Name: BONIFACIO, REDJ GUILLIAN F.	**Date Performed:** August 28, 2025
Course/Section: CPE31S4	**Date Submitted:**August 28, 2025
Instructor: Engr. VALENZUELA, ROBIN	**Semester and SY:** 2nd Semester 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.

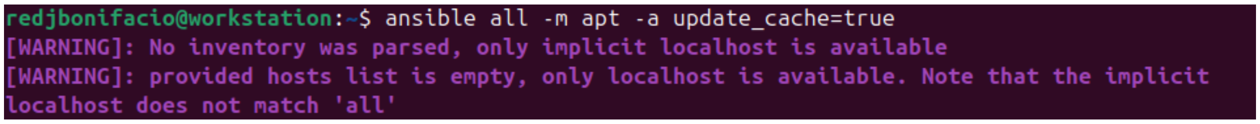
Playbooks record and execute **Ansible**'s configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals,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

   ansible all -m apt -a update_cache=true

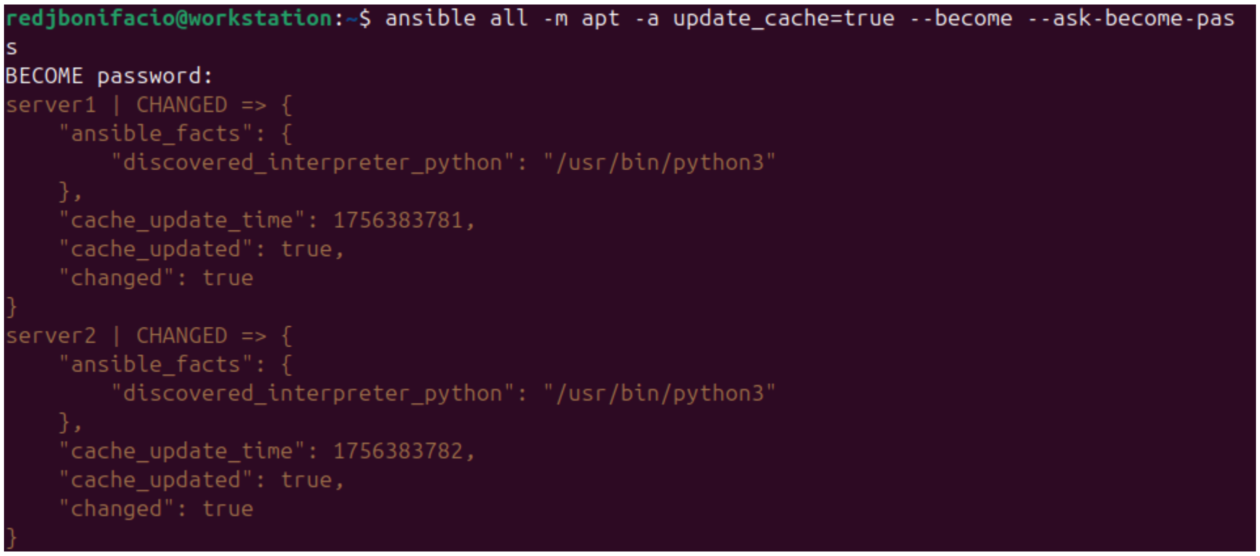
   What is the result of the command? Is it successful?



```
redjbonifacio@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'
```

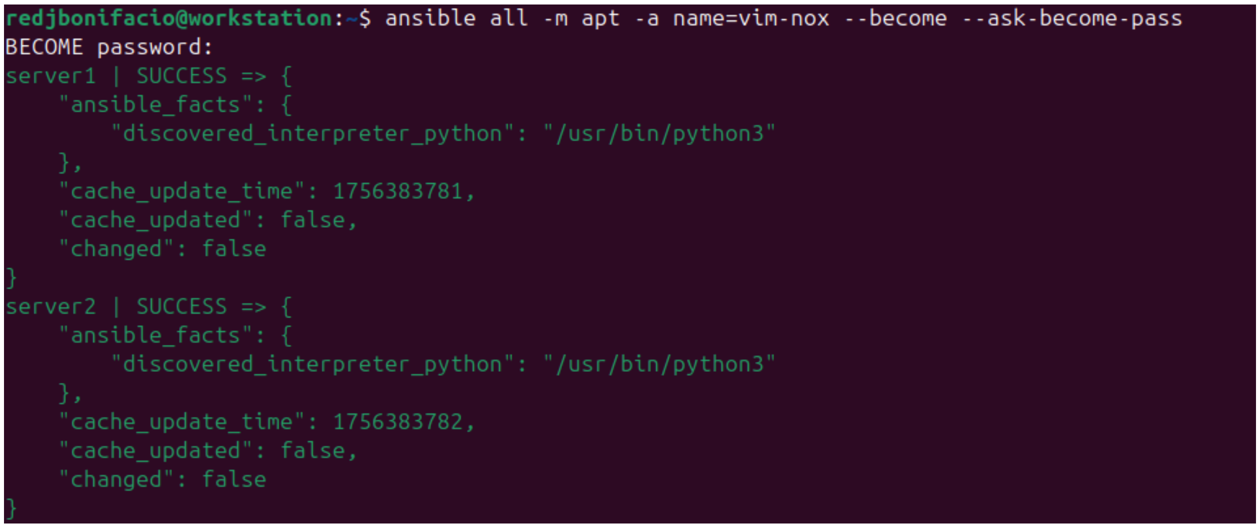
Try editing the command and add something that would elevate the privilege. Issue the command *ansible all -m apt -a update_cache=true --become --ask-become-pass.* Enter the sudo password when prompted. You will notice now that the output of this command is a success. The *update_cache=true* is the same 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.



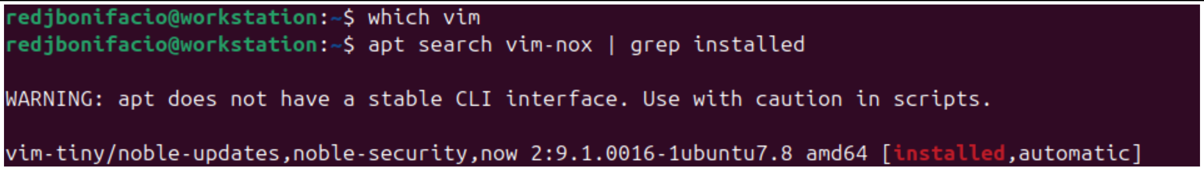
```
redjbonifacio@workstation:~$ ansible all -m apt -a update_cache=true --become --ask-become-pas
s
BECOME password:
server1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383781,
    "cache_updated": true,
    "changed": true
}
server2 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383782,
    "cache_updated": true,
    "changed": true
}
```

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.



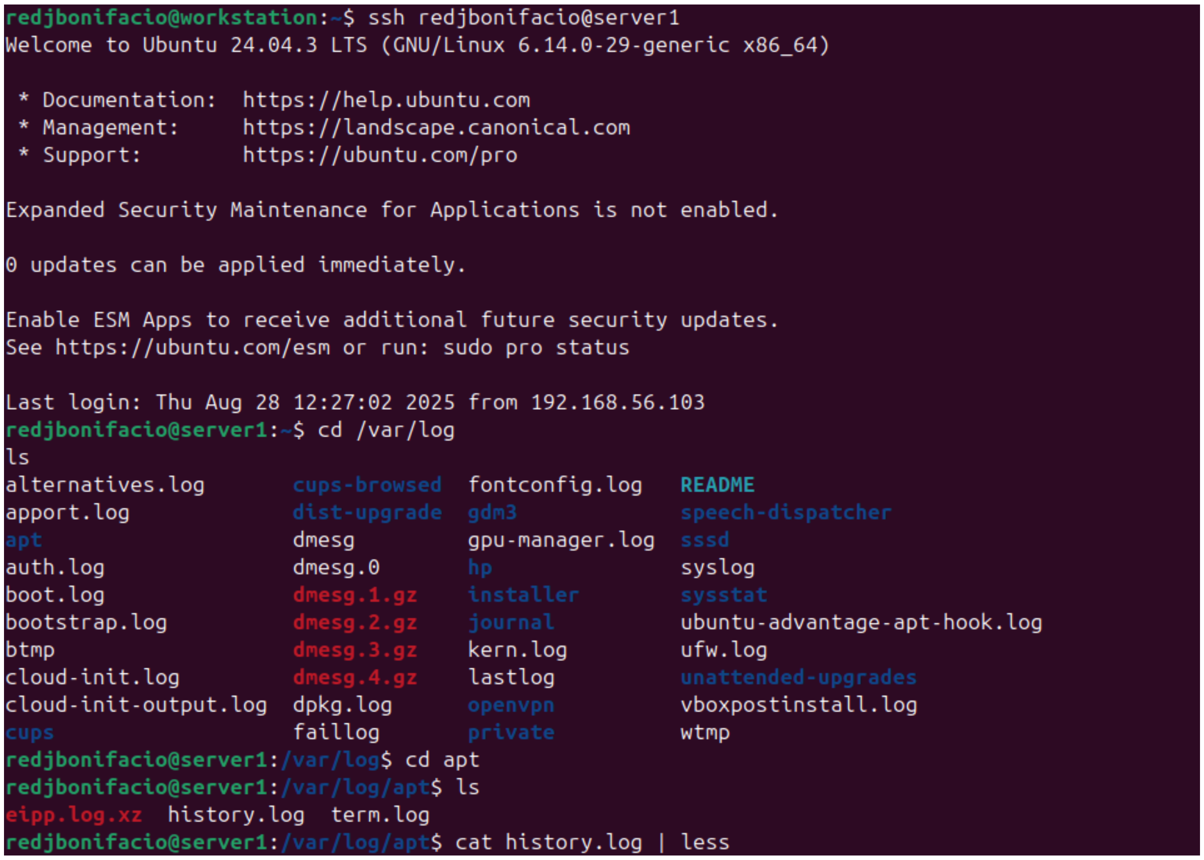
```
redjbonifacio@workstation:~$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
server1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383781,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383782,
    "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?



```
redjbonifacio@workstation:~$ which vim
redjbonifacio@workstation:~$ apt search vim-nox | grep installed

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

vim-tiny/noble-updates,noble-security,now 2:9.1.0016-1ubuntu7.8 amd64 [installed,automatic]
```

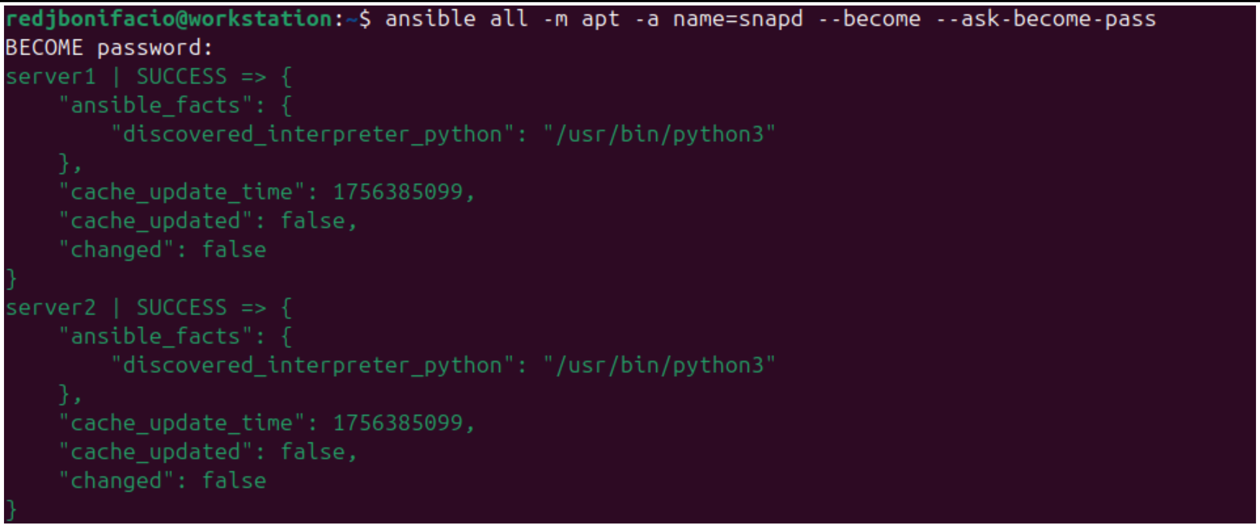
2.2 Check the logs in the servers using the following commands: *cd /var/log*. After this, issue the command *ls,* go to the folder *apt* and open history.log. Describe what you see in the history.log.



```
redjbonifacio@workstation:~$ ssh redjbonifacio@server1
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-29-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Thu Aug 28 12:27:02 2025 from 192.168.56.103
redjbonifacio@server1:~$ cd /var/log
ls
alternatives.log       cups-browsed  fontconfig.log   README
apport.log             dist-upgrade  gdm3             speech-dispatcher
apt                    dmesg         gpu-manager.log  sssd
auth.log               dmesg.0       hp               syslog
boot.log               dmesg.1.gz    installer        sysstat
bootstrap.log          dmesg.2.gz    journal          ubuntu-advantage-apt-hook.log
btmp                   dmesg.3.gz    kern.log         ufw.log
cloud-init.log         dmesg.4.gz    lastlog          unattended-upgrades
cloud-init-output.log  dpkg.log      openvpn          vboxpostinstall.log
cups                   faillog       private          wtmp
redjbonifacio@server1:/var/log$ cd apt
redjbonifacio@server1:/var/log/apt$ ls
eipp.log.xz  history.log  term.log
redjbonifacio@server1:/var/log/apt$ cat history.log | less
```

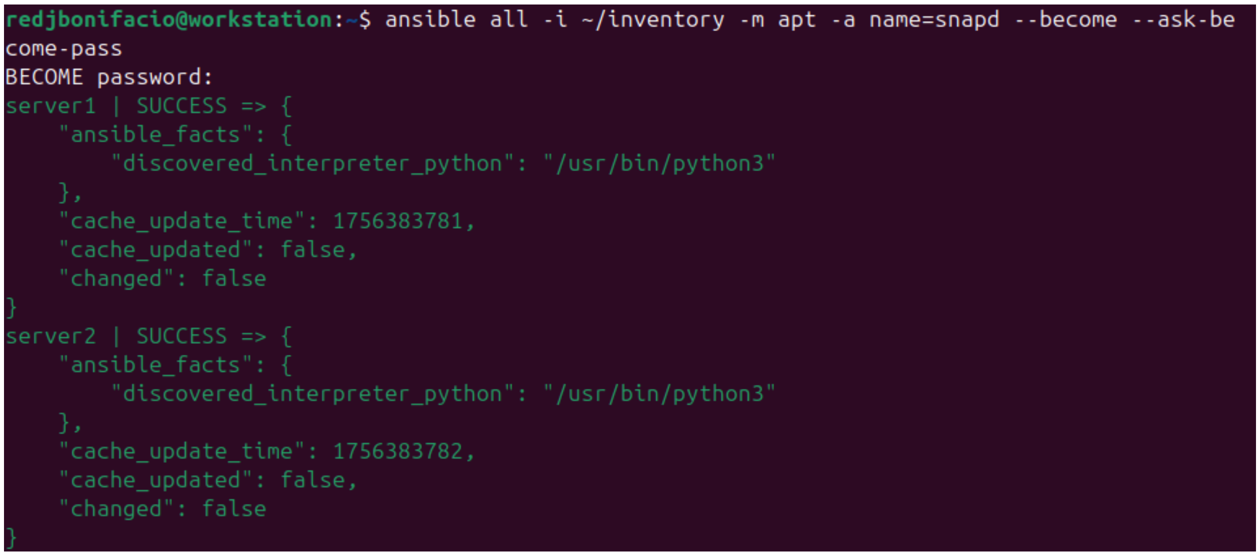
3. This time, we will install a package called snapd. Snap is pre-installed in Ubuntu system. However, our goal is to create a command that checks for the latest installation package.

   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?



```
redjbonifacio@workstation:~$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
server1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756385099,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756385099,
    "cache_updated": false,
    "changed": false
}
```

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.

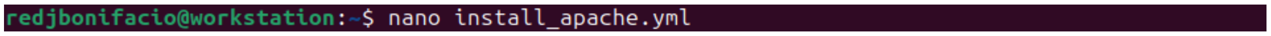


```
redjbonifacio@workstation:~$ ansible all -i ~/inventory -m apt -a name=snapd --become --ask-be
come-pass
BECOME password:
server1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383781,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756383782,
    "cache_updated": false,
    "changed": false
}
```

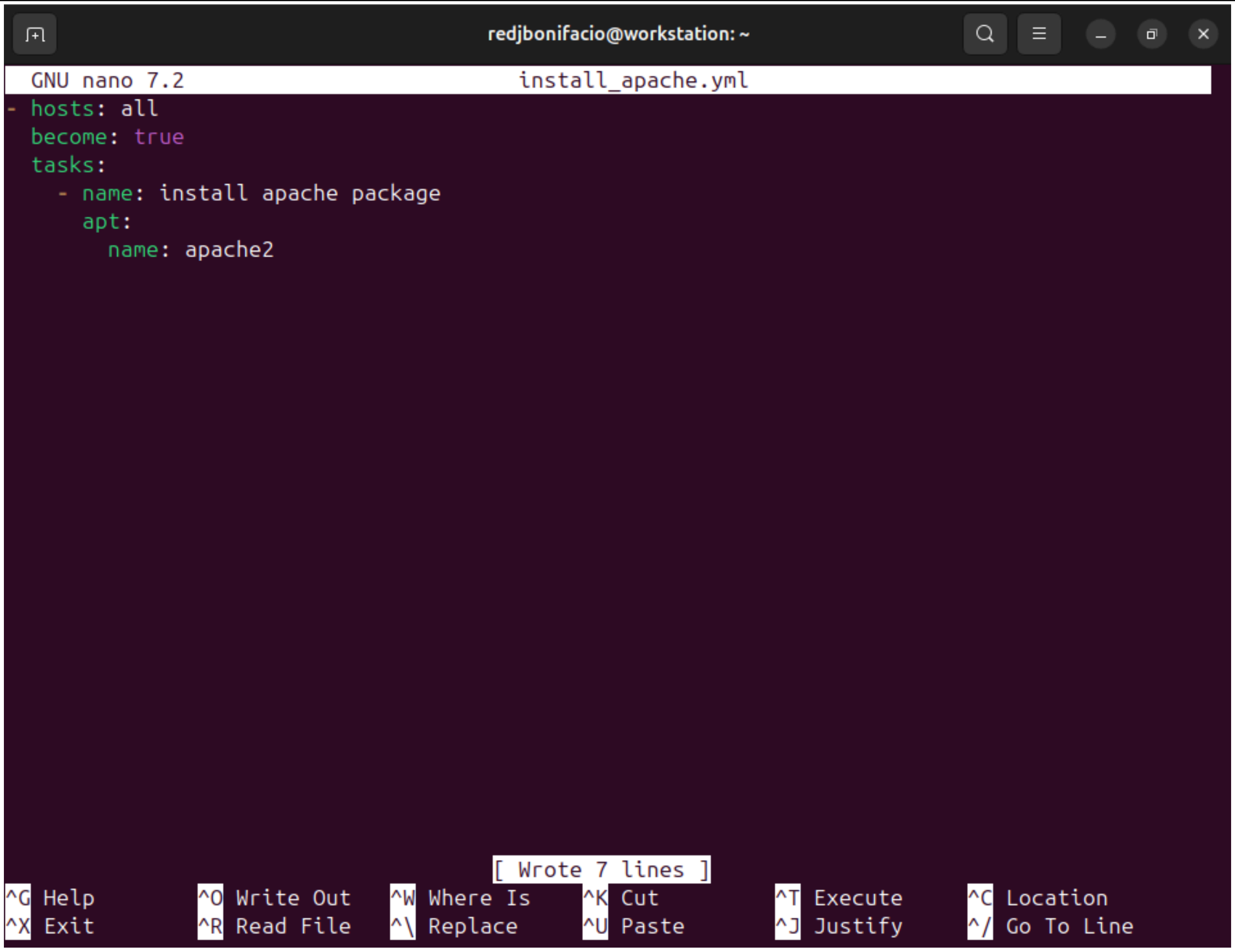
4. At this point, make sure to commit all changes to GitHub.

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.

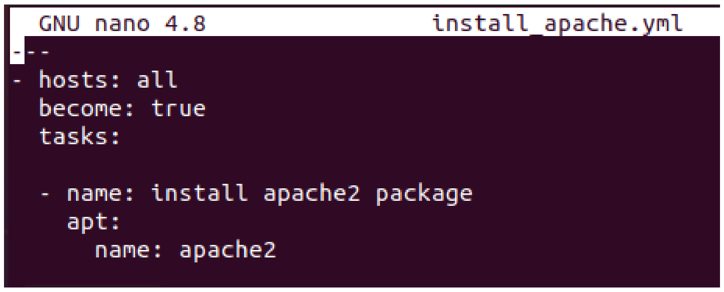


```
redjbonifacio@workstation:~$ nano install_apache.yml
```



```
redjbonifacio@workstation: ~
GNU nano 7.2                        install_apache.yml
- hosts: all
  become: true
  tasks:
    - name: install apache package
      apt:
        name: apache2

                          [ Wrote 7 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

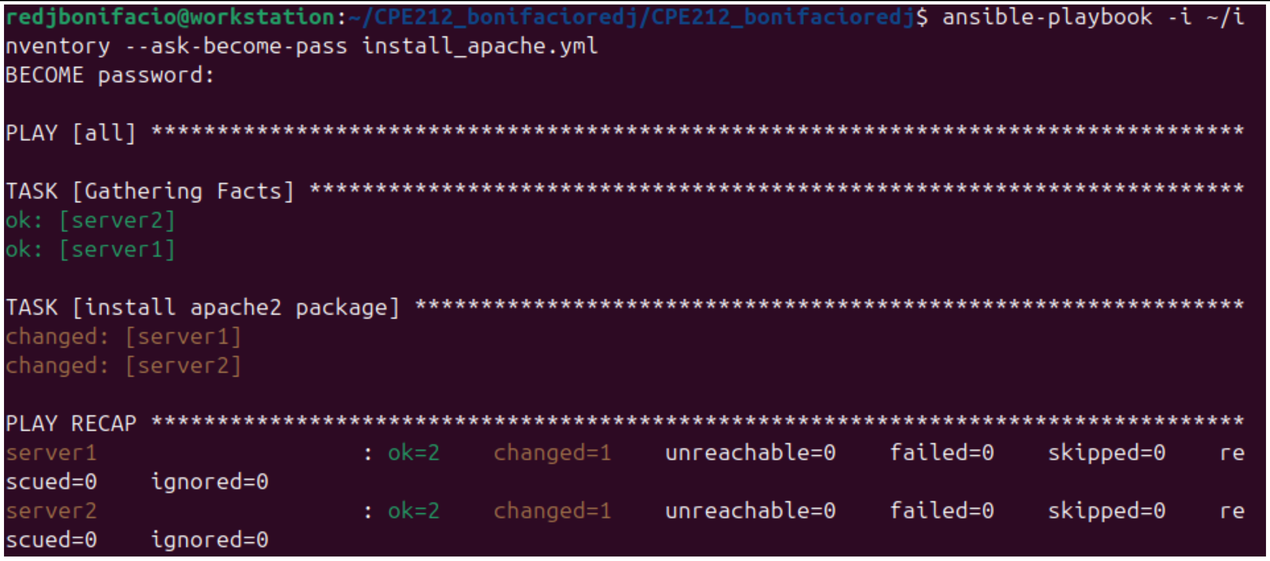
When the editor appears, type the following:



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

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

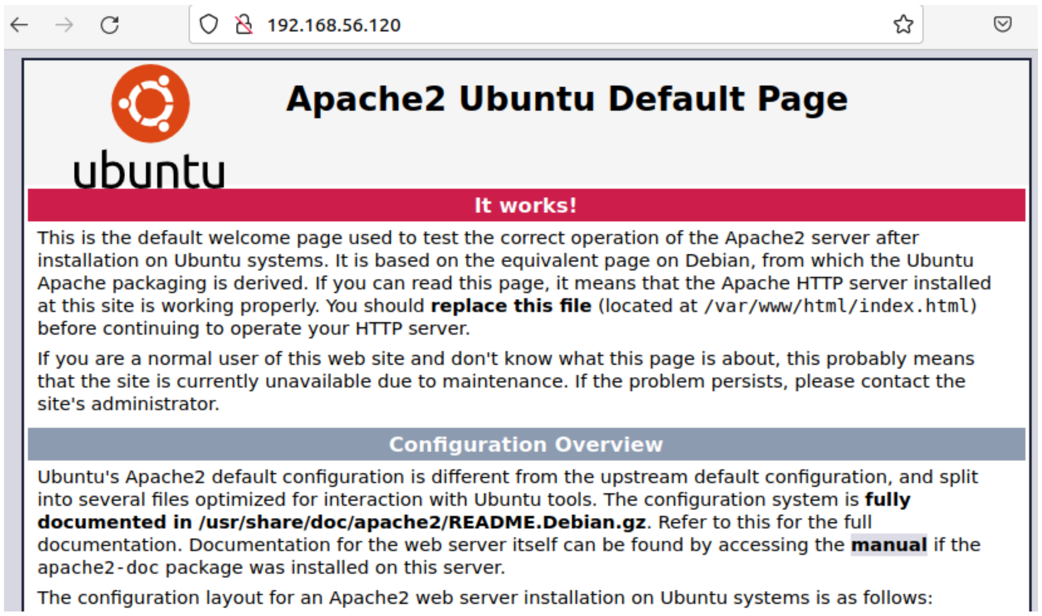
Make sure to save the file. Take note also of the alignments of the texts.

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



```
redjbonifacio@workstation:~/CPE212_bonifacioredj/CPE212_bonifacioredj$ ansible-playbook -i ~/i
nventory --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] ***************************************************************
ok: [server2]
ok: [server1]

TASK [install apache2 package] *******************************************************
changed: [server1]
changed: [server2]

PLAY RECAP ***************************************************************************
server1                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
server2                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    re
scued=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.



192.168.56.120

Apache2 Ubuntu Default Page

It works!

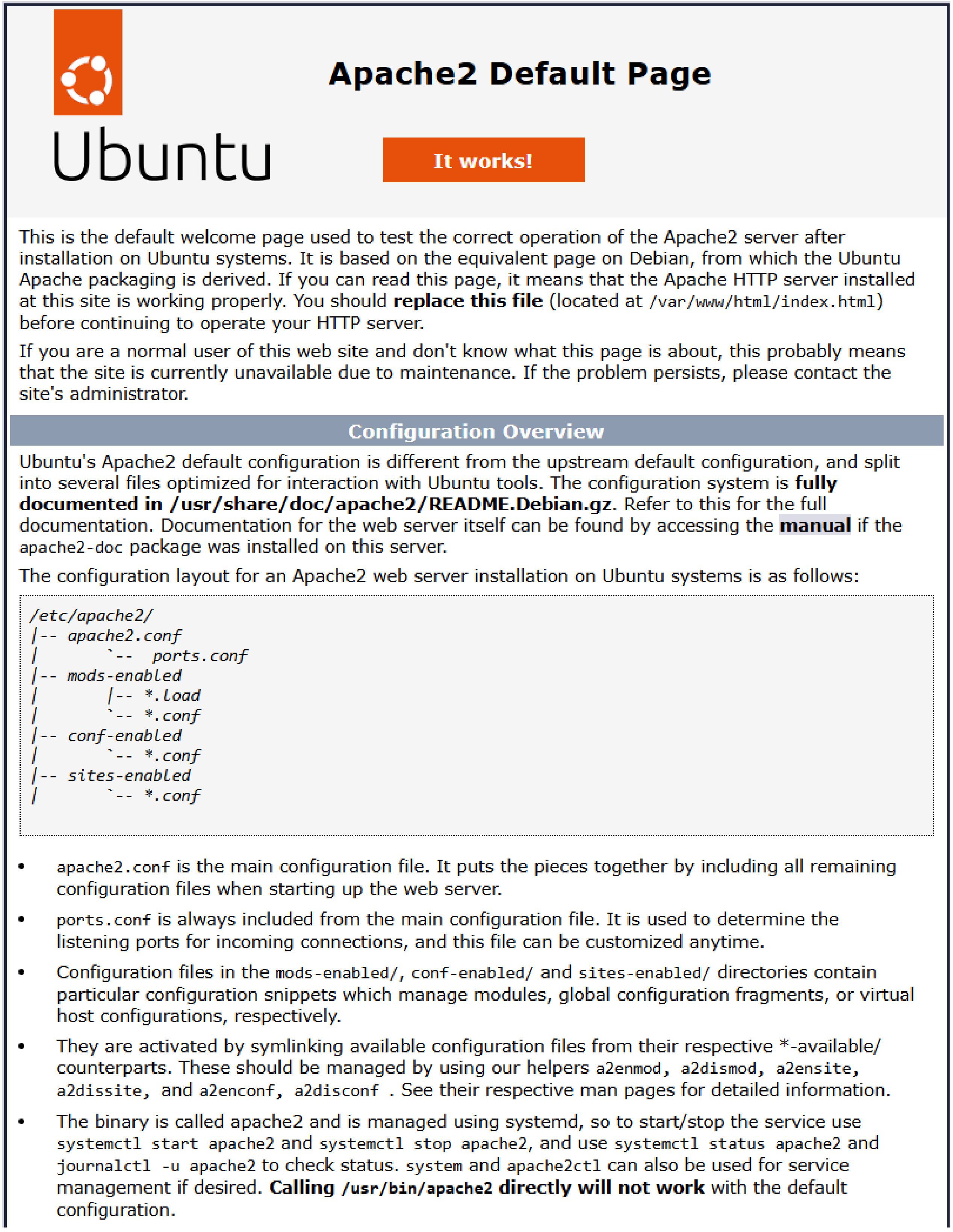
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:


Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

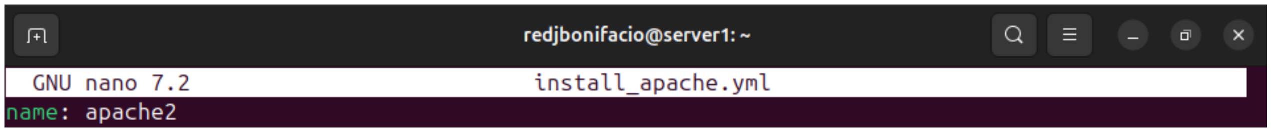
Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

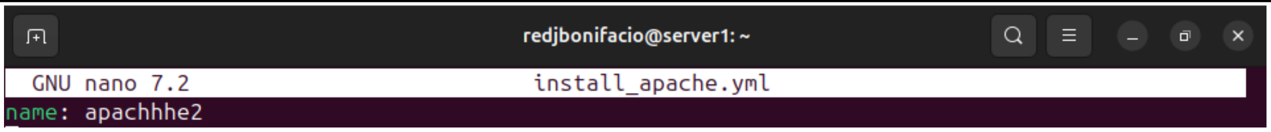
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf` . See their respective man pages for detailed information.
- The binary is called apache2 and is managed using systemd, so to start/stop the service use `systemctl start apache2` and `systemctl stop apache2`, and use `systemctl status apache2` and `journalctl -u apache2` to check status. `system` and `apache2ctl` can also be used for service management if desired. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

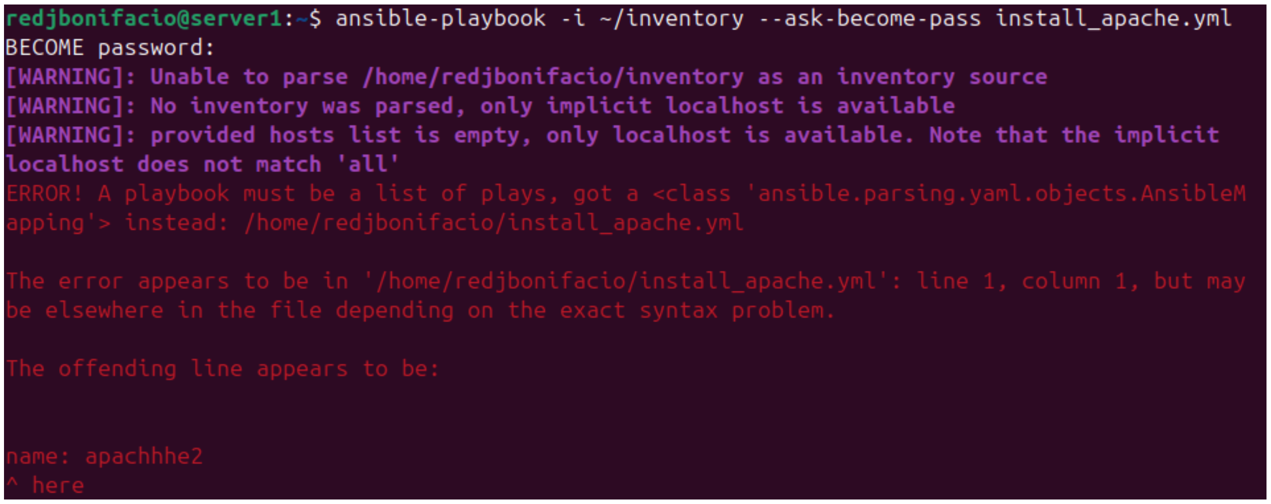
4. Try to edit the *install_apache.yml* and change the name of the package to any name that will not be recognized. What is the output?


redjbonifacio@server1: ~

```
  GNU nano 7.2                     install_apache.yml
name: apache2
```

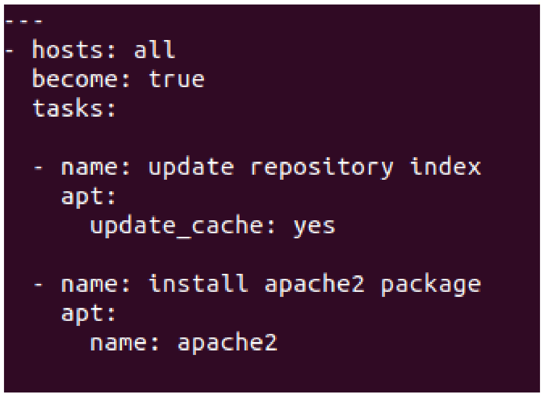


```
redjbonifacio@server1: ~
  GNU nano 7.2                         install_apache.yml
name: apachhhe2
```



```
redjbonifacio@server1:~$ ansible-playbook -i ~/inventory --ask-become-pass install_apache.yml
BECOME password:
[WARNING]: Unable to parse /home/redjbonifacio/inventory as an inventory source
[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'
ERROR! A playbook must be a list of plays, got a <class 'ansible.parsing.yaml.objects.AnsibleM
apping'> instead: /home/redjbonifacio/install_apache.yml

The error appears to be in '/home/redjbonifacio/install_apache.yml': line 1, column 1, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:


name: apachhhe2
^ here
```

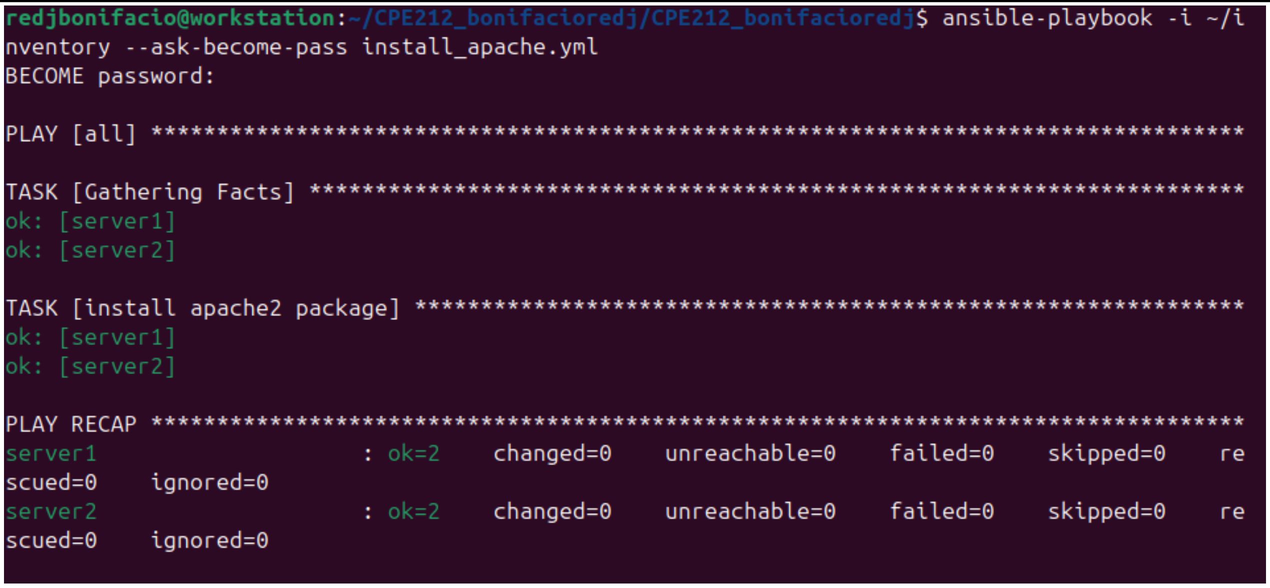
5. This time, we are going to put additional task to our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional command, which is the *update_cache.* This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities)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



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

  - name: update repository index
    apt:
      update_cache: yes

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

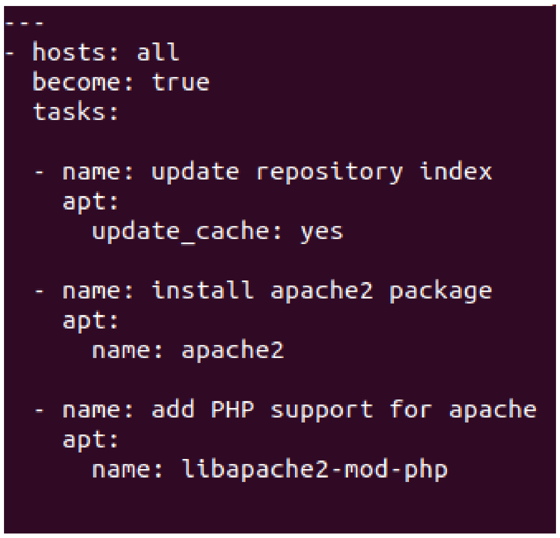
   Save the changes to this file and exit.

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



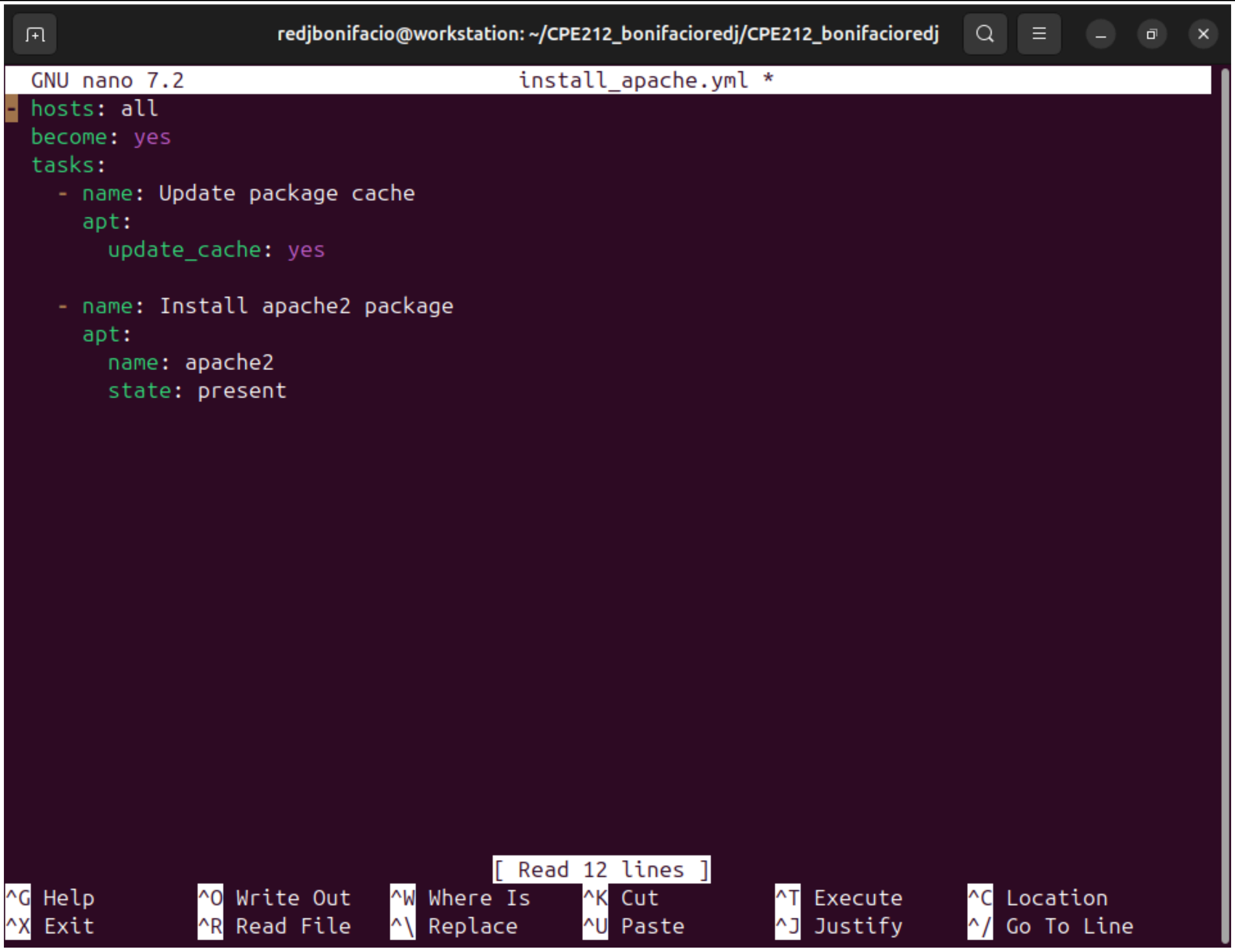
```
redjbonifacio@workstation:~/CPE212_bonifacioredj/CPE212_bonifacioredj$ ansible-playbook -i ~/i
nventory --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] ***********************************************************************
ok: [server1]
ok: [server2]

TASK [install apache2 package] ***************************************************************
ok: [server1]
ok: [server2]

PLAY RECAP ***********************************************************************************
server1                    : ok=2    changed=0    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
server2                    : ok=2    changed=0    unreachable=0    failed=0    skipped=0    re
scued=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
```

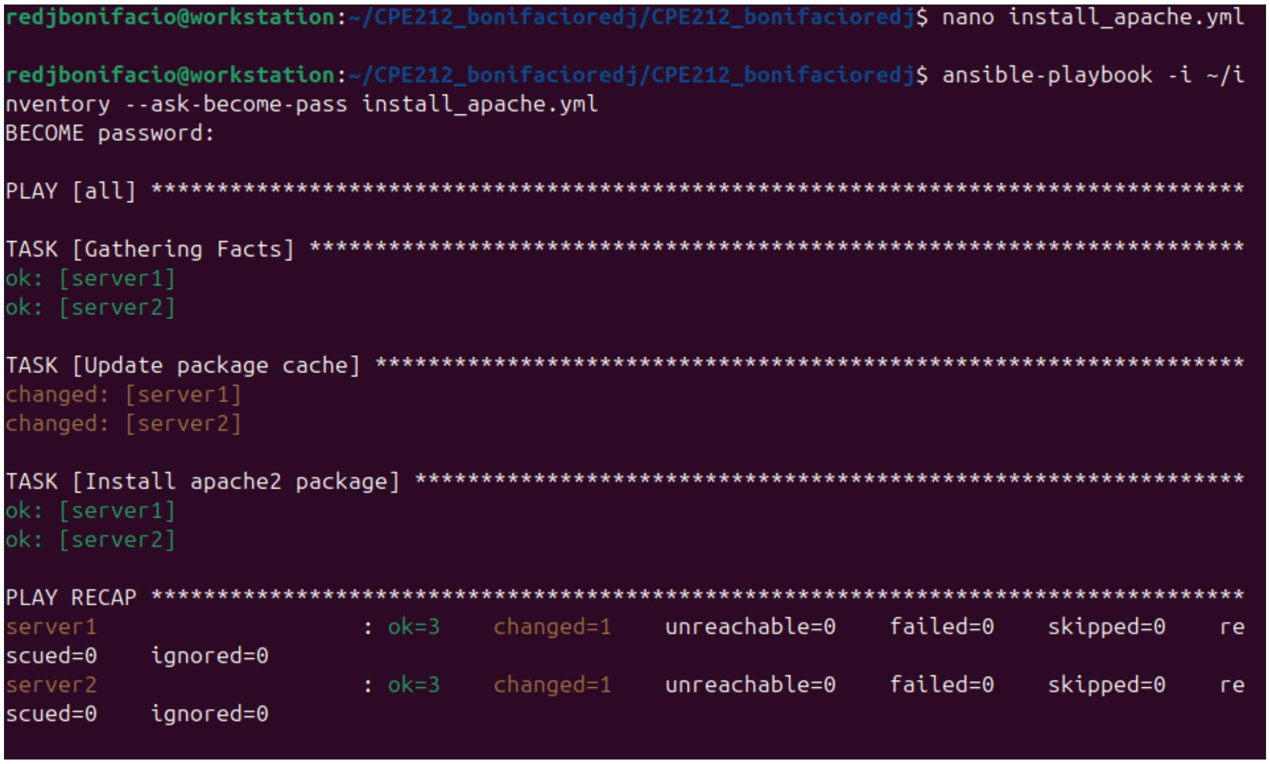
Save the changes to this file and exit.


redjbonifacio@workstation: ~/CPE212_bonifacioredj/CPE212_bonifacioredj

```
  GNU nano 7.2                        install_apache.yml *
- hosts: all
  become: yes
  tasks:
    - name: Update package cache
      apt:
        update_cache: yes

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

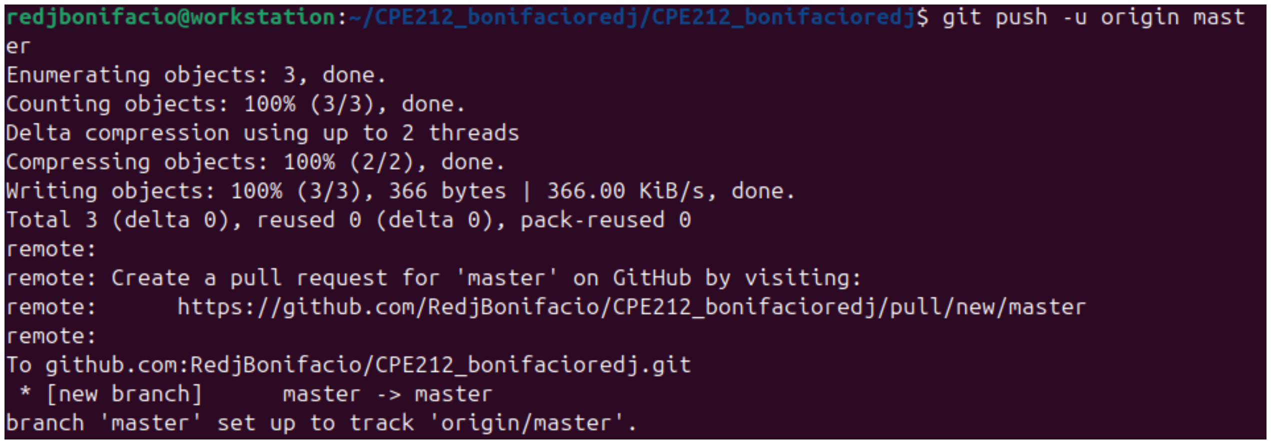
```
                                  [ Read 12 lines ]
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line
```

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



```
redjbonifacio@workstation:~/CPE212_bonifacioredj/CPE212_bonifacioredj$ nano install_apache.yml

redjbonifacio@workstation:~/CPE212_bonifacioredj/CPE212_bonifacioredj$ ansible-playbook -i ~/i
nventory --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] ***********************************************************************
ok: [server1]
ok: [server2]

TASK [Update package cache] ******************************************************************
changed: [server1]
changed: [server2]

TASK [Install apache2 package] ***************************************************************
ok: [server1]
ok: [server2]

PLAY RECAP ***********************************************************************************
server1                    : ok=3    changed=1    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
server2                    : ok=3    changed=1    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
```

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



```
redjbonifacio@workstation:~/CPE212_bonifacioredj/CPE212_bonifacioredj$ git push -u origin mast
er
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 366 bytes | 366.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/RedjBonifacio/CPE212_bonifacioredj/pull/new/master
remote:
To github.com:RedjBonifacio/CPE212_bonifacioredj.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

https://github.com/RedjBonifacio/CPE212_bonifacioredj/tree/master

Reflections:

Answer the following:

1. What is the importance of using a playbook?
   - Using a playbook in Ansible is important because it automates tasks across multiple Linux machines in a consistent, repeatable, and organized way. Instead of running commands manually, a playbook lets you define steps once and apply them everywhere, saving time and reducing human errors.
2. Summarize what we have done on this activity.
   - In this activity, we learned how to run elevated ad hoc commands in Ansible to update package information and install software like VIM and snapd on remote servers. We then created our first playbook to automate the installation of Apache2 and later added tasks for updating package indexes and installing PHP support. The activity showed how Ansible can manage multiple servers efficiently, make changes consistently, and reduce manual work.