

Name: LEE IVAN TITONG	Date Performed: 8-28-2025
Course/Section: CPE 212-CPE31S2	Date Submitted: 8-28-2025
Instructor: Engr. Robin Valenzuela	Semester and SY: 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

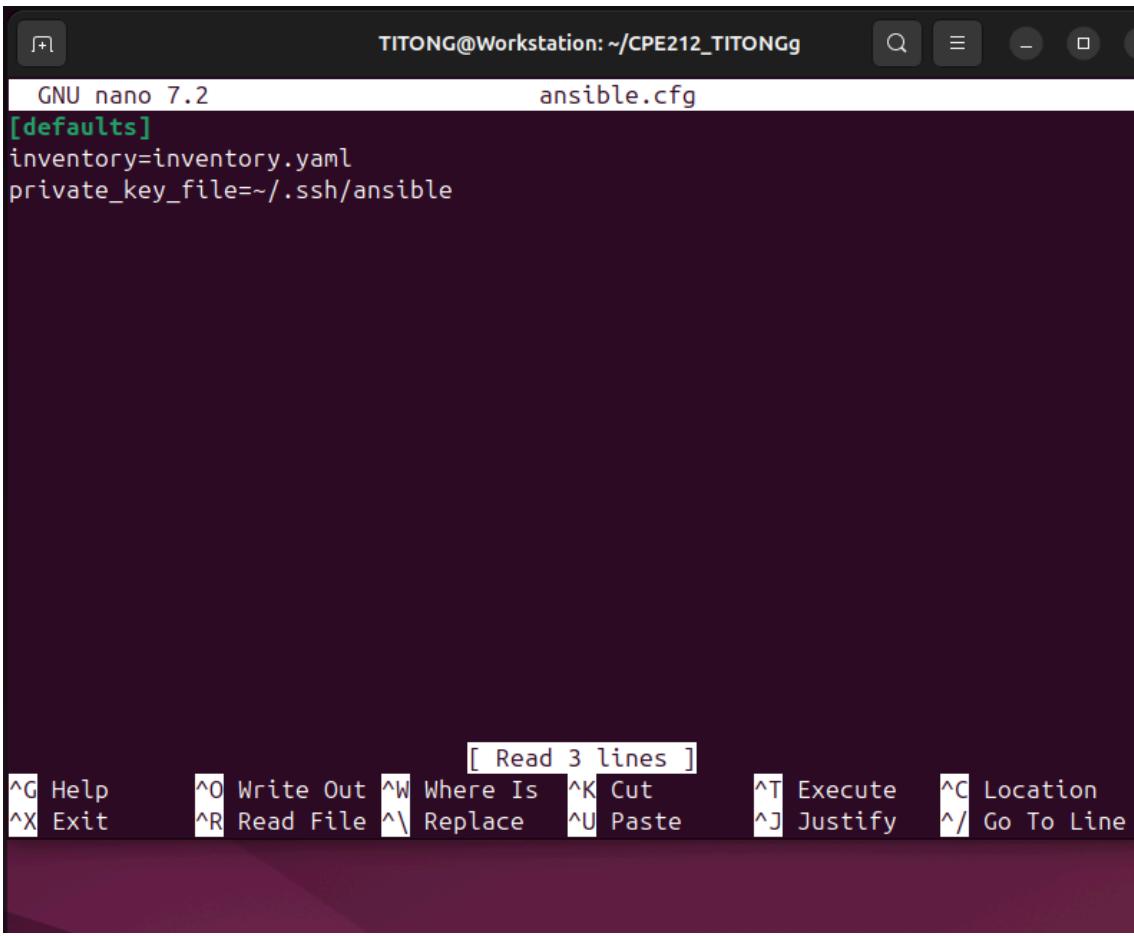
So far, we have not performed ansible commands that makes changes to the remote servers. We manage to gather facts and connect to the remote machines, but we still did not make changes on those machines. In this activity, we will learn to use commands that would install, update, and upgrade packages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



The screenshot shows a terminal window titled "TITONG@Workstation: ~/CPE212_TITONGg". The window contains the "GNU nano 7.2" text editor with the file "ansible.cfg" open. The content of the file is as follows:

```
[defaults]
inventory=inventory.yaml
private_key_file=~/ssh/ansible
```

At the bottom of the terminal window, there is a menu bar with various keyboard shortcuts for navigating and editing the file.

The screenshot shows a terminal window titled "TITONG@Workstation: ~/CPE212_TITONGg". The file being edited is "inventory.yaml". The content of the file is:

```
[Server]
192.168.56.110
192.168.56.111
```

The bottom of the terminal shows the nano editor's command bar with various keyboard shortcuts.

ansible all -m apt -a update_cache=true

```
TITONG@Workstation:~$ ansible all -m apt -a "update_cache=true"
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'
TITONG@Workstation:~$ ansible all -m apt -a update_cache=true
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the
implicit localhost does not match 'all'
TITONG@Workstation:~$
```

```
TITONG@Workstation:~$ ansible all -m apt -a "update_cache=true"
^Z
[5]+  Stopped                  ansible all -m apt -a "update_cache=true"
TITONG@Workstation:~$
```

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

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.

```
TITONG@Workstation:~$ ansible all -m apt -a update_cache=true --become --ask-become-pass
BECOME password:
192.168.56.102 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": true,
    "changed": true
}
192.168.56.104 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": true,
    "changed": true
}
TITONG@Workstation: $
```

```
TITONG@Workstation:~/CPE212_TITONGg$ ansible all -m ping --become --ask-become-pass
BECOME password:
192.168.56.110 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
192.168.56.111 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
TITONG@Workstation:~/CPE212_TITONGg$
```

The `update_cache=true` is the same thing as running `a`. 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 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.

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?

```
TITONG@Workstation:~$ ansible all -a "which vim" --become --ask-become-pass
BECOME password:
192.168.56.102 | CHANGED | rc=0 >>
/usr/bin/vim
192.168.56.104 | CHANGED | rc=0 >>
/usr/bin/vim
TITONG@Workstation:~$
```

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`.

```
TITONG@Workstation:~$ ansible all -a "cat /var/log/apt/history.log" --become --ask-become-pass
BECOME password:
192.168.56.102 | CHANGED | rc=0 >>

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), psmisc:amd64 (23.7-1build1, automatic), uuid-runtime:amd64 (2.39.3-9ubuntu6, automatic), bash-completion:amd64 (1:2.11-8, automatic), bsdextrautils:amd64 (2.39.3-9ubuntu6, 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 upgrade
Upgrade: libgpg-error-l10n:amd64 (1.47-3build2, 1.47-3build2.1), dpkg:amd64 (1.22.6ubuntu6, 1.22.6ubuntu6.1), libsmartcols1:amd64 (2.39.3-9ubuntu6, 2.39.3-9ubuntu6.3), 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, 1.20.1-6ubuntu2.6), 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, 1:10.0.6-1ubuntu3.1), iutils-ping:amd64 (3:0240117-1build1, 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-session:amd64 (1.14.10-4ubuntu4, 1.14.10-4ubuntu4.1), libacl1:amd64 (2.3.2-1build1, 2.3.2-1build1.1), systemd-timesyncd:amd64 (255.4-1ubuntu8, 255.4-1ubuntu8.10), tzdata:amd64 (2024a-2ubuntu1, 2025b-0ubuntu0.24.04.1), libunistring5:amd64 (1.1-2build1, 1.1-2build1.1), libidn2-0:amd64 (2.3.7-2build1, 2.3.7-2build1.1), libtasn1-6:amd64 (4.19.0-3build1, 4.19.0-3ubuntu0.24.04.1), python3-minimal:amd64 (3.12.3-0ubuntu1, 3.12.3-0ubuntu2), libpam-systemd:amd64 (255.4-1ubuntu8, 255.4-1ubuntu8.10), libselinux1:amd64 (3.5-2ubuntu2, 3.5-2ubuntu2.1), liblzma5:amd64 (5.6.1+really5.4.5-1, 5.6.1+really5.4.5-1ubuntu0.2), libhogweed6t64:amd64 (3.9.1-2.2build1, 3.9.1-2.2build1.1), e2fsprogs-l10n:amd64 (1.47.0-2.4~exp1ubuntu4, 1.47.0-2.4~exp1ubuntu4.1), libcap2-bin:amd64 (1:2.66-5ubuntu2, 1:2.66-5ubuntu2.2), gir1.2-glib-2.0:amd64 (2.80.0-6ubuntu1, 2.80.0-6ubuntu3.4), libext2fs2t64:amd64 (1.47.0-2.4~exp1ubuntu4, 1.47.0-2.4~exp1ubuntu4.1), xxd:amd64 (2:9.1.0016-1ubuntu7, 2:9.1.0016-1ubuntu7.8), libexpat1:amd64 (2.6.1-2build1, 2
```

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`

```
TITONG@Workstation:~$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": false,
    "changed": false
}
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": false,
    "changed": false
}
TITONG@Workstation:~$
```

Can you describe the result of this command? Is it a success? Did it change anything in the remote servers?

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

```
TITONG@Workstation:~$ ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": false,
    "changed": false
}
192.168.56.104 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756330030,
    "cache_updated": false,
    "changed": false
}
TITONG@Workstation:~$
```

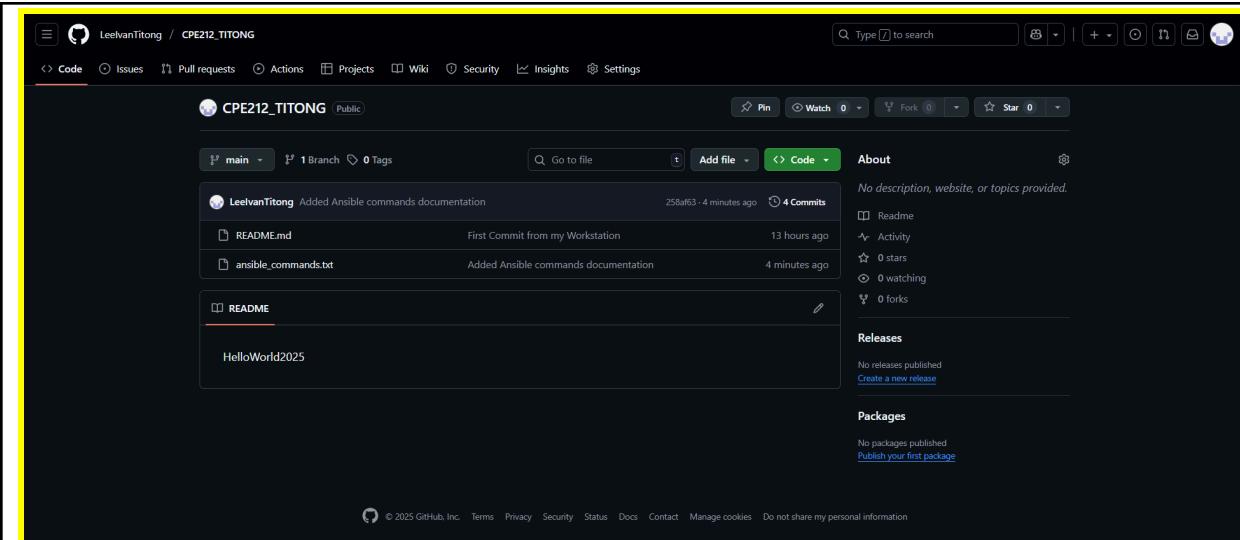
Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

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

```
TITONG@Workstation:~$ nano ~/CPE212_TITONG/ansible_commands.txt
TITONG@Workstation:~$ CD ~/CPE212_TITONG
CD: command not found
TITONG@Workstation:~$ cd ~/CPE212_TITONG
TITONG@Workstation:~/CPE212_TITONG$ ls
ansible_commands.txt README.md
TITONG@Workstation:~/CPE212_TITONG$ git add ansible_commands.txt
TITONG@Workstation:~/CPE212_TITONG$ git commit -m "Added Ansible commands documentation"
[main 258af63] Added Ansible commands documentation
 1 file changed, 5 insertions(+)
 create mode 100644 ansible_commands.txt
TITONG@Workstation:~/CPE212_TITONG$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 413 bytes | 413.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:LeeIvanTitong/CPE212_TITONG.git
 5806178..258af63 main -> main
TITONG@Workstation:~/CPE212_TITONG$
```

```
GNU nano 7.2                               ansible_commands.txt

Commands executed:
- ansible all -m apt -a "update_cache=true" --become --ask-become-pass
- ansible all -m apt -a "name=vim-nox" --become --ask-become-pass
- ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
```



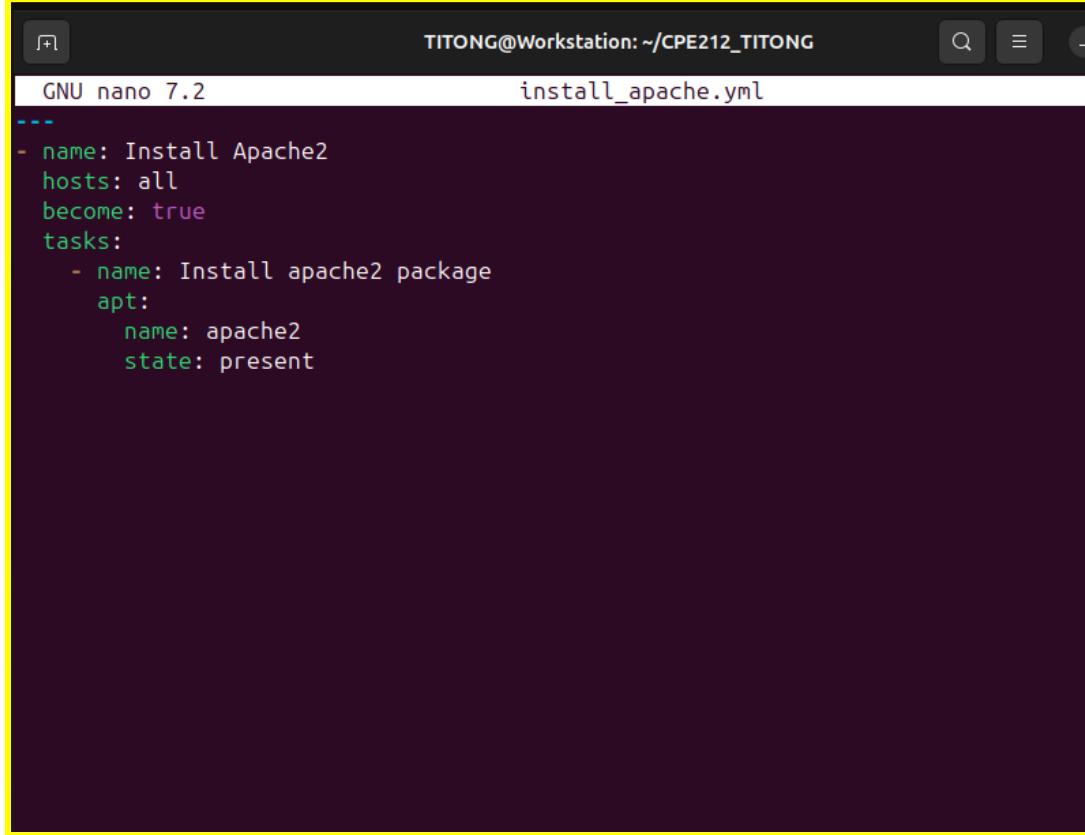
Task 2: Writing our First Playbook

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.

```
TITONG@Workstation:~$ cd ~/CPE212_TITONG
TITONG@Workstation:~/CPE212_TITONG$ nano install_apache.yml
TITONG@Workstation:~/CPE212_TITONG$
```

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
```

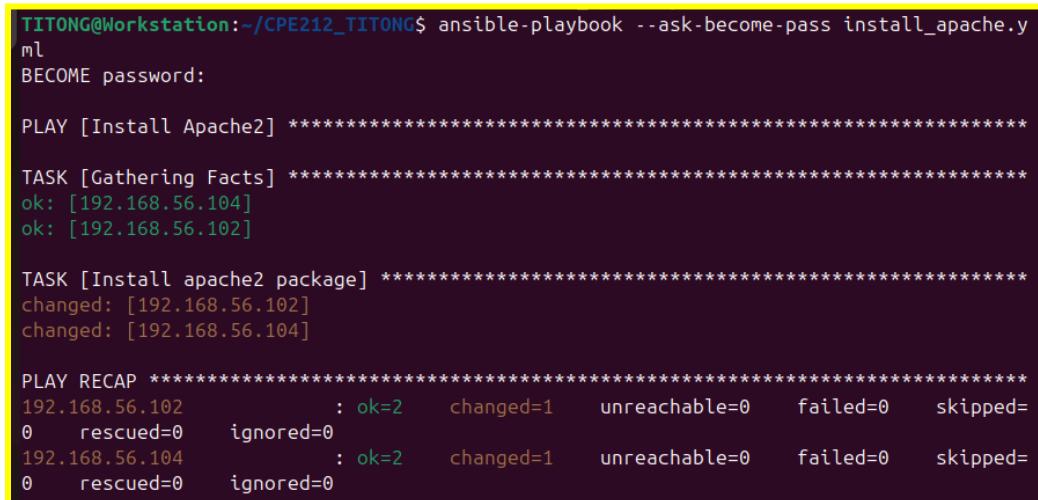


The screenshot shows a terminal window titled "TITONG@Workstation: ~/CPE212_TITONG". Inside the terminal, the file "install_apache.yml" is being edited with the nano text editor. The code in the file is:

```
GNU nano 7.2           install_apache.yml
---
- name: Install Apache2
  hosts: all
  become: true
  tasks:
    - name: Install apache2 package
      apt:
        name: apache2
        state: present
```

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.



The screenshot shows a terminal window with the following output from running the playbook:

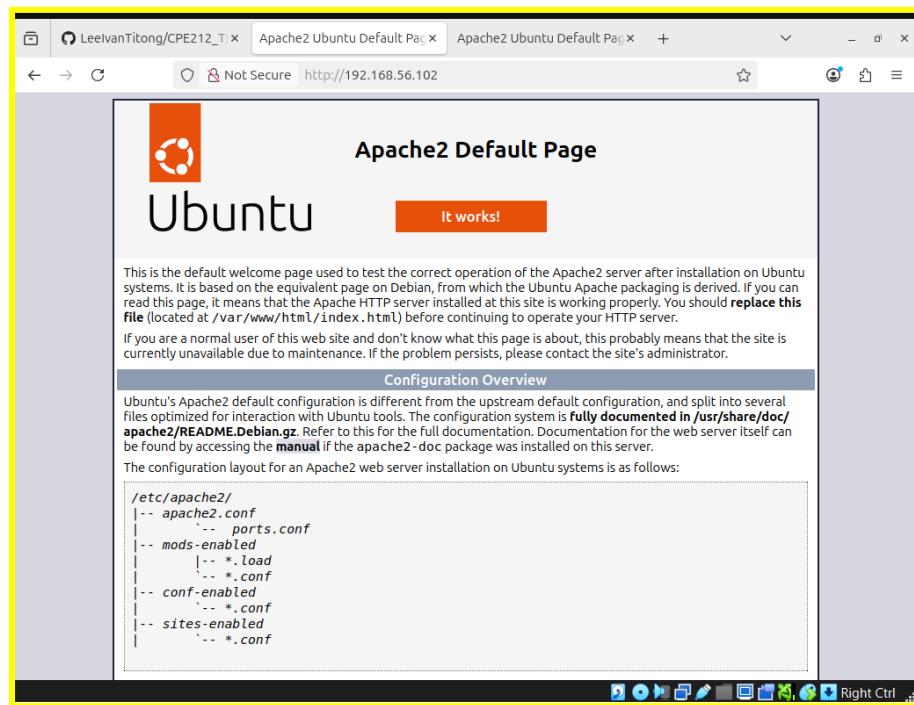
```
TITONG@Workstation:~/CPE212_TITONG$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

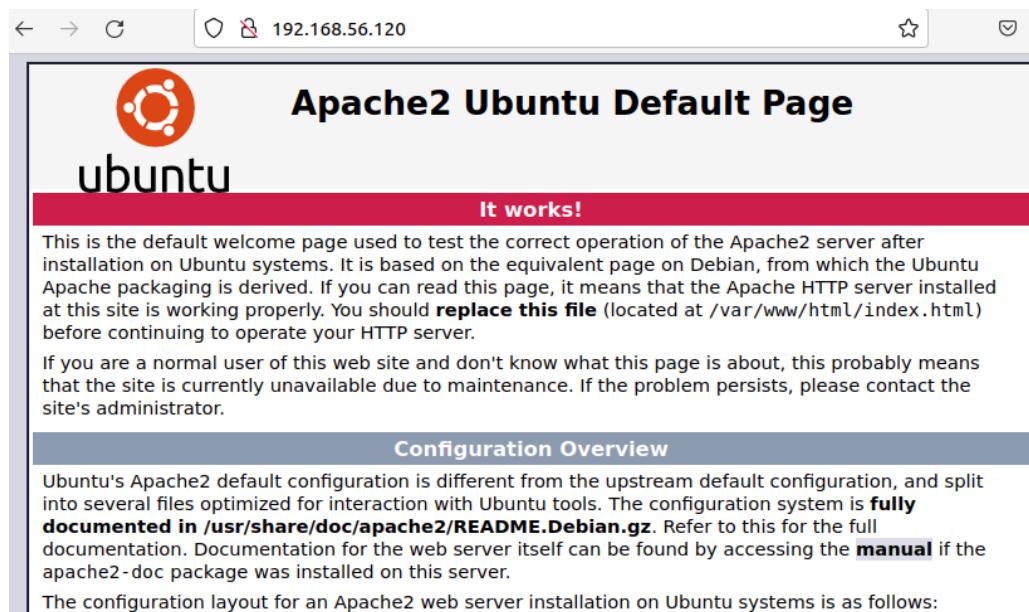
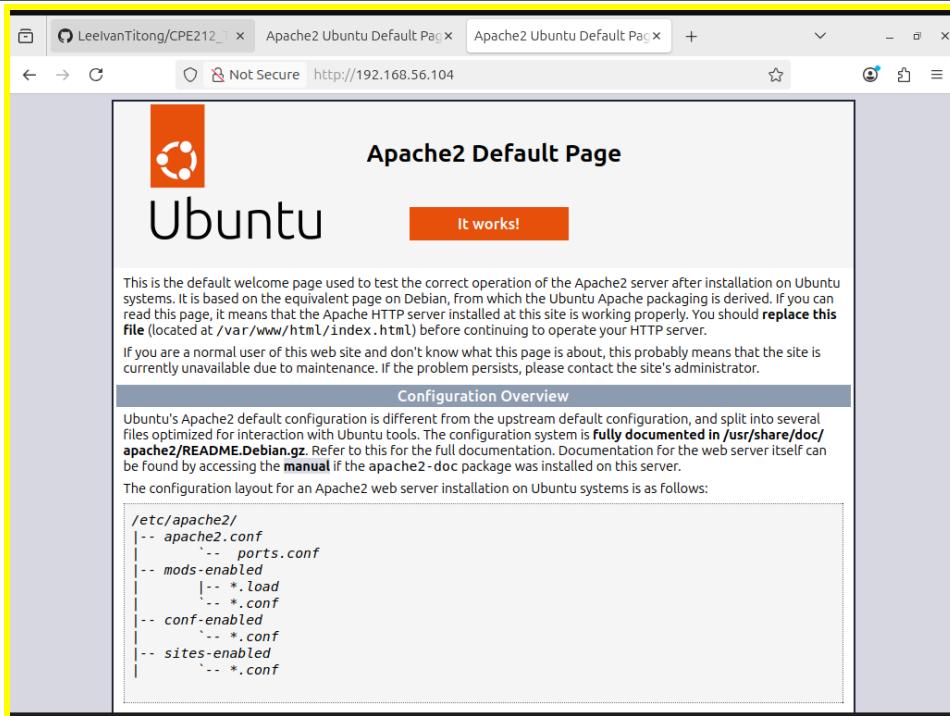
PLAY [Install Apache2] ****
TASK [Gathering Facts] ****
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [Install apache2 package] ****
changed: [192.168.56.102]
changed: [192.168.56.104]

PLAY RECAP ****
192.168.56.102      : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.104      : ok=2    changed=1    unreachable=0    failed=0    skipped=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?

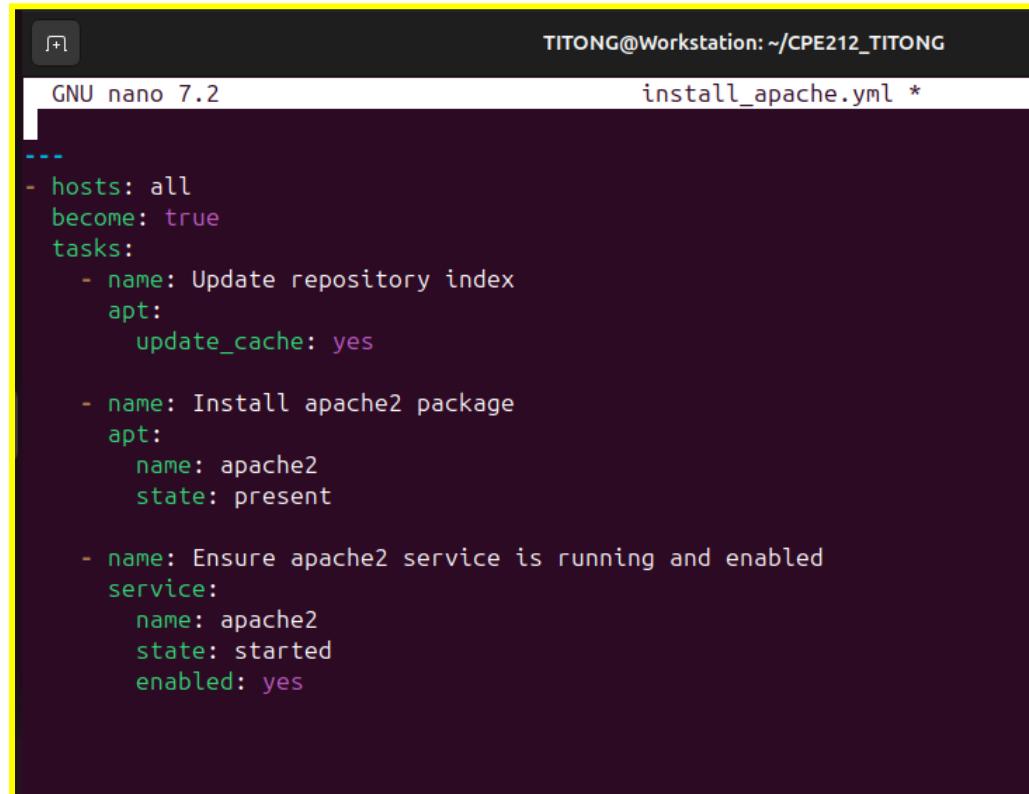
```
TITONG@Workstation:~/CPE212_TITONG$ nano install_apache.yml
```

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.



The screenshot shows a terminal window titled "TITONG@Workstation: ~/CPE212_TITONG". The window displays the "install_apache.yml" file in the "GNU nano 7.2" editor. The file contains the YAML code shown in the previous code block, with syntax highlighting for keywords like "hosts", "become", "tasks", "name", "apt", and "state". The file is currently being edited, as indicated by the asterisk (*) at the end of the file name in the title bar.

```
---
- hosts: all
  become: true
  tasks:
    - name: Update repository index
      apt:
        update_cache: yes
    - name: Install apache2 package
      apt:
        name: apache2
        state: present
    - name: Ensure apache2 service is running and enabled
      service:
        name: apache2
        state: started
        enabled: yes
```

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

```
TITONG@Workstation:~/CPE212_TITONG$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] ****
TASK [Gathering Facts] ****
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [Update repository index] ****
changed: [192.168.56.104]
changed: [192.168.56.102]

TASK [Install apache2 package] ****
ok: [192.168.56.102]
ok: [192.168.56.104]

TASK [Ensure apache2 service is running and enabled] ****
ok: [192.168.56.102]
ok: [192.168.56.104]

PLAY RECAP ****
192.168.56.102      : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
                      ignored=0
192.168.56.104      : ok=4    changed=1    unreachable=0    failed=0    skipped=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.

```
TITONG@Workstation:~/CPE212_TITONG$ nano 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
```

Save the changes to this file and exit.

```
TITONG@Workstation:~/CPE212_TITONG
GNU nano 7.2           install_apache.yml

---
- name: Install Apache2 and PHP support
  hosts: all
  become: true
  tasks:
    - name: Update repository index
      apt:
        update_cache: yes

    - name: Install apache2 package
      apt:
        name: apache2
        state: present

    - name: Ensure apache2 service is running and enabled
      service:
        name: apache2
        state: started
        enabled: yes

    - name: Add PHP support for apache
      apt:
        name: libapache2-mod-php
        state: present

[ Read 25 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut      ^T Execute  ^C Loc
^X Exit      ^R Read File  ^\ Replace   ^U Paste    ^J Justify  ^/ Go

```

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

```
TITONG@Workstation:~/CPE212_TITONG$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [Install Apache2 and PHP support] ****
TASK [Gathering Facts] ****
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [Update repository index] ****
changed: [192.168.56.104]
changed: [192.168.56.102]

TASK [Install apache2 package] ****
ok: [192.168.56.102]
ok: [192.168.56.104]

TASK [Ensure apache2 service is running and enabled] ****
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [Add PHP support for apache] ****
ok: [192.168.56.104]
ok: [192.168.56.102]

PLAY RECAP ****
192.168.56.102      : ok=5    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.104      : ok=5    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

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

```
TITONG@Workstation:~/CPE212_TITONG$ git add install_apache.yml
TITONG@Workstation:~/CPE212_TITONG$ git commit -m "Added Ansible playbook for Apache2 and PHP installation"
[main aa490c5] Added Ansible playbook for Apache2 and PHP installation
 1 file changed, 25 insertions(+)
  create mode 100644 install_apache.yml
TITONG@Workstation:~/CPE212_TITONG$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 576 bytes | 576.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:LeeIvanTitong/CPE212_TITONG.git
  258af63..aa490c5  main -> main
TITONG@Workstation:~/CPE212_TITONG$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
TITONG@Workstation:~/CPE212_TITONG$ git remote -v
origin  git@github.com:LeeIvanTitong/CPE212_TITONG.git (fetch)
origin  git@github.com:LeeIvanTitong/CPE212_TITONG.git (push)
```

Reflections:

Answer the following:

1. What is the importance of using a playbook?
 - Saves time: You can set up many servers at once instead of one by one.

- No mistakes: The same setup happens exactly the same way every time.
 - Easy to remember: The playbook is like a recipe that you can save and use again.
 - Easy to share: Others can use your playbook to set up the same thing.
2. Summarize what we have done on this activity.
- We used special commands to update and install software (like vim) on other computers from our own computer.
 - We learned to use sudo with Ansible by typing --become --ask-become-pass and our password.
 - We created a recipe file (playbook) named `install_apache.yml` that tells Ansible how to install Apache and PHP automatically.
 - We ran the recipe, and it set everything up for us on the other computers.
 - We saved our work on GitHub to keep it safe and shareable.