

Name: Potestades, North Nygel G.	Date Performed: 9/5/25
Course/Section: CPE31S2	Date Submitted: 9/12/25
Instructor: Engr. Robin Valenzuela	Semester and SY: 1st Semester 2025-2026

Activity 5: Consolidating Playbook plays

1. Objectives:

- 1.1 Use **when** command in playbook for different OS distributions
- 1.2 Apply refactoring techniques in cleaning up the playbook codes

2. Discussion:

We are going to look at a way that we can differentiate a playbook by a host in terms of which distribution the host is running. It's very common in most Linux shops to run multiple distributions, for example, Ubuntu shop or Debian shop and you need a different distribution for a one off-case or perhaps you want to run plays only on certain distributions.

It is a best practice in ansible when you are working in a collaborative environment to use the command git pull. git pull is a Git command used to update the local version of a repository from a remote. By default, git pull does two things. Updates the current local working branch (currently checked out branch) and updates the remote-tracking branches for all other branches. git pull essentially pulls down any changes that may have happened since the last time you worked on the repository.

Requirement:

In this activity, you will need to create a CentOS VM. Likewise, you need to activate the second adapter to a host-only adapter after the installations. Take note of the IP address of the CentOS VM. Make sure to use the command **ssh-copy-id** to copy the public key to CentOS. Verify if you can successfully SSH to CentOS VM.

Task 1: Use when command for different distributions

1. In the local machine, make sure you are in the local repository directory (**CPE232_yourname**). Issue the command git pull. When prompted, enter the correct passphrase or password. Describe what happened when you issue this command. Did something happen? Why?

```
north@workstation:~/CPE212_Potestades$ git pull
Already up to date.
```

It displayed "Already up to date.", because there have been no updates to the git repository that need to be downloaded.

2. Edit the inventory file and add the IP address of the Centos VM. Issue the command we used to execute the playbook (the one we used in the last activity): **ansible-playbook --ask-become-pass install_apache.yml**. After executing this command, you may notice that it did not become successful in the Centos VM. You can see that the Centos VM has failed=1. Only the two remote

servers have been changed. The reason is that Centos VM does not support "apt" as the package manager. The default package manager for Centos is "yum."

3. Edit the *install_apache.yml* file and insert the lines shown below.

```
---
- hosts: all
  become: true
  tasks:

    - name: update repository index
      apt:
        update_cache: yes
        when: ansible_distribution == "Ubuntu"

    - name: install apache2 package
      apt:
        name: apache2
        when: ansible_distribution == "Ubuntu"

    - name: add PHP support for apache
      apt:
        name: libapache2-mod-php
        when: ansible_distribution == "Ubuntu"
```

Make sure to save the file and exit.

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```
ok: [192.168.56.107]
ok: [192.168.56.105]
ok: [192.168.56.106]
```

The result is that it runs fine now, with all of them being ok. However, nothing is changed on the CentOS distribution.

If you have a mix of Debian and Ubuntu servers, you can change the configuration of your playbook like this.

```
- name: update repository index
  apt:
    update_cache: yes
    when: ansible_distribution in ["Debian", "Ubuntu"]
```

Note: This will work also if you try. Notice the changes are highlighted.

4. Edit the *install_apache.yml* file and insert the lines shown below.

```

---
- hosts: all
  become: true
  tasks:

    - name: update repository index
      apt:
        update_cache: yes
      when: ansible_distribution == "Ubuntu"

    - name: install apache2 package
      apt:
        name: apache2
        state: latest
      when: ansible_distribution == "Ubuntu"

    - name: add PHP support for apache
      apt:
        name: libapache2-mod-php
        state: latest
      when: ansible_distribution == "Ubuntu"

    - name: update repository index
      dnf:
        update_cache: yes
      when: ansible_distribution == "CentOS"

    - name: install apache2 package
      dnf:
        name: httpd
        state: latest
      when: ansible_distribution == "CentOS"

    - name: add PHP support for apache
      dnf:
        name: php
        state: latest
      when: ansible_distribution == "CentOS"

```

Make sure to save and exit.

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```

PLAY RECAP *****
192.168.56.107 : ok=4    changed=2    unreachable=0    failed=0    s
kipped=3      rescued=0    ignored=0
192.168.56.112 : ok=4    changed=3    unreachable=0    failed=0    s
kipped=3      rescued=0    ignored=0

```

The result is as expected, where the tasks indicated for the Ubuntu VMs are skipped by the CentOS VM, and vice-versa. They each have 3 skipped, as they each have 3 tasks which are not meant for them.

5. To verify the installations, go to CentOS VM and type its IP address on the browser. Was it successful? The answer is no. It's because the httpd service or the Apache HTTP server in the CentOS is not yet active. Thus, you need to activate it first.



Unable to connect

An error occurred during a connection to 192.168.56.107.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

Try Again

5.1 To activate, go to the CentOS VM terminal and enter the following:

systemctl status httpd

The result of this command tells you that the service is inactive.

```
[north@centos ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; vendor preset: enabled)
   Drop-In: /usr/lib/systemd/system/httpd.service.d;
           └─php-fpm.conf
   Active: inactive (dead)
   Docs: man:httpd.service(8)
```

5.2 Issue the following command to start the service:

sudo systemctl start httpd

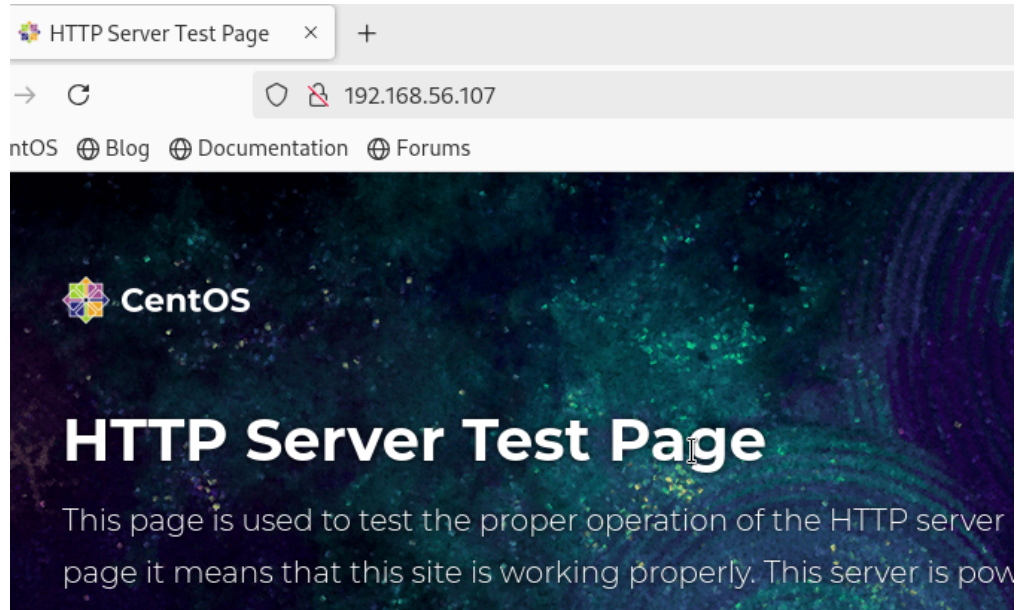
(When prompted, enter the sudo password)

sudo firewall-cmd --add-port=80/tcp

(The result should be a success)

```
[north@centos ~]$ sudo systemctl start httpd
[north@centos ~]$ sudo firewall-cmd --add-port=80/tcp
success
```

5.3 To verify the service is already running, go to CentOS VM and type its IP address on the browser. Was it successful? (Screenshot the browser)



Yes, it was successful.

Task 2: Refactoring playbook

This time, we want to make sure that our playbook is efficient and that the codes are easier to read. This will also make Ansible run more quickly if it has to execute fewer tasks to do the same thing.

1. Edit the playbook *install_apache.yml*. Currently, we have three tasks targeting our Ubuntu machines and 3 tasks targeting our CentOS machine. Right now, we try to consolidate some tasks that are typically the same. For example, we can consolidate two plays that install packages. We can do that by creating a list of installation packages as shown below:

```

---
- hosts: all
  become: true
  tasks:

    - name: update repository index Ubuntu
      apt:
        update_cache: yes
      when: ansible_distribution == "Ubuntu"

    - name: install apache2 and php packages for Ubuntu
      apt:
        name:
          - apache2
          - libapache2-mod-php
        state: latest
      when: ansible_distribution == "Ubuntu"

    - name: update repository index for CentOS
      dnf:
        update_cache: yes
      when: ansible_distribution == "CentOS"

    - name: install apache and php packages for CentOS
      dnf:
        name:
          - httpd
          - php
        state: latest
      when: ansible_distribution == "CentOS"

```

Make sure to save the file and exit.

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```

PLAY RECAP *****
192.168.56.107      : ok=3    changed=0    unreachable=0    failed=0    s
kipped=2    rescued=0    ignored=0
192.168.56.112    : ok=3    changed=1    unreachable=0    failed=0    s
kipped=2    rescued=0    ignored=0

```

The playbook now runs faster, as there are only 2 tasks to do for each distribution, with each task being done successfully.

2. Edit the playbook *install_apache.yml* again. In task 2.1, we consolidated the plays into one play. This time we can actually consolidated everything in just 2 plays. This can be done by removing the update repository play and putting the command *update_cache: yes* below the command *state: latest*. See below for reference:

```

---
- hosts: all
  become: true
  tasks:

    - name: install apache2 and php packages for Ubuntu
      apt:
        name:
          - apache2
          - libapache2-mod-php
        state: latest
        update_cache: yes
      when: ansible_distribution == "Ubuntu"

    - name: install apache and php packages for CentOS
      dnf:
        name:
          - httpd
          - php
        state: latest
        update_cache: yes
      when: ansible_distribution == "CentOS"

```

Make sure to save the file and exit.

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```
PLAY RECAP *****
192.168.56.107      : ok=2    changed=0    unreachable=0    failed=0    s
kipped=1    rescued=0    ignored=0
192.168.56.112      : ok=2    changed=0    unreachable=0    failed=0    s
kipped=1    rescued=0    ignored=0
```

It is now even faster, as they only have 1 task to complete each, with a total of 2 being processed by the control node.

3. Finally, we can consolidate these 2 plays in just 1 play. This can be done by declaring variables that will represent the packages that we want to install. Basically, the `apache_package` and `php_package` are variables. The names are arbitrary, which means we can choose different names. We also take out the line `when: ansible_distribution`. Edit the playbook *install_apache.yml* again and make sure to follow the below image. Make sure to save the file and exit.

```
---
- hosts: all
  become: true
  tasks:
    - name: install apache and php
      apt:
        name:
          - "{{ apache_package }}"
          - "{{ php_package }}"
        state: latest
        update_cache: yes
```

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```
PLAY RECAP *****
192.168.56.107      : ok=1    changed=0    unreachable=0    failed=1    s
kipped=0    rescued=0    ignored=0
192.168.56.112      : ok=1    changed=0    unreachable=0    failed=1    s
kipped=0    rescued=0    ignored=0
```

It was not successful, as there are undefined variables in the playbook.

4. Unfortunately, task 2.3 was not successful. It's because we need to change something in the inventory file so that the variables we declared will be in place. Edit the *inventory* file and follow the below configuration:

```
192.168.56.120 apache_package=apache2 php_package=libapache2-mod-php
192.168.56.121 apache_package=apache2 php_package=libapache2-mod-php
192.168.56.122 apache_package=httpd php_package=php
```

Make sure to save the *inventory* file and exit.

```
192.168.56.112 apache_package=apache2 php_package=libapache2-mod-php
192.168.56.113 apache_package=apache2 php_package=libapache2-mod-php
192.168.56.107 apache_package=httpd php_package=php
```

Finally, we still have one more thing to change in our *install_apache.yml* file. In task 2.3, you may notice that the package is assign as `apt`, which will not run in CentOS. Replace the *apt* with *package*. Package is a module in ansible that is generic, which is going to use whatever

package manager the underlying host or the target server uses. For Ubuntu it will automatically use *apt*, and for CentOS it will automatically use *dnf*. Make sure to save the file and exit. For more details about the ansible package, you may refer to this documentation: [ansible.builtin.package – Generic OS package manager — Ansible Documentation](#)

Run *ansible-playbook --ask-become-pass install_apache.yml* and describe the result.

```
PLAY RECAP *****
192.168.56.107      : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.56.112     : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

It now succeeds, having 2 successful operations, those being the task itself and the gathering facts step which is done whenever a playbook is run.

Supplementary Activity:

1. Create a playbook that could do the previous tasks in Red Hat OS.

```
Shell
---
- hosts: redhat
  become: true
  tasks:

  - name: install apache and php redhat
    dnf:
      name:
        - httpd
        - php
      state: latest
      update_cache: yes
```

Red Hat OS uses the same package managers as CentOS (DNF & YUM) because CentOS is a Red Hat-based distribution of Linux. This means that the previous playbooks would work with Red Hat OS as well.

GitHub Link: [northpotestades/CPE212 Potestades](#)

Reflections:

Answer the following:

1. Why do you think refactoring of playbook codes is important?

Refactoring is important because keeping code concise and compact is always an important part of optimizing code. This makes it so that it doesn't need to do multiple different tasks and keep switching between the managed nodes, and instead once it is done with the tasks it can move on to the next and not go back.

2. When do we use the “when” command in playbook?

We use the when command in a playbook whenever we want to add a conditional statement to the playbook. The example used in the HOA is the distribution, but it can also be used to add a conditional statement about the disk space or memory of the managed node.