabled: true**

* **defaults/main.yml:**

**---**

**apache\_port: 80**

* **templates/vhost.conf.j2:**

**jinja2**

**<VirtualHost \*:{{ apache\_port }}>**

**DocumentRoot "/var/www/html"**

**</VirtualHost>**

#### 

#### Playbook:

**---**

**- name: Deploy Apache Server**

**hosts: web**

**roles:**

**- apache**

### Advantages of Roles

1. **Reusability: Share roles across multiple projects.**
2. **Maintainability: Keep playbooks clean and manageable.**
3. **Scalability: Use roles to standardize configurations across environments.**
4. **Separation of Concerns: Divide responsibilities among different roles.**

**By using roles, you can create robust, reusable, and maintainable automation workflows.**

**Ansible galaxy**

**Ansible Galaxy is a repository for discovering, sharing, and managing Ansible content, such as roles, collections, and plugins. It allows you to download reusable roles or collections created by others, or share your own automation solutions with the community.**

### Key Features of Ansible Galaxy

1. **Role and Collection Sharing: Publish and share your roles or collections with the Ansible community.**
2. **Content Discovery: Search for existing roles and collections to speed up your automation tasks.**
3. **Integration with ansible-galaxy CLI: Simplifies downloading and managing Ansible content.**
4. **Version Control: Download specific versions of roles or collections for better consistency.**

### Role vs Collection

* **Roles: A way to package and distribute tasks, handlers, templates, variables, and files.**
* **Collections: A broader package format that includes roles, modules, plugins, and playbooks.**

### Using Ansible Galaxy

#### 1. Search for Roles/Collections

**Visit the** [**Ansible Galaxy website**](https://galaxy.ansible.com) **to search for content. You can filter by role name, tags, author, or collection name.**

#### 2. Install a Role

**Use the ansible-galaxy command to install a role:**

**ansible-galaxy install <role\_name>**

**Example:**

**ansible-galaxy install geerlingguy.apache**

**This downloads the role to the default roles directory (~/.ansible/roles).**

### Install a Role to a Specific Directory

**To install a role in a custom directory, use the -p option:**

**ansible-galaxy install -p roles geerlingguy.apache**

**This installs the role into the roles directory of your current working directory.**

### Install a Collection

**Collections are installed similarly but use a different namespace:**

**ansible-galaxy collection install <collection\_name>**

**Example:**

**ansible-galaxy collection install community.general**

### List Installed Roles/Collections

**To list all installed roles:**

**ansible-galaxy list**

**To list installed collections:**

**ansible-galaxy collection list**

### Using a Role in a Playbook

**After installing a role, reference it in your playbook:**

**---**

**- name: Configure Apache Server**

**hosts: web**

**roles:**

**- geerlingguy.apache**

### Creating Your Own Role

**To create a new role:**

**ansible-galaxy init <role\_name>**

**Example:**

**ansible-galaxy init my\_custom\_role**

**This creates a role directory structure:**

**my\_custom\_role/**

**defaults/**

**files/**

**handlers/**

**meta/**

**tasks/**

**templates/**

**tests/**

**vars/**

### Publishing to Ansible Galaxy

**You can share your role by publishing it on Ansible Galaxy.**

1. **Login to Ansible Galaxy: Use your GitHub account to log in.**
2. **Host Your Role on GitHub: Ensure your role has a meta/main.yml file and follows the proper directory structure.**
3. **Link the Repository: Sync your GitHub repository with Ansible Galaxy.**
4. **Publish the Role: Use the web interface or ansible-galaxy CLI to publish.**

**Example meta/main.yml for publishing:**

**---**

**galaxy\_info:**

**author: your\_name**

**description: An Apache server setup role**

**company: your\_company**

**license: MIT**

**min\_ansible\_version: 2.9**

**platforms:**

**- name: Ubuntu**

**versions:**

**- bionic**

**- focal**

**dependencies: []**

### Advanced Features

#### Install Roles from requirements.yml

**You can define roles in a requirements.yml file to install multiple roles at once:**

**# requirements.yml**

**- name: geerlingguy.apache**

**version: 3.0.0**

**- name: geerlingguy.mysql**

**scm: git**

**src: https://github.com/geerlingguy/ansible-role-mysql.git**

**version: master**

**Install roles using:**

**ansible-galaxy install -r requirements.yml**

#### Install Collections from requirements.yml

**You can also define collections:**

**# requirements.yml**

**collections:**

**- name: community.general**

**version: 3.0.0**

**- name: ansible.posix**

**Install collections using:**

**ansible-galaxy collection install -r requirements.yml**

### Best Practices with Ansible Galaxy

1. **Use Stable Roles/Collections: Choose well-maintained and highly rated roles or collections.**
2. **Version Locking: Specify versions in requirements.yml for consistency across environments.**
3. **Test Before Use: Test downloaded content in a staging environment.**
4. **Customize Locally: If needed, customize the role/collection for your specific requirements.**

**Ansible for aws**

**Ansible provides robust tools to manage and automate AWS infrastructure. With the help of modules specifically designed for AWS, you can provision, configure, deploy, and manage AWS resources seamlessly. Below is an overview of how to use Ansible for AWS automation.**

### Pre-requisites for Using Ansible with AWS

1. **AWS Account: Ensure you have an AWS account with sufficient permissions.**
2. **Ansible Installed: Install Ansible on your local system or control node.**

**3. AWS CLI: Configure AWS CLI to verify access to AWS resources:  
  
 aws configure**

**4. Python Boto3 Library: Required for AWS modules in Ansible. Install it using pip:  
 pip install boto3 botocore**

### Setting Up Credentials

#### Option 1: AWS CLI Configuration

**Configure the default AWS credentials:**

**aws configure**

#### Option 2: Environment Variables

**Set AWS credentials using environment variables:**

**export AWS\_ACCESS\_KEY\_ID=your\_access\_key**

**export AWS\_SECRET\_ACCESS\_KEY=your\_secret\_key**

**export AWS\_REGION=your\_region**

#### Option 3: Using an IAM Role

**If running Ansible from an EC2 instance, assign an IAM role to the instance.**

### Ansible Collections for AWS

**Ansible provides a collection called amazon.aws that includes AWS-specific modules and plugins.**

#### Install the Collection

**ansible-galaxy collection install amazon.aws**

#### Verify Installation

**ansible-galaxy collection list | grep amazon.aws**

### Basic AWS Modules in Ansible

1. **amazon.aws.ec2\_instance: Manage EC2 instances.**
2. **amazon.aws.s3\_bucket: Manage S3 buckets.**
3. **amazon.aws.rds\_instance: Manage RDS instances.**
4. **amazon.aws.cloudformation: Manage CloudFormation stacks.**
5. **amazon.aws.ec2\_vol: Manage EC2 volumes.**

### Example: Launch an EC2 Instance

#### Playbook

**- name: Launch an EC2 Instance**

**hosts: localhost**

**gather\_facts: false**

**collections:**

**- amazon.aws**

**tasks:**

**- name: Launch EC2 Instance**

**ec2\_instance:**

**aws\_access\_key: "{{ aws\_access\_key }}"**

**aws\_secret\_key: "{{ aws\_secret\_key }}"**

**region: "{{ aws\_region }}"**

**key\_name: "your-key-pair"**

**instance\_type: "t2.micro"**

**image\_id: "ami-0abcdef1234567890" # Replace with a valid AMI ID**

**count: 1**

**state: present**

**tags:**

**Name: "My-Ansible-Instance"**

**register: ec2**

**- name: Show instance information**

**debug:**

**var: ec2**

#### Run the Playbook

**ansible-playbook launch\_ec2.yml -e "aws\_access\_key=your\_key aws\_secret\_key=your\_secret aws\_region=your\_region"**

### Example: Create an S3 Bucket

**- name: Create an S3 Bucket**

**hosts: localhost**

**gather\_facts: false**

**collections:**

**- amazon.aws**

**tasks:**

**- name: Create S3 bucket**

**s3\_bucket:**

**name: my-ansible-s3-bucket**

**region: us-east-1**

**state: present**

**acl: public-read**

#### Run the Playbook

**ansible-playbook create\_s3.yml**

### Dynamic Inventory for AWS

**Instead of hardcoding instances, use Ansible's dynamic inventory to manage resources dynamically.**

#### Install the Dynamic Inventory Plugin

**pip install boto3**

#### Enable the Plugin

**Modify your ansible.cfg:**

**[defaults]**

**inventory = ./inventory.aws\_ec2.yaml**

#### Sample Dynamic Inventory Configuration

**inventory.aws\_ec2.yaml:**

**plugin: amazon.aws.aws\_ec2**

**regions:**

**- us-east-1**

**keyed\_groups:**

**- key: tags.Name**

**prefix: tag\_**

#### Run Playbook with Dynamic Inventory

**ansible-playbook -i inventory.aws\_ec2.yaml my\_playbook.yml**

### Advanced AWS Tasks with Ansible

#### Provision a VPC

**- name: Create a VPC**

**hosts: localhost**

**gather\_facts: false**

**collections:**

**- amazon.aws**

**tasks:**

**- name: Create a VPC**

**ec2\_vpc\_net:**

**name: my-ansible-vpc**

**cidr\_block: 10.0.0.0/16**

**region: us-east-1**

**state: present**

**register: vpc**

**- name: Show VPC details**

**debug:**

**var: vpc**

#### Create an RDS Instance

**- name: Create an RDS Instance**

**hosts: localhost**

**gather\_facts: false**

**collections:**

**- amazon.aws**

**tasks:**

**- name: Create RDS instance**

**rds\_instance:**

**db\_instance\_identifier: my-ansible-rds**

**db\_instance\_class: db.t3.micro**

**engine: mysql**

**master\_username: admin**

**master\_user\_password: secure\_password**

**allocated\_storage: 20**

**region: us-east-1**

**state: present**

**register: rds**

**- name: Show RDS details**

**debug:**

**var: rds**

### 

### 

### Best Practices

1. **Use Tags: Tag AWS resources to identify them easily for automation and cleanup.**
2. **Secure Credentials: Use AWS Secrets Manager, Ansible Vault, or environment variables for managing credentials.**
3. **Dynamic Inventory: Leverage dynamic inventory for large-scale environments.**
4. **Test Changes: Always test playbooks in a non-production environment.**
5. **Log Changes: Enable logging for compliance and troubleshooting.**

### Additional Resources

* **Ansible AWS Documentation**
* **AWS Modules Reference**