Activity 7: Managing Files and Creating Roles in Ansible	
Name: Atian, Catherine Joy D.	**Date Performed: Sep 19, 2025**
Course/Section: CPE212 - CPE31S4	**Date Submitted: Sep 19, 2025**
Instructor: Engr. Robin Valenzuela	**Semester and SY: 1st SEM, A.Y. 2025 - 2026**

Objective:

Describe the step-by-step execution and expected behaviors of Ansible tasks across Ubuntu and CentOS nodes within a controlled environment without manually executing it.

Introduction:

Before any playbook is executed, the control node must be properly configured and set up. Ansible should be installed and verified using the command ansible --version. The SSH keys must be generated and distributed to all managed nodes to enable passwordless or having secure access. The project directory should follow a modular structure, containing the main playbook which is the site.yml, a files/ directory for static content like default_site.html, and a roles/ directory with subfolders for each functional groups like web_servers, db_servers, workstations, etc.

Procedure:

Task 1: Copying a File to Remote Web Servers

This task involves deploying a static HTML file to both Ubuntu and CentOS web servers. The goal is to ensure that /var/www/html/index.html contains the same content across all nodes, enabling consistent web page delivery.

```
- name: copy default html file for site
  hosts: web_servers
  become: true
  tasks:
    - name: Copy default HTML file
      copy:
        src: default_site.html
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: '0644'
      tags: [apache, apache2, httpd]
```

What Happens

When executed, Ansible connects to both the Ubuntu and CentOS nodes via SSH. It first checks whether the destination file already exists and if it matches the source file.

If the file is missing or different, Ansible proceeds to copy the file from the control node to the managed node and applies the specified permissions and ownership settings.

How It Happens

The control node runs the copy module remotely on each target host. Ansible's idempotent design ensures that the task only updates the file if changes are required—preventing unnecessary modifications. Once in place, the file becomes accessible to the web server, making it available for users to view through a browser.

What the Code Does

The code targets all hosts within the web_servers group and uses become: true to gain administrative privileges, allowing it to perform file operations that require root access. It utilizes the copy module in Ansible to transfer the default_site.html file to the web server's root directory. Once copied, it sets the file ownership to root and applies file permissions of 0644, ensuring that the file is readable by everyone but only writable by the owner.

Result

On the Ubuntu node, running the command cat /var/www/html/index.html displays the HTML content of the deployed webpage. On the CentOS node, accessing the server's IP address through a browser renders the same webpage, confirming that the deployment was successful and the site is publicly accessible.

- *If I were to run the Ansible code for Task 1, the playbook would connect to both Ubuntu and CentOS servers listed under the web_servers group. The goal here is to copy a file called default_site.html from the control node to the /var/www/html/index.html path on each managed node. This file would serve as the default webpage, so when someone visits the server's IP address, they'd see the content I wrote in that HTML file. The playbook uses the copy module to handle this file transfer and sets the correct ownership and permissions so that the web server can read and serve the file properly. On Ubuntu, I'd be glad to see the HTML content by running cat /var/www/html/index.html, and on CentOS, I'd expect the webpage to load in a browser when I type the server's IP.*

Task 2: Installing Terraform on Workstations

This task installs Terraform on the Ubuntu workstation by downloading and extracting the binary from HashiCorp's official release.

```
- name: install terraform
  hosts: workstations
  become: true
  tasks:
    - name: Install unzip
      package:
```

```
            name: unzip
        - name: Download and extract Terraform
          unarchive:
            src:
    https://releases.hashicorp.com/terraform/0.12.28/terraform_0.1
    2.28_linux_amd64.zip
            dest: /usr/local/bin
            remote_src: yes
            mode: '0755'
            owner: root
            group: root
```

What Happens

During execution, Ansible uses the system's package manager, apt, to install the unzip utility if it isn't already present. It then fetches the Terraform zip file directly from its source and extracts it on the remote node. The extracted binary is placed in the system's path, making Terraform accessible from any terminal without specifying a full file path.

How It Happens

The unarchive module in Ansible manages both the downloading and extraction process remotely on the Ubuntu node. Once the extraction is complete, Ansible ensures that the Terraform binary has executable permissions and is owned by root. To verify the success of the installation, running the terraform command on the node should display Terraform's version and available commands.

What the Code Does

The code targets the workstations group, specifically the Ubuntu node, to automate the installation of Terraform. It first installs unzip as a prerequisite tool needed to extract compressed files. After that, it downloads the Terraform package and extracts it into the /usr/local/bin directory. Finally, it sets the appropriate executable permissions and ownership so that Terraform can be run system-wide by any user.

Result

After the playbook runs, Terraform is successfully installed and configured on the Ubuntu workstation. It's now ready for use in performing infrastructure automation tasks, such as provisioning, managing, and deploying cloud and on-premise resources efficiently.

- *For Task 2, if I were to run the Ansible code, it would install Terraform on the Ubuntu workstation listed under the workstations group. The playbook first installs the unzip package, which is needed to extract the Terraform binary. Then it downloads the Terraform zip file from HashiCorp's official release page and extracts it to /usr/local/bin, making it executable and accessible system-wide.*

Ansible would handle this process smoothly using the package and unarchive modules. It would ensure that unzip is installed before attempting to extract the file, and it would apply the correct permissions so that the Terraform binary can be run by any user. After the playbook runs, I'd hope to be able to type terraform on the Ubuntu workstation and see the version and usage instructions, confirming that the installation was successful.

Task 3: Creating and Implementing Roles

Roles modularize tasks for better organization, reuse, and scalability. Each role corresponds to a functional group and contains its own tasks/main.yml.

Directory Structure

```
mkdir -p
roles/{base,web_servers,file_servers,db_servers,workstations}/
tasks
```

roles/web_servers/tasks/main.yml

```
- name: Copy default HTML file
  copy:
    src: default_site.html
    dest: /var/www/html/index.html
    owner: root
    group: root
    mode: '0644'
```

roles/workstations/tasks/main.yml

```
- name: Install unzip
  package:
    name: unzip

- name: Download and extract Terraform
  unarchive:
    src:
https://releases.hashicorp.com/terraform/0.12.28/terraform_0.
12.28_linux_amd64.zip
    dest: /usr/local/bin
    remote_src: yes
    mode: '0755'
    owner: root
    group: root
```

site.yml

```
- hosts: web_servers
  roles:
    - web_servers

- hosts: workstations
  roles:
    - workstations
```

What the Code Does

The code defines roles for each host group, organizing tasks and configurations into modular sections. It automatically loads tasks from the main.yml file located inside each role's directory. Each host in the inventory runs only the tasks assigned to its group, allowing Ansible to execute operations based on the inventory group efficiently.

What Happens

When the playbook runs, Ansible matches each host to its designated role and executes the corresponding tasks. The output is grouped by role, making it easier to read, understand, and debug the results. This organization helps identify which part of the deployment belongs to which role, providing clear visibility into the automation process.

How It Happens

The playbook references roles using the roles that are directive. Ansible dynamically loads and applies each role's tasks during execution. This modular structure allows the playbook to handle multiple configurations and actions automatically, depending on which roles are assigned to specific host groups.

Result

The outcome is a cleaner and more scalable playbook structure that's easier to maintain and extend. Roles promote reusability, allowing the same configuration logic to be applied across different environments with minimal adjustments.

- *If I were to run the Ansible code for Task 3, I'd first restructure the playbook by creating roles for each server group. This means creating folders like roles/web_servers, roles/workstations, and so on, each with a tasks/main.yml file that contains the relevant tasks. Instead of writing everything in one big playbook, I'd reference these roles in site.yml, which makes the code cleaner and easier to maintain. When the playbook runs, Ansible would match each host to its role and execute the tasks defined in that role's main.yml. For example, the web_servers role would handle copying the HTML file, while the workstations role would handle installing Terraform. This modular approach improves scalability and makes it easier to debug or update specific tasks later. The output would be grouped by role, showing which tasks ran on which servers.*

Reflection:

Ansible playbooks execute from top to bottom, running tasks sequentially within each host group. Every task calls a module that enforces a specific desired state. Before applying any change, Ansible checks the current state to ensure actions are only taken when needed. If a task fails on one node, it doesn't affect others unless explicitly configured to stop. The output provides a per-host summary, showing success, failure, and skipped tasks. Using privilege escalation (become) enables root-level operations where necessary. By encapsulating logic into roles, Ansible keeps playbooks modular, reusable, and easy to maintain.

Creating roles significantly simplifies playbook management and promotes reusability across multiple projects. Managing files and configurations through roles ensures consistency and a uniform user experience across nodes. Built-in error handling helps detect issues such as unreachable hosts or missing files, providing clear feedback. Because of idempotency, rerunning the playbook is safe, it won't cause unnecessary or duplicate changes. Additionally, parallel execution allows Ansible to orchestrate multiple nodes at once, improving efficiency.

All in all, if I were to run the Ansible code on both managed nodes without actually doing it, I'd expect everything to work as long as the setup is correct. The control node would connect to each server, apply the tasks based on their group, and ensure that files are copied and software is installed consistently. Using roles makes the playbook easier to manage, especially when working with different types of servers. Ansible's idempotent behavior means I can run the playbook multiple times without worrying about breaking anything and it only makes changes when needed. Even if something goes wrong, like a missing file or a server that's offline, Ansible would show me exactly where the error happened so I can fix it quickly. Overall, this setup would allow me to manage both Ubuntu and CentOS nodes efficiently, without having to log into each one manually.

References:

https://spacelift.io/blog/ansible-tutorial
https://dev.to/kennibravo/ansible-tutorial-automate-and-configure-remote-linux-servers-3l9
https://forum.ansible.com/t/how-to-make-and-run-ansible-playbooks-in-centos-7/33792
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html
https://stackoverflow.com/questions/28550619/ansible-playbook-to-execute-commands-from-user-shell
https://www.digitalocean.com/community/tutorials/how-to-use-ansible-to-automate-initial-server-setup-on-ubuntu-20-04
https://labex.io/tutorials/ansible-how-to-execute-an-ansible-playbook-on-the-control-node-417418