Name: Bacong, El Cid A.	**Date Performed:** 9/5/2025
Course/Section: CPE212 - CPE31S4	**Date Submitted:** 9/5/2025
Instructor: Engr. Robin Valenzuela	**Semester and SY:** 1st Sem 2025-2026

Activity 4: Running Elevated Ad hoc Commands

1. **Objectives:**

1.1 Use commands that makes changes to remote machines

1.2 Use playbook in automating ansible commands

2. **Discussion:**

Provide screenshots for each task.

Elevated Ad hoc commands

So far, we have not performed ansible commands that makes changes to the remote servers. We manage to gather facts and connect to the remote machines, but we still did not make changes on those machines. In this activity, we will learn to use commands that would install, update, and upgrade packages in the remote machines. We will also create a playbook that will be used for automations.

Playbooks record and execute **Ansible**'s configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material. At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way. You can check this documentation if you want to learn more about playbooks. Working with playbooks — Ansible Documentation

Task 1: Run elevated ad hoc commands

1. Locally, we use the command *sudo apt update* when we want to download package information from all configured resources. The sources often defined in /etc/apt/sources.list file and other files located in /etc/apt/sources.list.d/ directory. So, when you run update command, it downloads the package information from the Internet. It is useful to get info on an updated version of packages or their dependencies. We can only run

an apt update command in a remote machine. Issue the following command:

ansible all -m apt -a update_cache=true

What is the result of the command? Is it successful?

```
cidee@workstation:~/CPE212_elcid$ ansible all -m apt -a update_cache=true -i inv
entory.ini
[WARNING]: Updating cache and auto-installing missing dependency: python3-apt
192.168.56.104 | FAILED! => {
    "changed": false,
    "cmd": "update",
    "msg": "[Errno 2] No such file or directory: b'update'",
    "rc": 2,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "",
    "stdout_lines": []
}
192.168.56.103 | FAILED! => {
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock directory
 /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)"
}
```

```
cidee@workstation:~/CPE212_elcid$ ansible dbserver -m dnf -a update_cache=true -
i inventory.ini
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "pkg_mgr": "dnf"
    },
    "changed": false,
    "msg": "Cache updated",
    "rc": 0,
    "results": []
}
```

It wasn't since my Ansible inventory file is empty. Ansible has no list of servers to run the command on.

Try editing the command and add something that would elevate the privilege. Issue the command *ansible all -m apt -a update_cache=true --become --ask-become-pass.* Enter the sudo password when prompted.

```
cidee@workstation:~/CPE212_elcid$ ansible webserver -m apt -a update_cache=true
--become --ask-become-pass -i inventory.ini
BECOME password:
192.168.56.103 | CHANGED => {
    "cache_update_time": 1757686575,
    "cache_updated": true,
    "changed": true
}
cidee@workstation:~/CPE212_elcid$ ansible dbserver -m dnf -a update_cache=true -
-become --ask-become-pass -i inventory.ini
BECOME password:
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "pkg_mgr": "dnf"
    },
    "changed": false,
    "msg": "Cache updated",
    "rc": 0,
    "results": []
}
```

You will notice now that the output of this command is a success. The *update_cache=true* is the same thing as running *sudo apt update*. The --become command elevate the privileges and the *--ask-become-pass* asks for the password. For now, even if we only have changed the packaged index, we were able to change something on the remote server.

You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

2. Let's try to install VIM, which is an almost compatible version of the UNIX editor Vi. To do this, we will just changed the module part in 1.1 instruction. Here is the command: *ansible all -m apt -a name=vim-nox --become --ask-become-pass.*

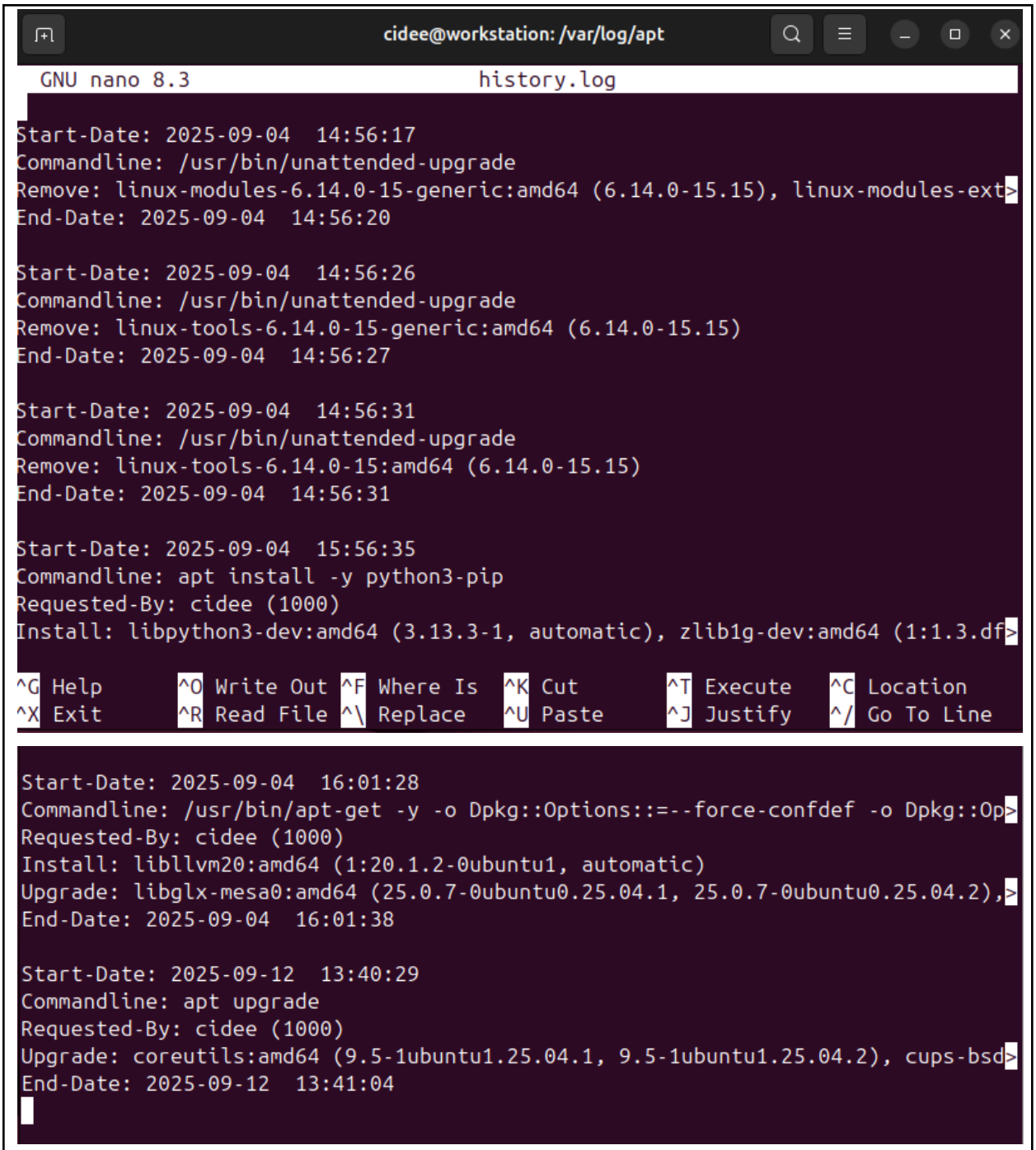
```
cidee@workstation:~/CPE212_elcid$ ansible dbserver -m dnf -a name=vim-enhanced -
-become --ask-become-pass
BECOME password:
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "pkg_mgr": "dnf"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
cidee@workstation:~/CPE212_elcid$ ansible webserver -m apt -a name=vim-nox --bec
ome --ask-become-pass
BECOME password:
192.168.56.103 | SUCCESS => {
    "cache_update_time": 1757687468,
    "cache_updated": false,
    "changed": false
}
```

The command would take some time after typing the password because the local machine instructed the remote servers to actually install the package.

2.1 Verify that you have installed the package in the remote servers. Issue the command *which vim* and the command *apt search vim-nox* respectively. Was the command successful?

```
cidee@workstation:~/CPE212$ which vim
cidee@workstation:~/CPE212$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/bionic-updates,bionic-security 2:8.0.1453-1ubuntu1.13 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/bionic-updates,bionic-security,now 2:8.0.1453-1ubuntu1.13 amd64 [insta
lled]
  Vi IMproved - enhanced vi editor - compact version
```

2.2 Check the logs in the servers using the following commands: *cd /var/log*. After this, issue the command *ls,* go to the folder *apt* and open history.log. Describe what you see in the history.log.



3. This time, we will install a package called snapd. Snap is pre-installed in Ubuntu system. However, our goal is to create a command that checks for the latest installation package.

   3.1 Issue the command: *ansible all -m apt -a name=snapd --become --ask-become-pass*

Can you describe the result of this command? Is it a success? Did it change anything in the remote servers? ***No change happened***

```
cidee@workstation:~/CPE212_elcid$ ansible webserver -m apt -a name=snapd --becom
e --ask-become-pass
BECOME password:
192.168.56.103 | SUCCESS => {
    "cache_update_time": 1757687468,
    "cache_updated": false,
    "changed": false
}
cidee@workstation:~/CPE212_elcid$ ansible dbserver -m dnf -a name=snapd --become
 --ask-become-pass
BECOME password:
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "pkg_mgr": "dnf"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
```

3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*

Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

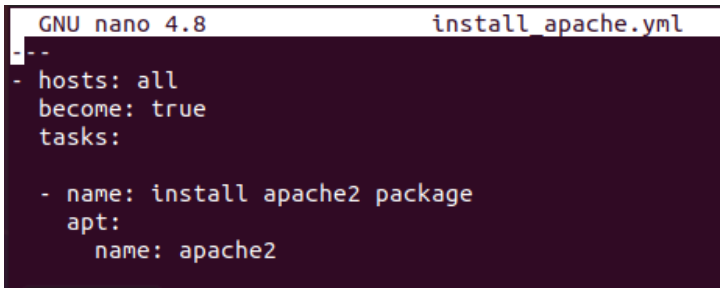
```
cidee@workstation:~/CPE212_elcid$ ansible webserver -m apt -a "name=snapd state=
latest" --become --ask-become-pass
BECOME password:
192.168.56.103 | SUCCESS => {
    "cache_update_time": 1757687468,
    "cache_updated": false,
    "changed": false
}
cidee@workstation:~/CPE212_elcid$ ansible dbserver -m dnf -a "name=snapd state=l
atest" --become --ask-become-pass
BECOME password:
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "pkg_mgr": "dnf"
    },
    "changed": false,
    "msg": "Nothing to do",
    "rc": 0,
    "results": []
}
```

4. At this point, make sure to commit all changes to GitHub.

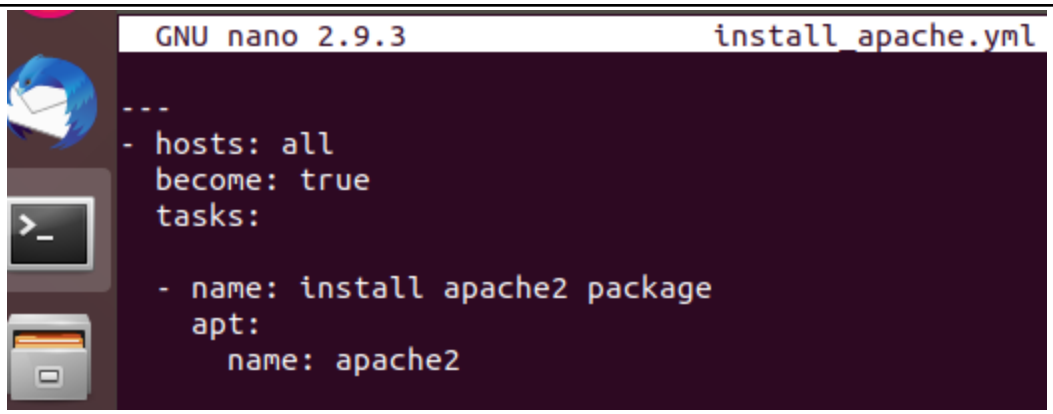
Task 2: Writing our First Playbook

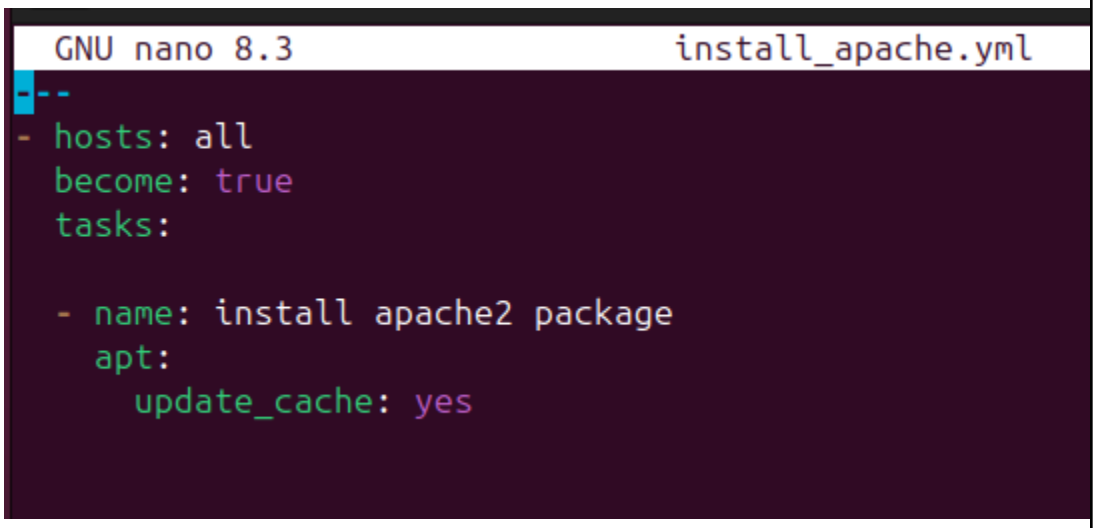
1. With ad hoc commands, we can simplify the administration of remote servers. For example, we can install updates, packages, and applications, etc. However, the real strength of ansible comes from its playbooks. When we write a playbook, we can define the state that we want our servers to be in and the place or commands that ansible will carry out to bring to that state. You can use an editor to create a playbook. Before we proceed, make sure that you are in the directory of the repository that we use in the previous activities (*CPE232_yourname*). Issue the command *nano install_apache.yml*. This will create a playbook file called *install_apache.yml*. The .yml is the basic standard extension for playbook files.

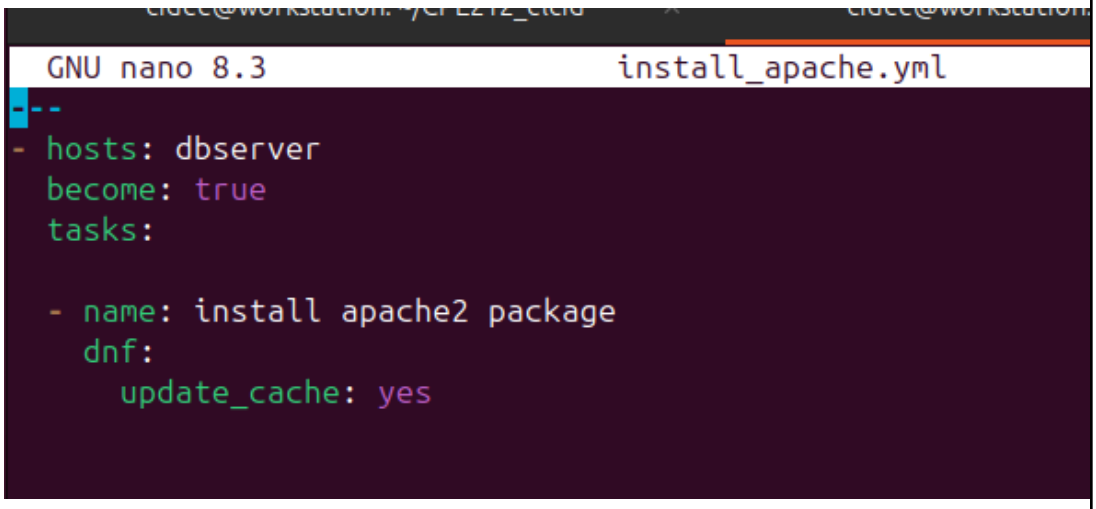
   When the editor appears, type the following:



   Make sure to save the file. Take note also of the alignments of the texts.





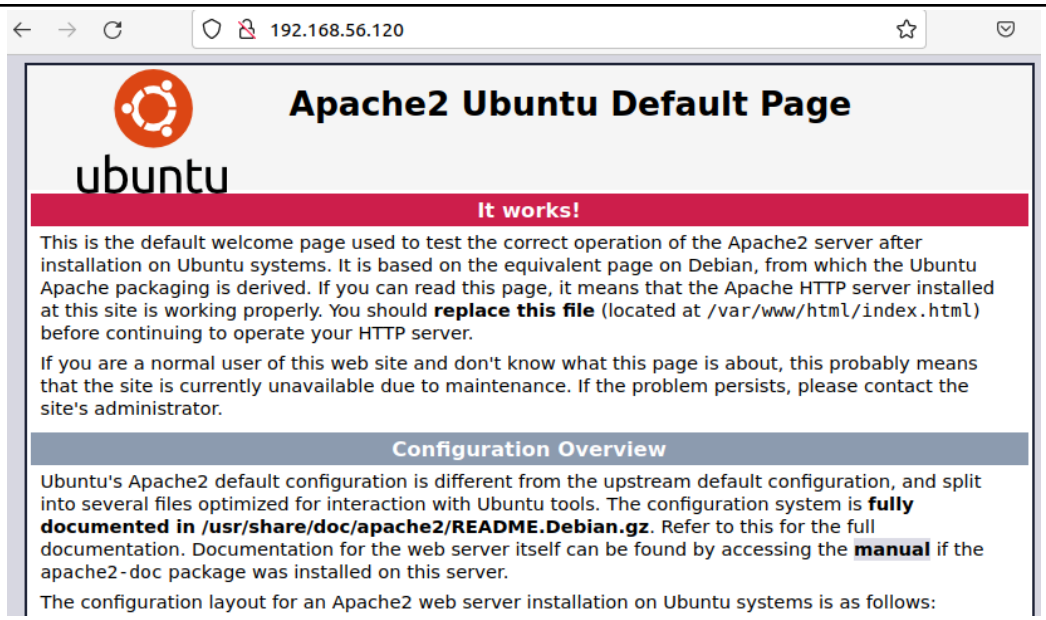


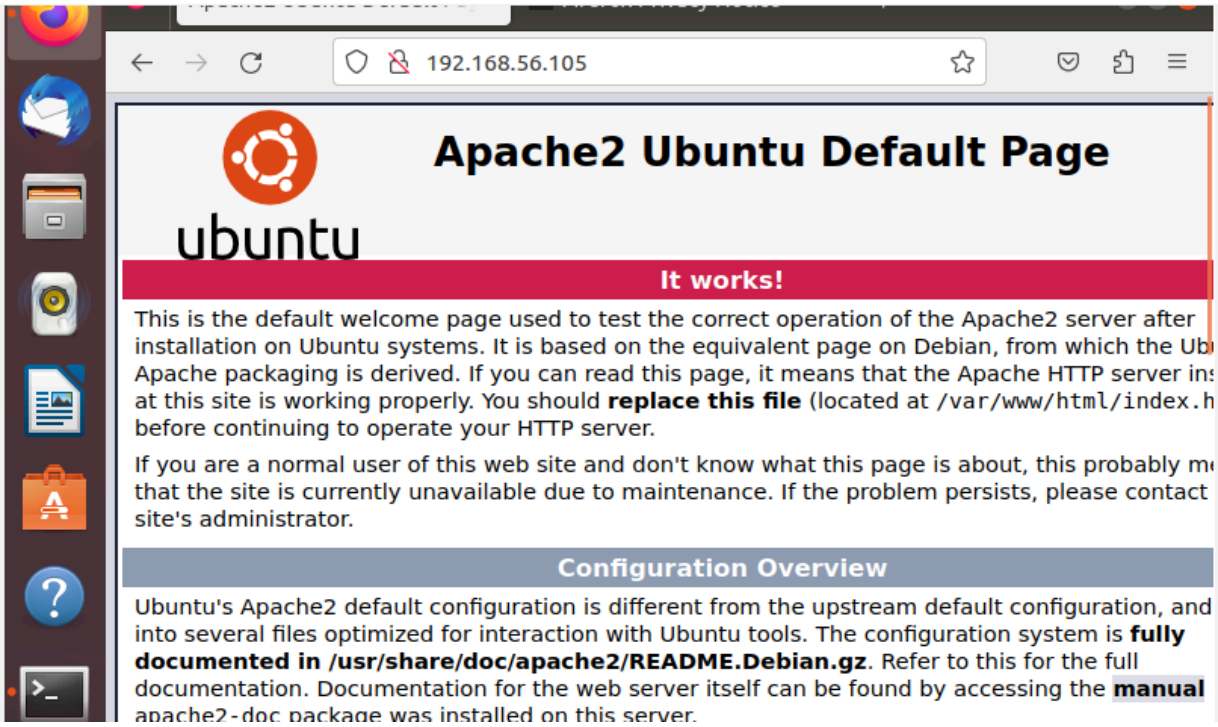
2. Run the yml file using the command: *ansible-playbook --ask-become-pass install_apache.yml*. Describe the result of this command.

```
cidee@workstation:~/CPE212$ ansible-playbook --ask-become-pass install_apache.y
ml
SUDO password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]

TASK [install apache2 package] *************************************************
*
changed: [192.168.56.105]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=2    changed=1    unreachable=0    failed=0
```

```
cidee@workstation:~/CPE212_elcid$ ansible-playbook --ask-become-pass install_apa
che.yml
BECOME password:

PLAY [dbserver] ****************************************************************

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.104]

TASK [install apache2 package] *************************************************
ok: [192.168.56.104]

PLAY RECAP *********************************************************************
192.168.56.104             : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

cidee@workstation:~/CPE212_elcid$ sudo nano install_apache.yml
cidee@workstation:~/CPE212_elcid$
```

3. To verify that apache2 was installed automatically in the remote servers, go to the web browsers on each server and type its IP address. You should see something like this.





4. Try to edit the *install_apache.yml* and change the name of the package to any name that will not be recognized. What is the output?

```
cidee@workstation:~/CPE212$ ansible-playbook --ask-become-pass install_apache.y
ml
SUDO password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]

TASK [install apache2 package] *************************************************
*
fatal: [192.168.56.105]: FAILED! => {"changed": false, "msg": "No package match
ing 'peache2' is available"}
        to retry, use: --limit @/home/cidee/CPE212/install_apache.retry

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=1    changed=0    unreachable=0    failed=1
```

5. This time, we are going to put additional task to our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional command, which is the *update_cache.* This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities) that were being installed.

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

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
```

Save the changes to this file and exit.

```
 GNU nano 2.9.3                    install_apache.yml

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

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
```

6. Run the playbook and describe the output. Did the new command change anything on the remote servers?

```
cidee@workstation:~/CPE212$ ansible-playbook --ask-become-pass install_apache.y
ml
SUDO password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]

TASK [update repository index] *************************************************
*
changed: [192.168.56.105]

TASK [install apache2 package] *************************************************
*
ok: [192.168.56.105]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=3    changed=1    unreachable=0    failed=0
```

7. Edit again the *install_apache.yml*. This time, we are going to add a PHP support for the apache package we installed earlier.

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

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2

  - name: add PHP support for apache
    apt:
      name: libapache2-mod-php
```

Save the changes to this file and exit.

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

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2

  - name: add PHP support for apache
    apt:
      name: libapache2-mod-php
```

8. Run the playbook and describe the output. Did the new command change anything on the remote servers?

```
cidee@workstation:~/CPE212$ ansible-playbook --ask-become-pass install_apache.y
ml
SUDO password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]

TASK [update repository index] *************************************************
*
changed: [192.168.56.105]

TASK [install apache2 package] *************************************************
*
ok: [192.168.56.105]

TASK [add PHP support for apache] **********************************************
*
changed: [192.168.56.105]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=4    changed=2    unreachable=0    failed=0
```

9. Finally, make sure that we are in sync with GitHub. Provide the link of your GitHub repository.

https://github.com/moussecake22/CPE212_elcid.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?
   A playbook is important because it allows automation and consistency in managing remote servers. Instead of typing ad hoc commands repeatedly, a playbook saves the tasks in a file, making it easier to reuse, share, and ensure that all machines follow the same configuration. It also helps avoid human error and makes system administration more efficient.
2. Summarize what we have done on this activity.

In this activity, I was able to run elevated ad hoc commands in Ansible to update, upgrade, and install packages on remote servers. I observed how commands would fail at first but succeeded when I used --become for privilege escalation. I installed packages like vim-nox and snapd, then verified their installation and checked the logs. After that, I created my first playbook to install Apache, tested it, and later modified it to include updating the cache and adding PHP support. Finally, I committed and synced everything to GitHub to keep track of my progress.