Cloud Infotech Solutions Academy

Training / IT Consultant

8688253560==========

Ansible playbook

These playbooks cover basic concepts like installing packages, managing files, configuring services, and more. By Cloud infotech

---

1. Install and Start Apache

Playbook: `install_apache.yml`

yaml

```
---
- name: Install and Start Apache
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
```


```
    state: present

- name: Start and Enable Apache Service
  service:
    name: apache2
    state: started
    enabled: true
```

What it Does:

1. Installs Apache on Debian-based systems.
2. Ensures the Apache service is started and enabled to run on boot.

---

2. Create a File

Playbook: `create_file.yml`

yaml

```
---
- name: Create a File
```


```
hosts: all

tasks:

  - name: Create a file with some content

    copy:

      dest: /tmp/hello_world.txt

      content: "Hello, Ansible!"
```

What it Does:

1. Creates a file named `hello_world.txt` in the `/tmp` directory.
2. Writes the text "Hello, Ansible!" into the file.

---

3. Add Users

Playbook: `add_users.yml`

yaml

```
---

- name: Add Users

  hosts: all
```


```
become: yes

tasks:

  - name: Create users

    user:

      name: "{{ item }}"

      state: present

  with_items:

    - alice

    - bob
```

What it Does:

1. Adds two users: `alice` and `bob`.
2. Uses a loop to simplify the task.

---

4. Configure NGINX

Playbook: `configure_nginx.yml`

yaml

Cloud Infotech Solutions Academy

Training / IT Consultant

8688253560==========

```
---
- name: Install and Configure NGINX
  hosts: webservers
  become: yes
  tasks:
    - name: Install NGINX
      apt:
        name: nginx
        state: present

    - name: Start and Enable NGINX
      service:
        name: nginx
        state: started
        enabled: true

    - name: Create a simple HTML file
```


```
    copy:
      dest: /var/www/html/index.html
      content: "<h1>Welcome to NGINX</h1>"
```

What it Does:

1. Installs NGINX.
2. Starts and enables the NGINX service.
3. Creates a simple HTML file to serve as the default page.

---

5. Manage Multiple Packages

Playbook: `install_packages.yml`

yaml

```
---
- name: Install Multiple Packages
  hosts: all
  become: yes
  tasks:
```


```
  - name: Install common tools
    apt:
      name: "{{ item }}"
      state: present
    with_items:
      - curl
      - git
      - unzip
```

What it Does:

1. Installs a list of common tools (`curl`, `git`, `unzip`).
2. Uses a loop to iterate over the package names.

---

6. Deploy a Template

Playbook: `deploy_template.yml`

yaml

```
---
```

Cloud Infotech Solutions Academy

Training / IT Consultant

8688253560==========

```
- name: Deploy a Configuration File from Template
  hosts: webservers
  become: yes
  vars:
    server_name: example.com
  tasks:
    - name: Deploy virtual host configuration
      template:
        src: templates/vhost.conf.j2
        dest: /etc/nginx/sites-available/{{ server_name }}.conf

    - name: Enable site configuration
      command: ln -s /etc/nginx/sites-available/{{ server_name }}.conf
/etc/nginx/sites-enabled/
      args:
        creates: /etc/nginx/sites-enabled/{{ server_name }}.conf

    - name: Reload NGINX
```


```
    service:
      name: nginx
      state: reloaded
```

Template File: `templates/vhost.conf.j2`

nginx

```
server {
    listen 80;
    server_name {{ server_name }};
    root /var/www/{{ server_name }};
    index index.html;
}
```

What it Does:

1. Deploys a dynamic virtual host configuration for NGINX.
2. Reloads NGINX to apply changes.

---

7. Set Up a Basic LAMP Stack

Playbook: `lamp_stack.yml`

yaml

```
---
- name: Install LAMP Stack
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache, MySQL, and PHP
      apt:
        name: "{{ item }}"
        state: present
      with_items:
        - apache2
        - mysql-server
        - php
```


```
      - php-mysql

  - name: Start and Enable Apache
    service:
      name: apache2
      state: started
      enabled: true

  - name: Start and Enable MySQL
    service:
      name: mysql
      state: started
      enabled: true
```

What it Does:

1. Installs Apache, MySQL, and PHP for a basic LAMP stack.
2. Ensures both Apache and MySQL services are started and enabled.

8. Conditional Task Execution

Playbook: `conditional_tasks.yml`

yaml

```yaml
---
- name: Execute Tasks Based on Conditions
  hosts: all
  tasks:
    - name: Install Apache on Debian
      apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"

    - name: Install httpd on Red Hat
      yum:
        name: httpd
```


```
    state: present

  when: ansible_os_family == "RedHat"
```

What it Does:

1. Installs `apache2` on Debian-based systems.
2. Installs `httpd` on Red Hat-based systems.

---

9. Debugging Variables

Playbook: `debug_variables.yml`

yaml

```
---
- name: Debug Variables
  hosts: all
  tasks:
    - name: Display the OS family
      debug:
        msg: "The OS family is {{ ansible_os_family }}"
```


```
  - name: Show all variables
    debug:
      var: ansible_facts
```

What it Does:

1. Prints the operating system family.
2. Displays all available Ansible facts for the host.

---

10. Create a Directory

Playbook: `create_directory.yml`

yaml

```
---
- name: Create a Directory
  hosts: all
  become: yes
  tasks:
```


```
- name: Create a directory
  file:
    path: /opt/my_directory
    state: directory
    mode: '0755'
```

What it Does:

1. Creates a directory named `/opt/my_directory` with permissions set to `0755`.

These examples introduce key concepts like:

- Installing packages.
- Managing files and directories.
- Deploying templates.
- Using loops (`with_items`) and conditionals (`when`).

Here are **Ansible modules examples** to set up **LAMP stack**, **WordPress**, and other common applications. These examples demonstrate how to use Ansible modules to automate application deployments.

1. LAMP Stack Setup

Playbook for LAMP Stack:

yaml

```
---
- name: Setup LAMP Stack
  hosts: webservers
  become: yes
  vars:
    server_name: example.com
    document_root: /var/www/html
  tasks:
    - name: Install required packages
      apt:
        name: "{{ item }}"
        state: present
      with_items:
```


```
      - apache2
      - mysql-server
      - php
      - libapache2-mod-php
      - php-mysql

  - name: Start and enable Apache
    service:
      name: apache2
      state: started
      enabled: true

  - name: Start and enable MySQL
    service:
      name: mysql
      state: started
      enabled: true
```


```
- name: Set up document root
  file:
    path: "{{ document_root }}"
    state: directory
    mode: '0755'

- name: Deploy Apache Virtual Host
  template:
    src: vhost.conf.j2
    dest: /etc/apache2/sites-available/{{ server_name }}.conf
  notify: Restart Apache

- name: Enable site configuration
  command: a2ensite {{ server_name }}
  notify: Restart Apache
```


```
    - name: Create a PHP test page
      copy:
        dest: "{{ document_root }}/info.php"
        content: |
          <?php
          phpinfo();
          ?>
  handlers:
    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

Template (`templates/vhost.conf.j2`):

apache

```
<VirtualHost *:80>
    ServerName {{ server_name }}
```


```
    DocumentRoot {{ document_root }}
    <Directory {{ document_root }}>
        AllowOverride All
    </Directory>
</VirtualHost>
```

---

2. WordPress Setup

Playbook for WordPress Deployment:

yaml

```
---
- name: Deploy WordPress
  hosts: webservers
  become: yes
  vars:
    wp_db_name: wordpress
```


```
    wp_db_user: wp_user
    wp_db_password: secretpassword
    wp_db_host: localhost
    wp_url: https://wordpress.org/latest.tar.gz
    document_root: /var/www/html
  tasks:
    - name: Install required packages
      apt:
        name: "{{ item }}"
        state: present
      with_items:
        - apache2
        - mysql-server
        - php
        - php-mysql
        - wget
        - unzip
```


```
- name: Download WordPress
  get_url:
    url: "{{ wp_url }}"
    dest: /tmp/wordpress.tar.gz

- name: Extract WordPress
  unarchive:
    src: /tmp/wordpress.tar.gz
    dest: "{{ document_root }}"
    remote_src: yes

- name: Set permissions on WordPress directory
  file:
    path: "{{ document_root }}"
    state: directory
    mode: '0755'
```


```
    recurse: yes

- name: Create WordPress database
  mysql_db:
    name: "{{ wp_db_name }}"
    state: present

- name: Create WordPress database user
  mysql_user:
    name: "{{ wp_db_user }}"
    password: "{{ wp_db_password }}"
    priv: "{{ wp_db_name }}.*:ALL"
    state: present

- name: Configure WordPress wp-config.php
  template:
    src: wp-config.php.j2
```


```
        dest: "{{ document_root }}/wp-config.php"
```

Template (`templates/wp-config.php.j2`):

php

```
<?php
define( 'DB_NAME', '{{ wp_db_name }}' );
define( 'DB_USER', '{{ wp_db_user }}' );
define( 'DB_PASSWORD', '{{ wp_db_password }}' );
define( 'DB_HOST', '{{ wp_db_host }}' );
define( 'DB_CHARSET', 'utf8' );
define( 'DB_COLLATE', '' );

$table_prefix = 'wp_';

define( 'WP_DEBUG', false );

if ( ! defined( 'ABSPATH' ) ) {
```


```
    define( 'ABSPATH', dirname( __FILE__ ) . '/' );
}

require_once ABSPATH . 'wp-settings.php';
```

---

3. Most Common Apps

NGINX

yaml

```
- name: Install and Configure NGINX
  hosts: webservers
  become: yes
  tasks:
    - name: Install NGINX
      apt:
        name: nginx
```


```
      state: present

  - name: Start and enable NGINX
    service:
      name: nginx
      state: started
      enabled: true

  - name: Configure NGINX site
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/sites-available/default
    notify: Reload NGINX

handlers:
  - name: Reload NGINX
    service:
```


```
        name: nginx
        state: reloaded
```

Docker

yaml

```
- name: Install Docker
  hosts: servers
  become: yes
  tasks:
    - name: Install prerequisites
      apt:
        name:
          - apt-transport-https
          - ca-certificates
          - curl
          - software-properties-common
        state: present
```


```
    - name: Add Docker GPG key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: Add Docker repository
      apt_repository:
        repo: deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable
        state: present

    - name: Install Docker
      apt:
        name: docker-ce
        state: present

    - name: Ensure Docker is started
```


```
    service:
      name: docker
      state: started
      enabled: true
```

Node.js

yaml

```
- name: Install Node.js
  hosts: servers
  become: yes
  tasks:
    - name: Add NodeSource APT repository
      apt_repository:
        repo: deb https://deb.nodesource.com/node_14.x focal main
        state: present

    - name: Install Node.js
```


```
    apt:
      name: nodejs
      state: present
```

These playbooks demonstrate modular approaches to automate common application installations using Ansible modules. Adjust the variables as needed for your environment!

Ansible roles are a structured way to organize Ansible tasks, handlers, variables, templates, and other resources. Roles help in reusability and modularity of your playbooks. Below is a step-by-step guide on how to create an Ansible role for setting up a **LAMP stack** (Linux, Apache, MySQL, PHP):

Step 1: Create a Role

You can create a role manually or use the `ansible-galaxy` command:

bash

```
ansible-galaxy init lamp
```

This will create a directory structure like this:

CSS

```
lamp/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Step 2: Define Role Tasks

Edit the `tasks/main.yml` file to include steps for setting up the LAMP stack.

yaml

```
---
- name: Install Apache
  apt:
    name: apache2
    state: present
```


```
  notify: Restart Apache

- name: Install MySQL server
  apt:
    name: mysql-server
    state: present

- name: Install PHP and related modules
  apt:
    name: "{{ item }}"
    state: present
  with_items:
    - php
    - php-mysql
    - libapache2-mod-php

- name: Enable Apache rewrite module
  command: a2enmod rewrite
  notify: Restart Apache

- name: Create a test PHP file
  copy:
    dest: /var/www/html/info.php
    content: |
      <?php
      phpinfo();
      ?>
```

Step 3: Add Handlers

Handlers are tasks that are triggered by notifications. Define a handler to restart Apache.

`handlers/main.yml`:
yaml

```yaml
---
- name: Restart Apache
  service:
    name: apache2
    state: restarted
```

Step 4: Define Variables (Optional)

Define any default variables in `defaults/main.yml` or specific ones in `vars/main.yml`.

`defaults/main.yml`:
yaml

```yaml
---
php_packages:
  - php
  - php-mysql
  - libapache2-mod-php
```

Update `tasks/main.yml` to use these variables:
yaml


```
- name: Install PHP and related modules
  apt:
    name: "{{ item }}"
    state: present
  with_items: "{{ php_packages }}"
```

Step 5: Include Templates or Files

You can store configuration files or templates in the `templates/` or `files/` directories.

Example: Apache Virtual Host Template

Save the file as `templates/vhost.conf.j2`:

apache

```
<VirtualHost *:80>
    DocumentRoot /var/www/html
    ServerName {{ server_name }}

    <Directory /var/www/html>
        AllowOverride All
    </Directory>
</VirtualHost>
```

Add a Task to Use the Template

yaml

```
- name: Deploy Apache virtual host configuration
```


```
  template:
    src: vhost.conf.j2
    dest: /etc/apache2/sites-available/000-default.conf
  notify: Restart Apache
```

Step 6: Use the Role in a Playbook

Create a playbook (`site.yml`) to use the role.

yaml

```
---
- hosts: webservers
  become: yes
  roles:
    - lamp
```

Step 7: Run the Playbook

Ensure you have an inventory file defining the `webservers` group and run the playbook:

bash

```
ansible-playbook -i inventory site.yml
```

Final Notes

- Replace `apt` with `yum` if you're working on a Red Hat-based system.
- Customize the role further by adding additional configurations for MySQL (e.g., secure installation) or PHP (e.g., php.ini settings).
- Test the role using Ansible Molecule or in a virtualized environment.

===

Creating Ansible roles manually involves setting up a directory structure and organizing your playbook components such as tasks, handlers, templates, files, and variables. Below is a step-by-step guide to create Ansible roles manually:

Step 1: Understand the Role Directory Structure

An Ansible role typically follows a specific directory structure:

css

```
roles/
└── <role_name>/
    ├── tasks/
    │   └── main.yml
    ├── handlers/
    │   └── main.yml
    ├── templates/
    ├── files/
    ├── vars/
    │   └── main.yml
    ├── defaults/
    │   └── main.yml
    ├── meta/
```


```
|   └── main.yml
└── tests/
    ├── inventory
    └── test.yml
```

- **tasks/**: Contains the main tasks of the role.
- **handlers/**: Defines tasks triggered by notifications (e.g., service restarts).
- **templates/**: Stores Jinja2 template files for configuration files.
- **files/**: Stores static files to be copied to the target machine.
- **vars/**: Defines role-specific variables.
- **defaults/**: Contains default variables with the lowest precedence.
- **meta/**: Contains metadata about the role (dependencies, author, etc.).
- **tests/**: Contains tests to verify the role.

Step 2: Manually Create the Directory Structure

Create the base directory for your role:

bash

```
mkdir -p
roles/<role_name>/{tasks,handlers,templates,files,vars,defaults,meta,t
ests}
```

Replace `<role_name>` with the name of your role.

Step 3: Populate the Directories

Here's how to populate each directory:

1. tasks/main.yml

Define the main tasks to be executed in this file. For example:

yaml

```yaml
---
- name: Install Apache
  apt:
    name: apache2
    state: present

- name: Start and enable Apache service
  service:
    name: apache2
    state: started
    enabled: true
```

2. handlers/main.yml

Define any tasks that should be notified and run at the end of a playbook run. For example:

yaml

```yaml
---
- name: Restart Apache
  service:
    name: apache2
    state: restarted
```

3. templates/

Store Jinja2 template files here. For example, create a template `vhost.conf.j2`:

apache

```
<VirtualHost *:80>
    DocumentRoot /var/www/html
    ServerName {{ server_name }}
</VirtualHost>
```

4. files/

Place static files here, such as a static `index.html` file:

csharp

```
Hello, World! This is a static file served by Apache.
```

5. vars/main.yml

Store variables that are specific to the role:

yaml

```
---
server_name: localhost
```

6. defaults/main.yml

Define default variables with low precedence:

yaml

```
---
apache_packages:
  - apache2
```

7. meta/main.yml

Provide metadata about the role:

yaml

```
---
dependencies: []
```

8. tests/ (Optional)

Create test files to verify the role functionality. For example: **`tests/inventory`**:

ini

```
localhost ansible_connection=local
```

`tests/test.yml`:

yaml

```
---
- hosts: localhost
  roles:
    - <role_name>
```

Step 4: Use the Role in a Playbook

Define a playbook to apply the role:

site.yml:

yaml

```
---
- hosts: webservers
  become: yes
  roles:
    - <role_name>
```

Step 5: Run the Playbook

Ensure you have an inventory file and run the playbook:

bash

```
ansible-playbook -i inventory site.yml
```

Additional Notes

- When organizing manually, always follow the directory structure for clarity and compatibility with Ansible's role framework.
- Use meaningful names for variables and templates to make your role reusable.

- Test your role in different environments (e.g., staging, production) to ensure it works as intended.

====

Ansible's `with_*` and `when` plugins enable dynamic task execution and conditional logic, respectively. Here's a collection of examples showcasing their usage:

---

1. Using `with_*` Loops

Example: Install Multiple Packages

yaml

```yaml
- name: Install required packages
  apt:
    name: "{{ item }}"
    state: present
  with_items:
    - apache2
    - mysql-server
    - php
    - libapache2-mod-php
```

Example: Create Multiple Users

yaml


```
- name: Create users
  user:
    name: "{{ item.name }}"
    state: present
    shell: "{{ item.shell }}"
  with_items:
    - { name: alice, shell: /bin/bash }
    - { name: bob, shell: /bin/zsh }
```

---

2. Using when Conditionals

Example: Conditional Package Installation

yaml

```
- name: Install Apache on Debian-based systems
  apt:
    name: apache2
    state: present
  when: ansible_os_family == "Debian"

- name: Install httpd on Red Hat-based systems
  yum:
    name: httpd
    state: present
  when: ansible_os_family == "RedHat"
```

Example: Check Service Status

yaml

```
- name: Ensure Apache is started if installed
  service:
    name: apache2
    state: started
  when: "'apache2' in ansible_facts.packages"
```

---

3. Combining `with_*` and `when`

Example: Create Users Based on Conditions

yaml

```
- name: Create users only if they are admins
  user:
    name: "{{ item.name }}"
    state: present
    shell: /bin/bash
  with_items:
    - { name: admin1, is_admin: true }
    - { name: user1, is_admin: false }
  when: item.is_admin
```

---

4. Using `with_items` and `when` for Files

Example: Copy Files Based on a Condition

yaml

```yaml
- name: Copy configuration files if on Debian
  copy:
    src: "{{ item }}"
    dest: /etc/myapp/
  with_items:
    - config1.conf
    - config2.conf
  when: ansible_os_family == "Debian"
```

---

5. Using `with_dict`

Example: Configure Multiple Services

yaml

```yaml
- name: Configure services
  service:
    name: "{{ item.key }}"
    state: "{{ item.value }}"
  with_dict:
    apache2: started
    mysql: restarted
```

6. Using `with_nested`

Example: Install Packages for Multiple Hosts

yaml

```
- name: Install packages for each host
  apt:
    name: "{{ item[1] }}"
    state: present
  with_nested:
    - "{{ groups['webservers'] }}"
    - [apache2, mysql-server]
```

---

7. Using `with_fileglob`

Example: Copy All Templates from a Directory

yaml

```
- name: Copy all templates
  template:
    src: "{{ item }}"
    dest: /etc/myapp/templates/
  with_fileglob:
    - templates/*.j2
```

8. Using `with_sequence`

Example: Create Multiple Directories

yaml

```
- name: Create directories
  file:
    path: /var/app/{{ item }}
    state: directory
  with_sequence: start=1 end=5
```

---

9. Using `when` with Facts

Example: Perform Actions Based on OS Version

yaml

```
- name: Install packages for specific Ubuntu versions
  apt:
    name: apache2
    state: present
  when: ansible_distribution == "Ubuntu" and
ansible_distribution_version == "20.04"
```

---

10. Advanced Example: Combine `when` and Loops

Example: Configure Services Based on OS

yaml

```yaml
- name: Install services based on the OS
  service:
    name: "{{ item }}"
    state: started
  with_items:
    - apache2
    - mysql
  when: ansible_os_family == "Debian"

- name: Install services on Red Hat
  service:
    name: "{{ item }}"
    state: started
  with_items:
    - httpd
    - mariadb
  when: ansible_os_family == "RedHat"
```

Key Points:

- `with_*` plugins loop over lists, dictionaries, files, sequences, etc., to simplify repetitive tasks.
- `when` enables conditional execution of tasks, handlers, or roles.
- You can combine `with_*` and `when` for dynamic and flexible automation workflows.

Cloud Infotech Solutions Academy

Training / IT Consultant

8688253560==========