Name: Reyes, Alexzander J.	Date Performed: 05/09/2025
Course/Section: CPE212-CPE31S2	Date Submitted: 05/09/2025
Instructor: Engr. Robin Valenzuela	Semester and SY: 1st & 2025-202

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.

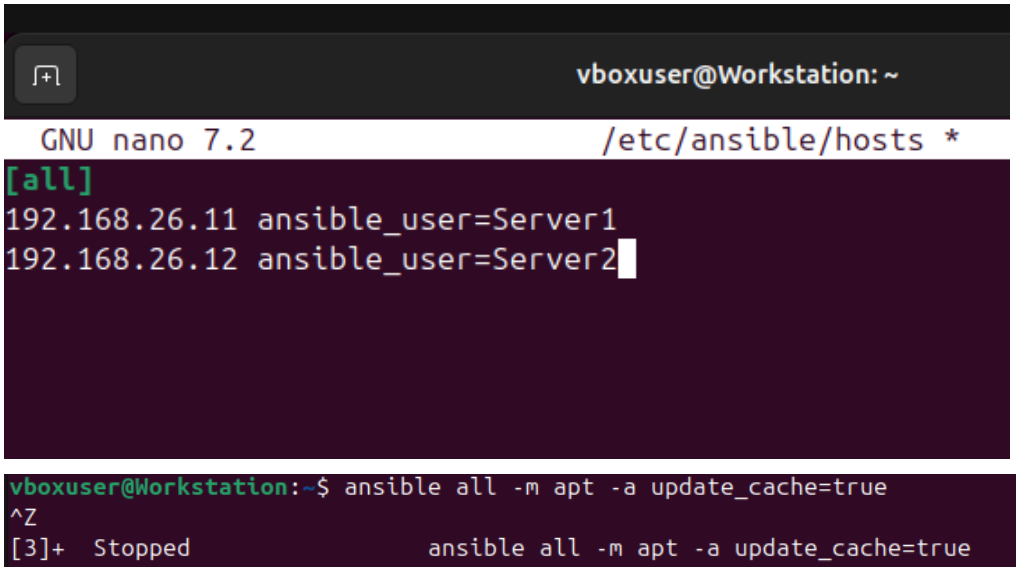
Playbooks record and execute **Ansible**'s configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, 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?

- **No, the command is not successful.**
- **there's no output**

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 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.

```
vboxuser@Workstation:~$ ansible all -m apt -a name=vim-nox --become --ask-become
-pass
BECOME password:
192.168.26.12 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757072683,
    "cache_updated": false,
    "changed": false
}
192.168.26.11 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757072737,
    "cache_updated": false,
    "changed": false
}
```

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

```
vboxuser@Workstation:~$ ansible all -m apt -a name=vim-nox --become --ask-become
-pass
BECOME password:
192.168.26.11 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756465515,
    "cache_updated": false,
    "changed": false
}
192.168.26.12 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756465515,
    "cache_updated": false,
    "changed": false
}
```

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?

```
vboxuser@Server1:~$ which vim
/usr/bin/vim
```

```
vboxuser@Workstation:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/noble-updates,noble-security 2:9.1.0016-1ubuntu7.8 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/noble-updates,noble-security,now 2:9.1.0016-1ubuntu7.8 amd64 [installed
,automatic]
  Vi IMproved - enhanced vi editor - compact version
```

- **Yes, the command is successful.**
- **If vim-nox was not already installed, Ansible will show "changed": true.**

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.

```
vboxuser@Workstation:~$ cd /var/log
vboxuser@Workstation:/var/log$ ls
alternatives.log       dmesg.0          lastlog
apport.log             dmesg.1.gz       openvpn
apt                    dmesg.2.gz       private
auth.log               dmesg.3.gz       README
boot.log               dmesg.4.gz       speech-dispatcher
boot.log.1             dpkg.log         sssd
bootstrap.log          faillog          syslog
btmp                   fontconfig.log   sysstat
cloud-init.log         gdm3             ubuntu-advantage-apt-hook.log
cloud-init-output.log  gpu-manager.log  ufw.log
cups                   hp               unattended-upgrades
cups-browsed           installer        vboxpostinstall.log
dist-upgrade           journal          wtmp
dmesg                  kern.log
vboxuser@Workstation:/var/log$ cd apt
vboxuser@Workstation:/var/log/apt$ cat history.log
```

```
Start-Date: 2025-08-05  16:48:47
Commandline: apt-get -y --fix-policy install
Install: libgpg-error-l10n:amd64 (1.47-3build2, automatic), e2fsprogs-l10n:amd64
 (1.47.0-2.4~exp1ubuntu4, automatic), libgpm2:amd64 (1.20.7-11, automatic), psmi
sc:amd64 (23.7-1build1, automatic), uuid-runtime:amd64 (2.39.3-9ubuntu6, automat
ic), bash-completion:amd64 (1:2.11-8, automatic), bsdextrautils:amd64 (2.39.3-9u
buntu6, automatic)
End-Date: 2025-08-05  16:48:49

Start-Date: 2025-08-05  16:49:12
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes --force-yes upg
rade
Upgrade: libgpg-error-l10n:amd64 (1.47-3build2, 1.47-3build2.1), dpkg:amd64 (1.2
2.6ubuntu6, 1.22.6ubuntu6.1), libsmartcols1:amd64 (2.39.3-9ubuntu6, 2.39.3-9ubun
tu6.3), libpam-runtime:amd64 (1.5.3-5ubuntu5, 1.5.3-5ubuntu5.4), udev:amd64 (255
.4-1ubuntu8, 255.4-1ubuntu8.10), krb5-locales:amd64 (1.20.1-6ubuntu2, 1.20.1-6ub
untu2.6), python3.12:amd64 (3.12.3-1, 3.12.3-1ubuntu0.7), libgssapi-krb5-2:amd64
 (1.20.1-6ubuntu2, 1.20.1-6ubuntu2.6), dhcpcd-base:amd64 (1:10.0.6-1ubuntu3, 1:1
0.0.6-1ubuntu3.1), iputils-ping:amd64 (3:20240117-1build1, 3:20240117-1ubuntu0.1
), libaudit-common:amd64 (1:3.1.2-2.1build1, 1:3.1.2-2.1build1.1), apt:amd64 (2.
7.14build2, 2.8.3), xkb-data:amd64 (2.41-2ubuntu1, 2.41-2ubuntu1.1), dbus-user-s
ession:amd64 (1.14.10-4ubuntu4, 1.14.10-4ubuntu4.1), libacl1:amd64 (2.3.2-1build
1, 2.3.2-1build1.1), systemd-timesyncd:amd64 (255.4-1ubuntu8, 255.4-1ubuntu8.10)
```

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?

   - **The command will be successful.**
   - **If snapd was already installed: "changed": false**
   - **If it was not installed: "changed": true**

   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.

```
vboxuser@Workstation:~$ ansible all -m apt -a "name=snapd state=latest" --become
 --ask-become-pass
BECOME password:
192.168.26.11 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757072737,
    "cache_updated": false,
    "changed": false
}
192.168.26.12 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757072683,
    "cache_updated": false,
    "changed": false
}
```

- ***This command ensures that snapd is installed and up to date.***
- ***If snapd was already at the latest version: Output shows "changed": false***
- ***This confirms that the remote system has the latest version of snapd.***

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

```
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git add .
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git commit -m "Completed
 Ansible ad-hoc tasks for apt update, vim, and snapd"
[main 7610301] Completed Ansible ad-hoc tasks for apt update, vim, and snapd
 2 files changed, 11 insertions(+)
 create mode 100644 .install_apache.yml.swp
 create mode 100644 install_apache.yml
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git push
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), 610 bytes | 610.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:AlexzanderReyes/CPE212_REYES_ALEXZANDER-LAPTOP-.git
   158ce5c..7610301  main -> main
```

Task 2: Writing our First Playbook

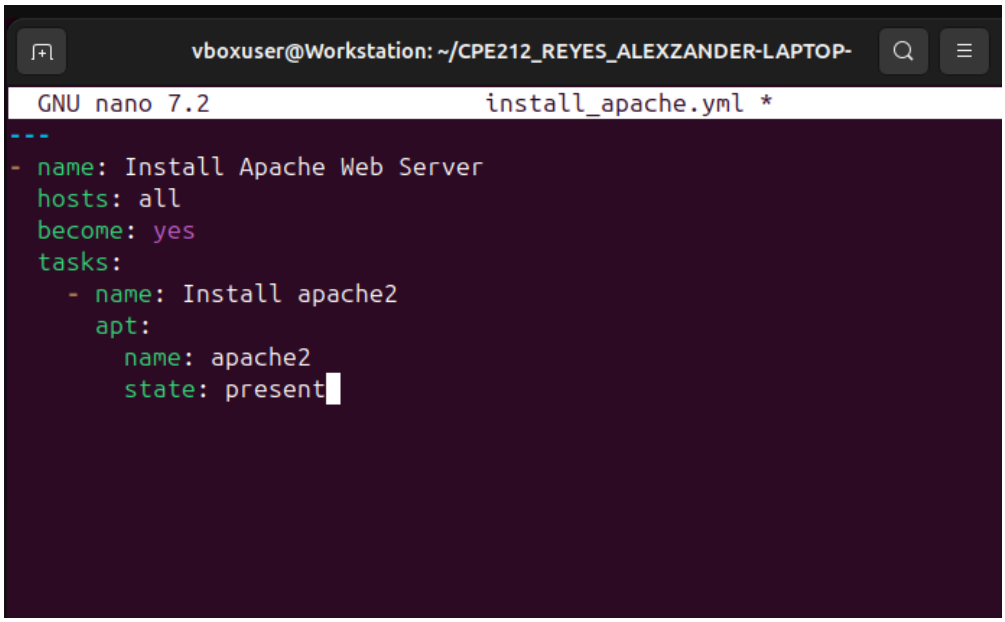
1. With ad hoc commands, we can simplify the administration of remote servers. For example, we can install updates, packages, and applications, etc. However, the real strength of ansible comes from its playbooks. When we write a playbook, we can define the state that we want our servers to be in and 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
```

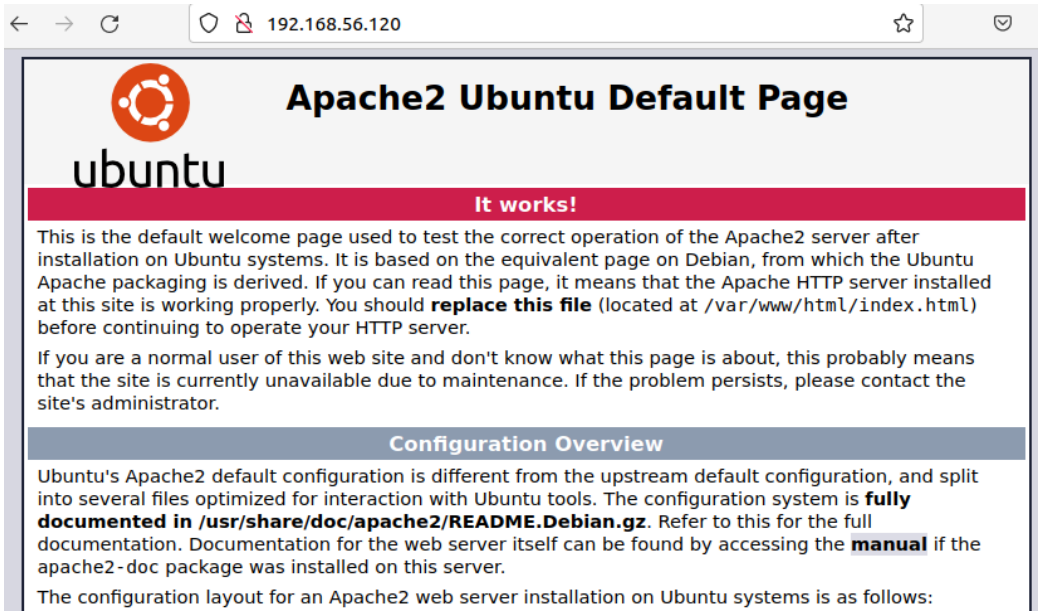
Make sure to save the file. Take note also of the alignments of the texts.



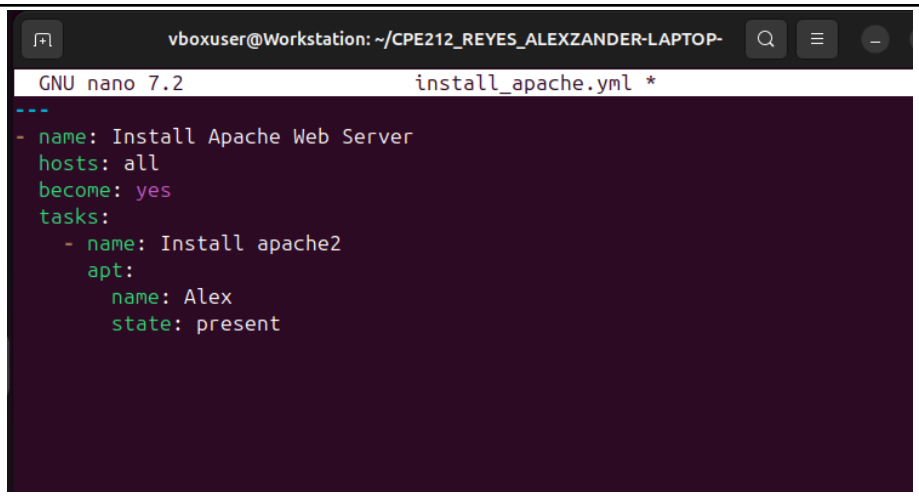
2. Run the yml file using the command: *ansible-playbook --ask-become-pass install_apache.yml.* Describe the result of this command.

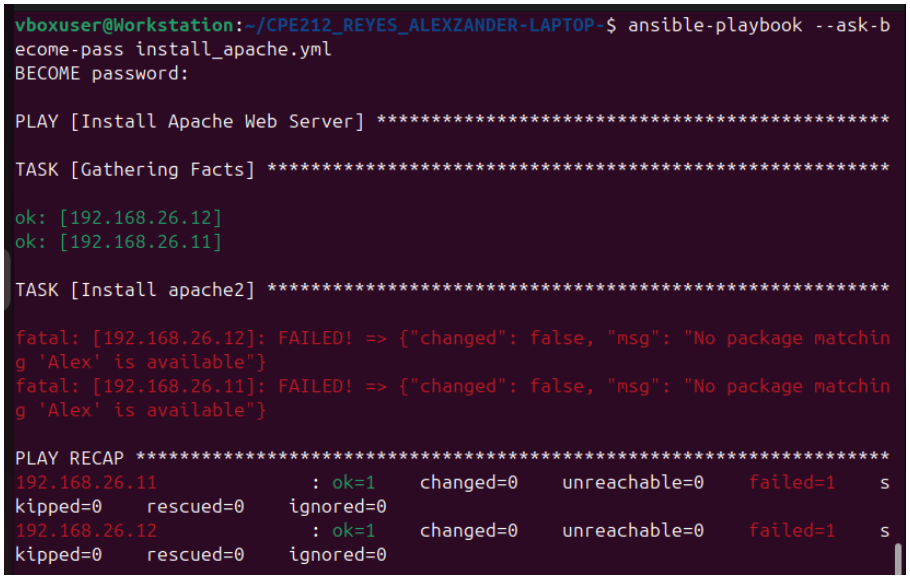
```
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ ansible-playbook --ask-b
ecome-pass install_apache.yml
BECOME password:

PLAY [Install Apache Web Server] ***********************************************

TASK [Gathering Facts] *********************************************************

ok: [192.168.26.11]
ok: [192.168.26.12]

TASK [Install apache2] *********************************************************
ok: [192.168.26.12]
ok: [192.168.26.11]

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

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?



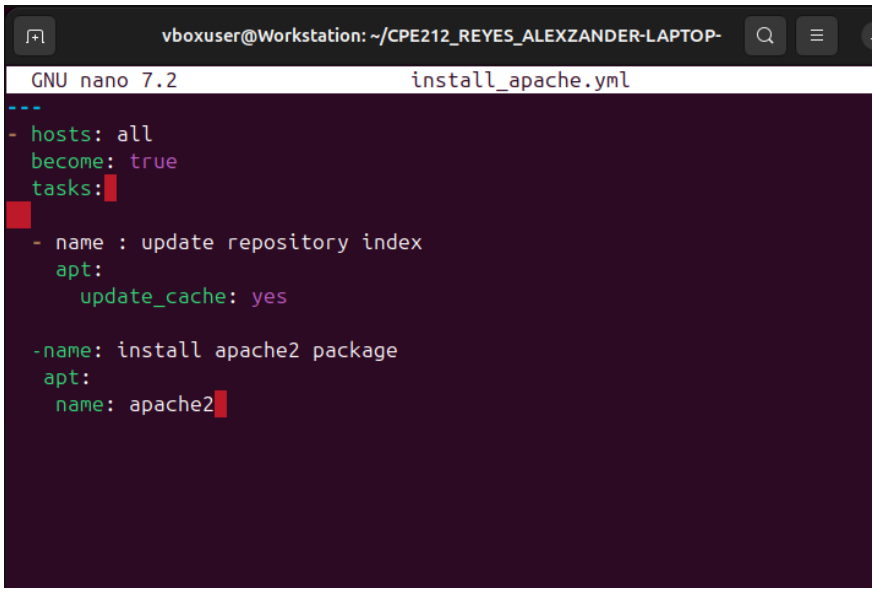


- **This shows that the playbook fails gracefully when packages don't exist.**

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.



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

- yes, add update repository update

```
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP$ nano install_apache.yml
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP$ ansible-playbook --ask-become-p
ass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.26.12]
ok: [192.168.26.11]

TASK [update repository index] *************************************************
changed: [192.168.26.11]
changed: [192.168.26.12]

TASK [install apache2 package] *************************************************
ok: [192.168.26.11]
ok: [192.168.26.12]

PLAY RECAP *********************************************************************
192.168.26.11              : ok=3    changed=1    unreachable=0    failed=0    skipped=
0    rescued=0    ignored=0
192.168.26.12              : ok=3    changed=1    unreachable=0    failed=0    skipped=
```

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.

```
vboxuser@Workstation: ~/CPE212_REYES_ALEXZANDER-LAPTOP-
 GNU nano 7.2                    install_apache.yml
---
- 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
```

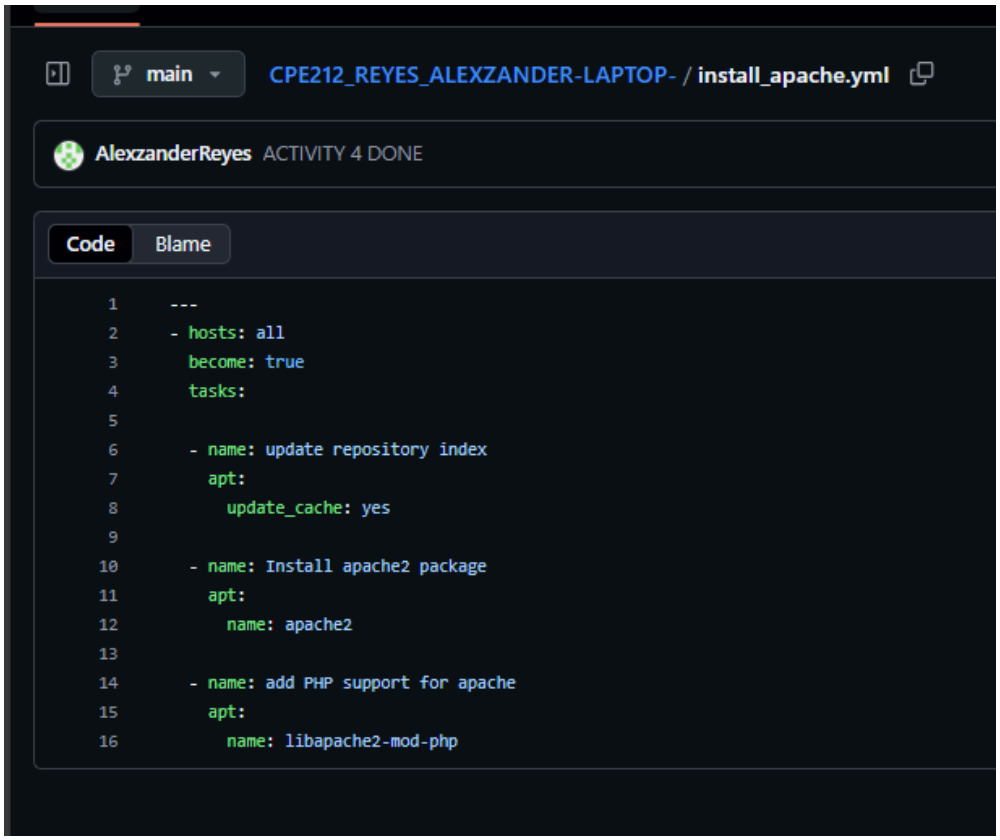
8. Run the playbook and describe the output. Did the new command change anything on the remote servers?
   - yes, it's add PHP support for apache2

```
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ sudo nano install_apache
.yml
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ ansible-playbook --ask-b
ecome-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.26.11]
ok: [192.168.26.12]

TASK [update repository index] *************************************************
changed: [192.168.26.11]
changed: [192.168.26.12]

TASK [Install apache2 package] *************************************************
ok: [192.168.26.12]
ok: [192.168.26.11]
```

```
TASK [add PHP support for apache] ******************************************
changed: [192.168.26.11]
changed: [192.168.26.12]

PLAY RECAP *********************************************************************
192.168.26.11              : ok=4    changed=2    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.26.12              : ok=4    changed=2    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

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

```
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git add install_apache.y
ml
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git commit -m "ACTIVITY
4 DONE"
[main 0122b2c] ACTIVITY 4 DONE
 1 file changed, 6 insertions(+), 1 deletion(-)
vboxuser@Workstation:~/CPE212_REYES_ALEXZANDER-LAPTOP-$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 375 bytes | 375.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:AlexzanderReyes/CPE212_REYES_ALEXZANDER-LAPTOP-.git
   7610301..0122b2c  main -> main
```



Reflections:

Answer the following:

1. What is the importance of using a playbook?
    - Using a playbook is important because it provides a clear, repeatable set of instructions to automate tasks, whether in IT, education, business, or other fields. It ensures consistency by documenting the exact steps needed to achieve a goal, reducing the chance of errors and saving time. Playbooks also make it easier for teams to collaborate, as they serve as a shared reference that anyone can follow. Whether setting up software, managing classroom procedures, or handling customer service scenarios, a playbook helps streamline processes and maintain quality across repeated activities.
2. Summarize what we have done on this activity.

    - In this activity, we used Ansible to manage remote servers by running elevated ad hoc commands and creating playbooks. We updated package information, installed Vim, and managed the snapd package while learning the importance of privilege escalation with --become. Then, we wrote a playbook to automate Apache installation, verified its deployment, tested error handling with invalid package names, and enhanced the playbook by adding cache updates and PHP support. This activity highlighted how Ansible playbooks simplify and automate system administration, ensuring consistent configurations across servers while emphasizing the need for proper privilege management.