Name: Cruz, Patrick Danielle C.	**Date Performed: Aug 26, 2025**
Course/Section: CPE212 - CPE31S4	**Date Submitted: Aug 25, 2025**
Instructor: Engr. Robin Valenzuela	**Semester and SY: 1st semester**

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 make changes to the remote servers. We managed 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

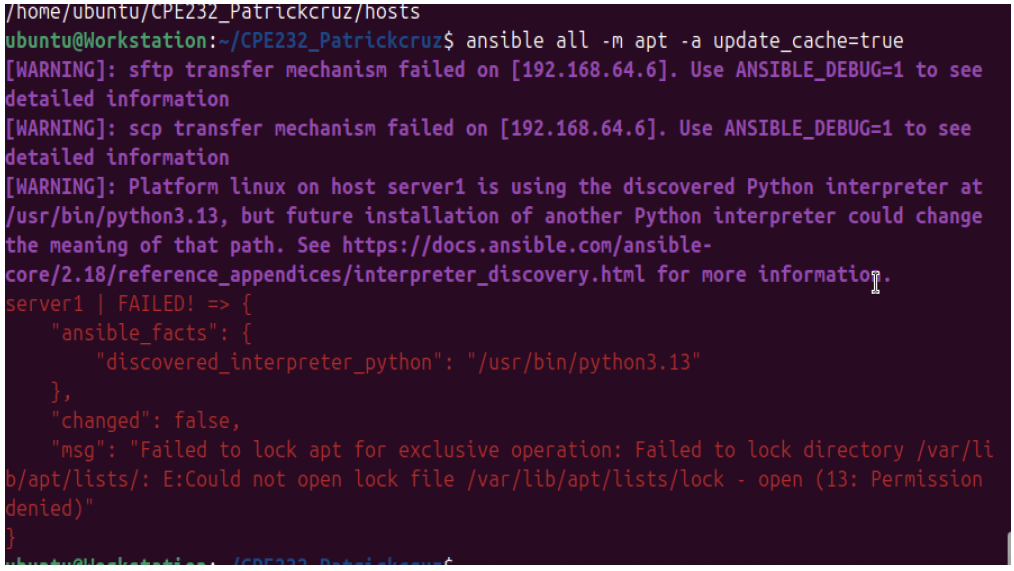
Task 1: Run elevated ad hoc commands

1. Locally, we use the command *sudo apt update* when we want to download package information from all configured resources. The sources are often defined in /etc/apt/sources.list file and other files located in /etc/apt/sources.list.d/ directory. So, when you run the 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?

- Is it successful? No, it is not successful. The command failed with a "Permission denied" error because apt update requires sudo (root) privileges to run on the remote server, but the command was executed without the --become option to elevate privileges.



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. 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 elevates the privileges and the *--ask-become-pass* asks for the password. For now, even if we only have changed the

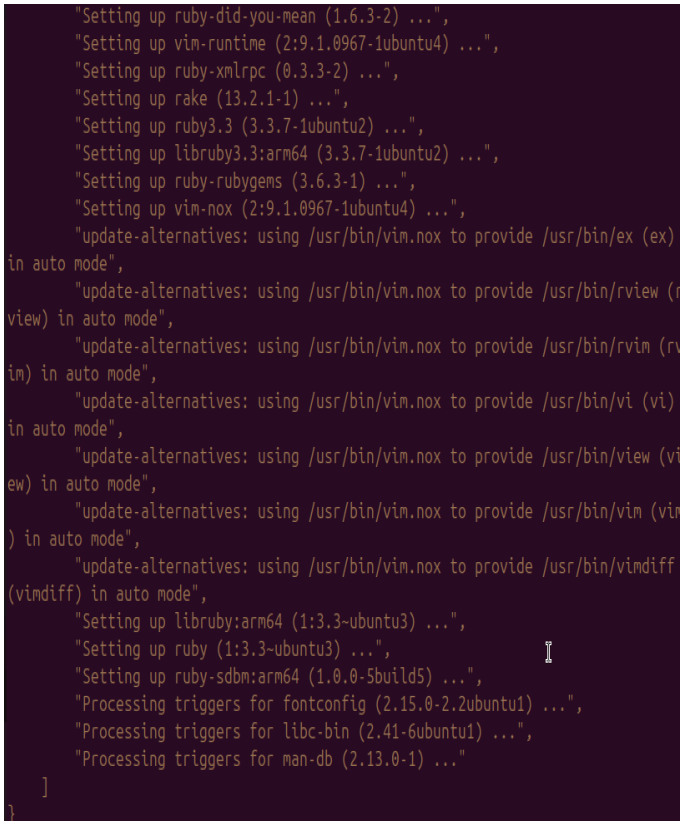
packaged index, we were able to change something on the remote server.

```
ubuntu@Workstation:~/CPE232_Patrickcruz$ ansible all -m apt -a update_cache=true --becom
e --ask-become-pass
BECOME password:
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1 to see
detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1 to see
detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python interpreter at
/usr/bin/python3.13, but future installation of another Python interpreter could change
the meaning of that path. See https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
server1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.13"
    },
    "cache_update_time": 1756207427,
    "cache_updated": true,
    "changed": true
}
ubuntu@Workstation:~/CPE232_Patrickcruz$
```

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

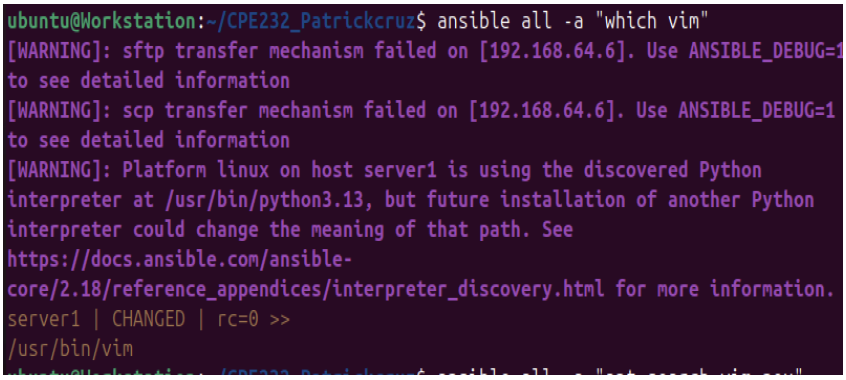
2. Let's try to install VIM, which is an almost compatible version of the UNIX editor Vi. To do this, we will just change the module part in 1.1 instruction. Here is the command: *ansible all -m apt -a name=vim-nox --become --ask-become-pass.* The command would take some time after typing the password because the local machine instructed the remote servers to actually install the package.

```
ubuntu@Workstation:~/CPE232_Patrickcruz$ ansible all -m apt -a name=vim-nox --bec
ome --ask-become-pass
BECOME password:
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
server1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.13"
    },
    "cache_update_time": 1756207427,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading sta
te information...\nSolving dependencies...\nThe following additional packages wil
l be installed:\n  fonts-lato javascript-common libjs-jquery liblua5.1-0 libruby
libruby3.3\n  rake ruby ruby-did-you-mean ruby-minitest ruby-net-telnet ruby-powe
r-assert\n  ruby-rubygems ruby-sdbm ruby-test-unit ruby-webrick ruby-xmlrpc ruby3
.3\n  rubygems-integration vim-runtime\nSuggested packages:\n  apache2 | lighttpd
```



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?

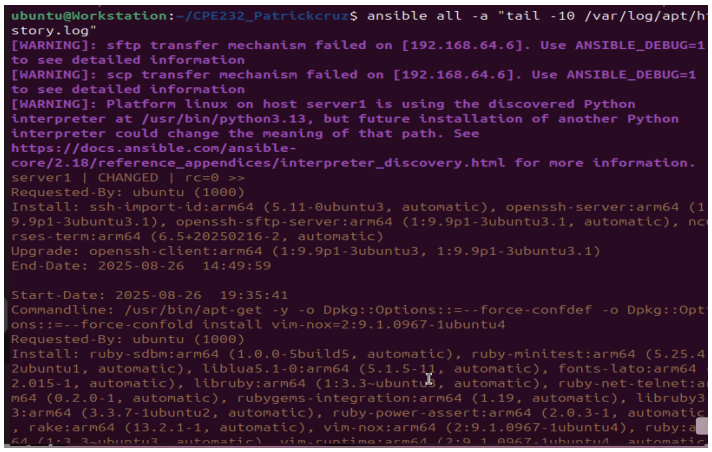
- Yes, the command was completely successful.



```
ubuntu@Workstation:~/CPE232_Patrickcruz$ ansible all -a "apt search vim-nox"
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
server1 | CHANGED | rc=0 >>
Sorting...
Full Text Search...
vim-nox/plucky,now 2:9.1.0967-1ubuntu4 arm64 [installed]
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/plucky,now 2:9.1.0967-1ubuntu4 arm64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

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.

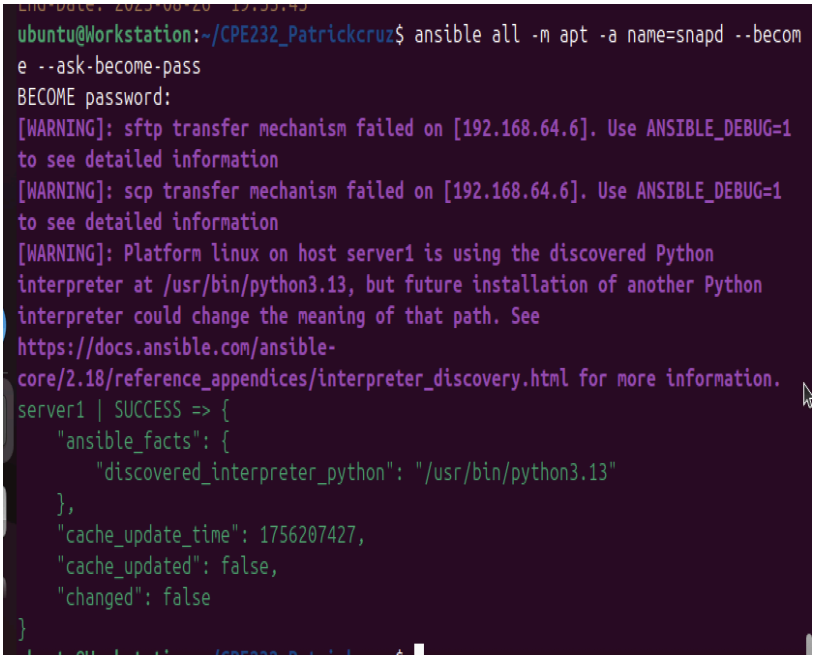
- The APT history log shows that the first session was mainly for setting up SSH by installing packages like openssh-server and ssh-import-id. Later, in the second session, VIM-nox was installed with its exact version, and this also pulled in several dependencies like Ruby and Lua libraries, development tools, fonts, and the VIM runtime. Together, these entries confirm the successful setup of both SSH access and a fully functional VIM-nox editor on the server.



3. This time, we will install a package called snapd. Snap is pre-installed in the 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?

- The command worked fine, but Ansible didn't need to do anything because snapd was already installed. That's why it showed "changed: false" and "SUCCESS." It just means everything is okay, and the server already has what you asked for. The warnings don't matter, so the task was successful.

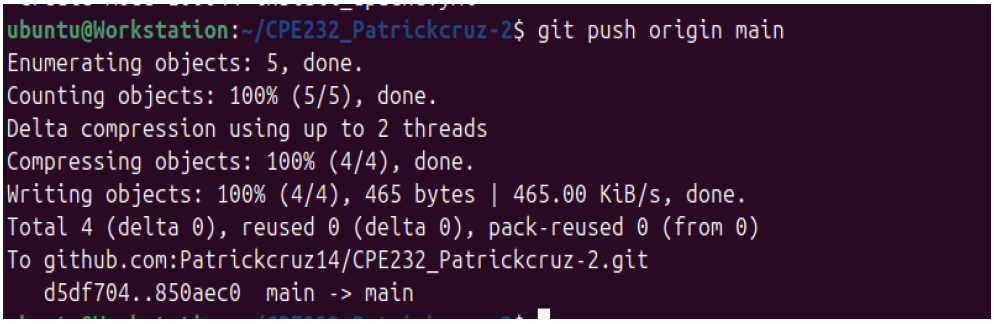


3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*
Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

```
ubuntu@Workstation:~/CPE232_Patrickcruz$ ansible all -m apt -a "name=snapd state=
latest" --become --ask-become-pass
BECOME password:
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
server1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.13"
    },
    "cache_update_time": 1756207427,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading sta
te information...\nSolving dependencies...\nThe following packages will be upgrad
ed:\n  snapd\n1 upgraded, 0 newly installed, 0 to remove and 285 not upgraded.\nN
eed to get 32.0 MB of archives.\nAfter this operation, 11.9 MB of additional di
 space will be used.\nGet:1 http://ports.ubuntu.com/ubuntu-ports plucky-updates/m
```

```
        "(Reading database ... 30%",
        "(Reading database ... 35%",
        "(Reading database ... 40%",
        "(Reading database ... 45%",
        "(Reading database ... 50%",
        "(Reading database ... 55%",
        "(Reading database ... 60%",
        "(Reading database ... 65%",
        "(Reading database ... 70%",
        "(Reading database ... 75%",
        "(Reading database ... 80%",
        "(Reading database ... 85%",
        "(Reading database ... 90%",
        "(Reading database ... 95%",
        "(Reading database ... 100%",
        "(Reading database ... 216973 files and directories currently installed.)
",
        "Preparing to unpack .../snapd_2.68.5+ubuntu25.04.2_arm64.deb ...",
        "Unpacking snapd (2.68.5+ubuntu25.04.2) over (2.67.1+25.04) ...",
        "Setting up snapd (2.68.5+ubuntu25.04.2) ...",
        "snapd.failure.service is a disabled or a static unit not running, not st
arting it.",
        "snapd.snap-repair.service is a disabled or a static unit not running, no
t starting it.",
        "Processing triggers for gnome-menus (3.36.0-1.1ubuntu3) ...",
        "Processing triggers for man-db (2.13.0-1) ...",
        "Processing triggers for dbus (1.16.2-2ubuntu1) ...",
        "Processing triggers for desktop-file-utils (0.28-1) ..."
    ]
}
```

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


ubuntu@Workstation:~/CPE232_Patrickcruz-2$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 465 bytes | 465.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Patrickcruz14/CPE232_Patrickcruz-2.git
d5df704..850aec0 main -> main

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:

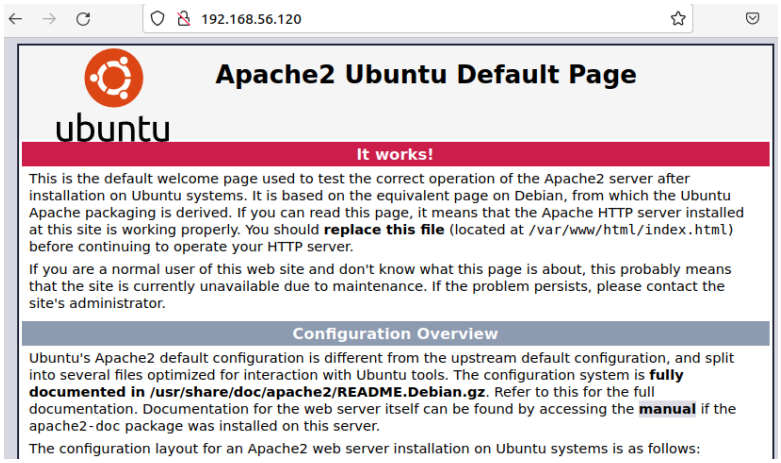
```
  GNU nano 4.8                  install_apache.yml
---
- hosts: all
  become: true
  tasks:

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

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.

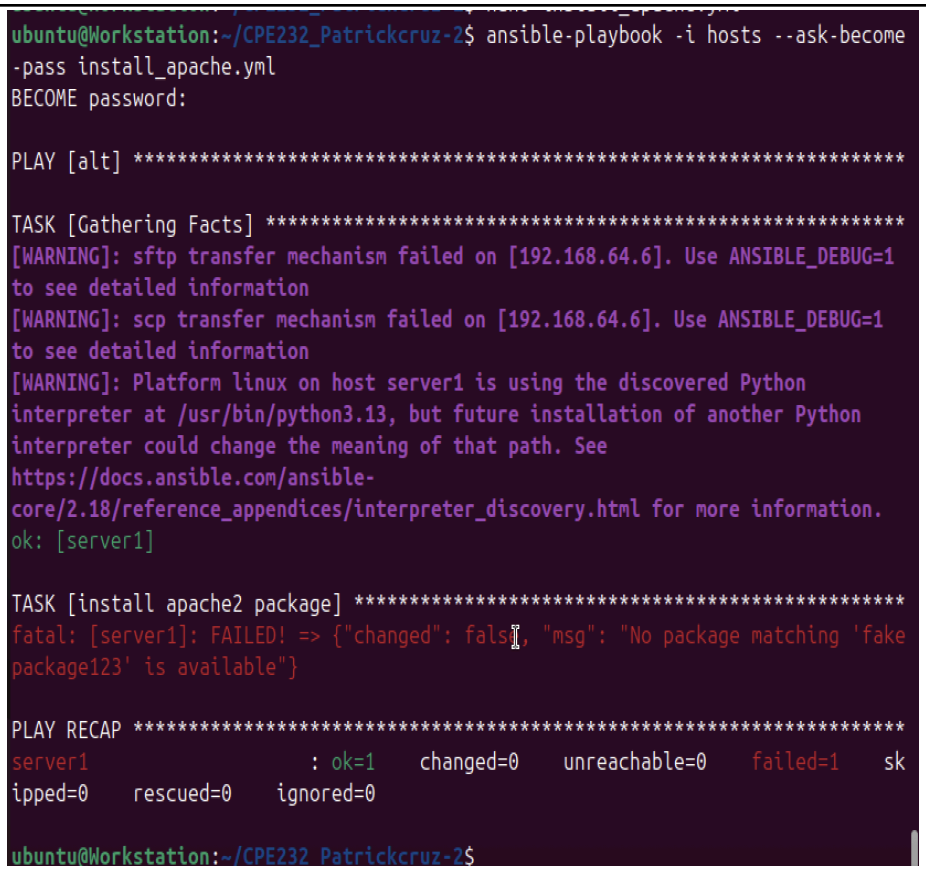
```
ERROR! the playbook: install could not be found
ubuntu@Workstation:~/CPE232_Patrickcruz-2$ ansible-playbook -i hosts --ask-become
-pass install_apache.yml
BECOME password:

PLAY [alt] *********************************************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
ok: [server1]

TASK [install apache2 package] *************************************************
changed: [server1]

PLAY RECAP *********************************************************************
server1                    : ok=2    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0

ubuntu@Workstation:~/CPE232_Patrickcruz-2$
```

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?

- **It failed**



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.

```
ubuntu@Workstation:~/CPE232_Patrickcruz-2$ nano install_apache.yml
ubuntu@Workstation:~/CPE232_Patrickcruz-2$ ansible-playbook -i hosts --ask-become
-pass install_apache.yml
BECOME password:

PLAY [alt] *********************************************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
ok: [server1]

TASK [update repository index] *************************************************
changed: [server1]

TASK [install apache2 package] *************************************************
ok: [server1]

PLAY RECAP *********************************************************************
server1                    : ok=3    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=0
```

```
---
- 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.

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

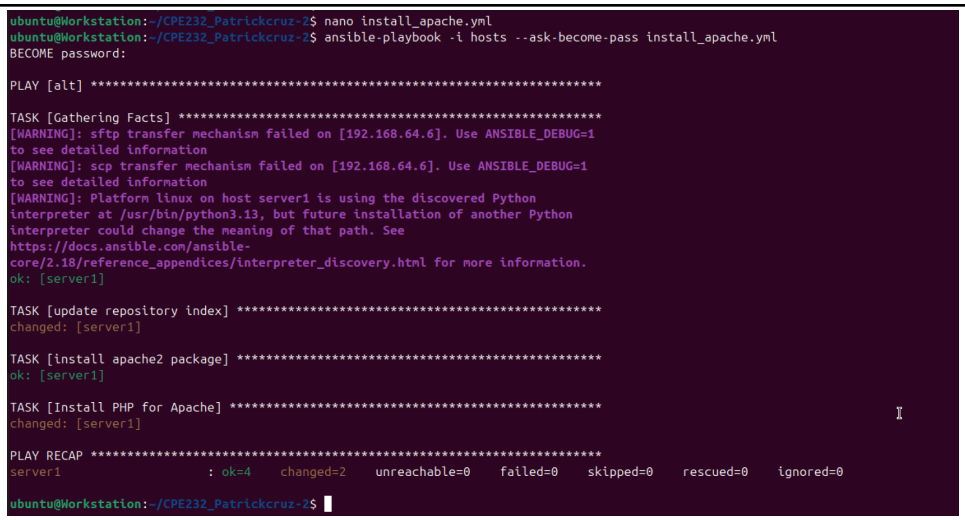
- Yes, the new command made a change on the server. The update_cache: yes updated the package list, just like running apt update, so it showed CHANGED. The Apache2 part didn't change anything because it was already installed, so it showed OK. Basically, the playbook updated the repository and made sure everything is up-to-date before installing.

```
ubuntu@Workstation:~/CPE232_Patrickcruz-2$ nano install_apache.yml
ubuntu@Workstation:~/CPE232_Patrickcruz-2$ ansible-playbook -i hosts --ask-become
-pass install_apache.yml
BECOME password:

PLAY [alt] *********************************************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: sftp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: scp transfer mechanism failed on [192.168.64.6]. Use ANSIBLE_DEBUG=1
to see detailed information
[WARNING]: Platform linux on host server1 is using the discovered Python
interpreter at /usr/bin/python3.13, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.18/reference_appendices/interpreter_discovery.html for more information.
ok: [server1]

TASK [update repository index] *************************************************
changed: [server1]

TASK [install apache2 package] *************************************************
ok: [server1]

PLAY RECAP *********************************************************************
server1                    : ok=3    changed=1    unreachable=0    failed=0    sk
ipped=0    rescued=0    ignored=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.



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

- Yes, the new command really did change things on the servers. The package list was updated, Apache stayed the same since it was already installed, and PHP was newly installed and set up for Apache. The recap shows everything worked: 4 tasks ran successfully, and 2 of them made actual changes. This confirms the playbook is doing its job of keeping the system updated and adding PHP support.

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

- https://github.com/Patrickcruz14/CPE232_Patrickcruz-2/blob/main/hosts

Reflections:

Answer the following:

1. What is the importance of using a playbook?

- A playbook in Ansible is important because it makes work easier, faster, and more consistent. Instead of doing long and complicated tasks step by step on each server, you can write them once in a playbook and run them all at once. This not only saves time but also keeps everything the same across all servers, avoiding mistakes. A playbook also works like a guide or

documentation since it clearly shows how your system should look, like making sure Apache and PHP are installed. The best part is that it's safe to run a playbook multiple times because it will only change things if needed, making your setup reliable and efficient.

2. Summarize what we have done on this activity.

- In this activity, I started by using ad-hoc commands with command privileges to handle basic administrative tasks on remote servers, like updating packages and installing vim-nox and snapd. This taught me the importance of the --become flag whenever sudo access is needed. From there, I moved on to using playbooks, which felt more powerful and efficient. I created an install_apache.yml playbook to automatically set up Apache, then improved it by adding steps to update the package cache and install PHP support. What stood out to me was how playbooks turn multiple manual steps into one clean, repeatable process that saves time and avoids errors. To make my work more professional and organized, I also pushed everything to GitHub, which gave me a sense of real-world workflow and version control.