Name: NICOLAS, KHELVIN P.	**Date Performed: 09/11/2024**
Course/Section: CPE 212 - CPE31S2	**Date Submitted: 09/16/2024**
Instructor: Engr. Robin Valenzuela	**Semester and SY: 1st Sem 2024- 2025**

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.

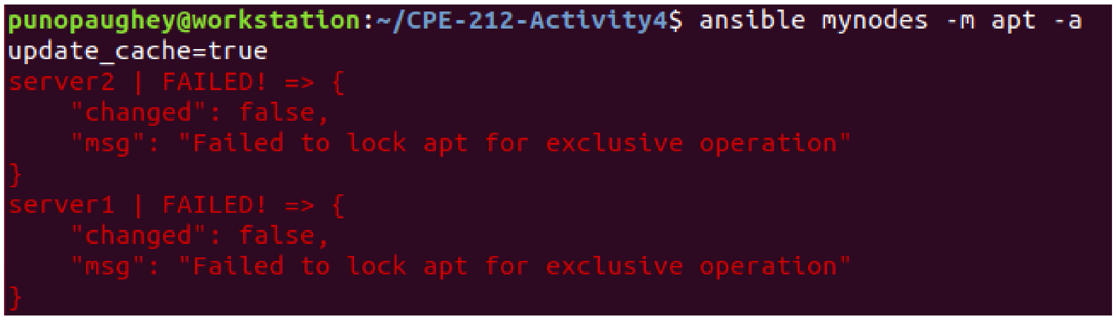
Elevated Ad hoc commands

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:

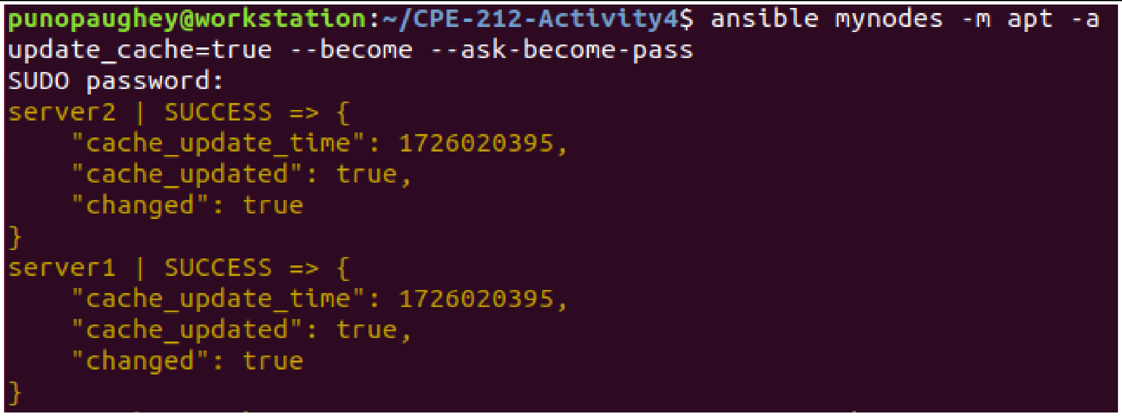


```
punopaughey@workstation:~/CPE-212-Activity4$ ansible mynodes -m apt -a
update_cache=true
server2 | FAILED! => {
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation"
}
server1 | FAILED! => {
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation"
}
```

ansible all -m apt -a update_cache=true
What is the result of the command? Is it successful?

- **The text is all red. No**

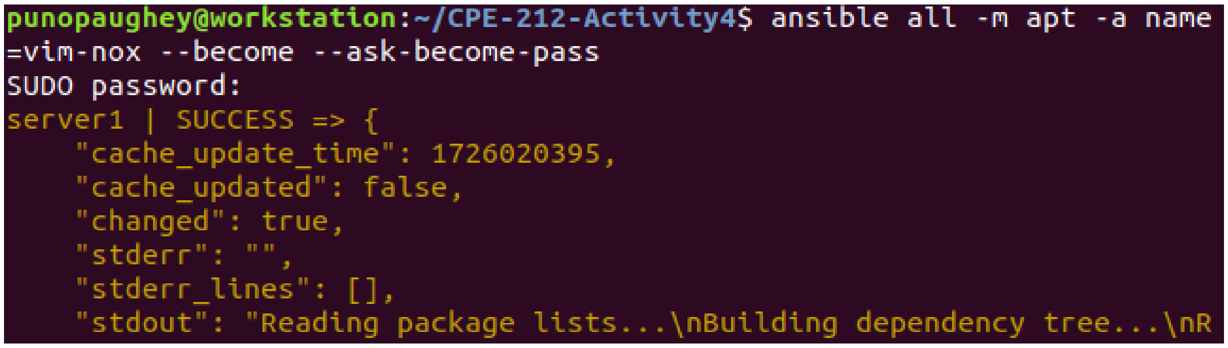
Try editing the command and add something that would elevate the privilege. Issue the command *ansible all -m apt -a update_cache=true --become --ask-become-pass.* Enter the sudo password when prompted. You will notice now that the output of this command is a success. The *update_cache=true* is the same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



```
punopaughey@workstation:~/CPE-212-Activity4$ ansible mynodes -m apt -a
update_cache=true --become --ask-become-pass
SUDO password:
server2 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": true,
    "changed": true
}
server1 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": true,
    "changed": true
}
```

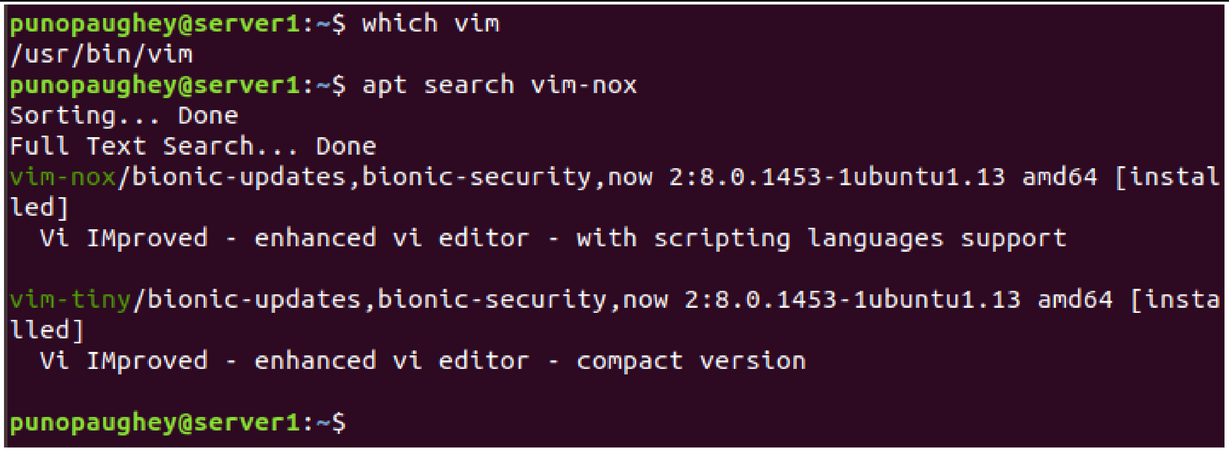
You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

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

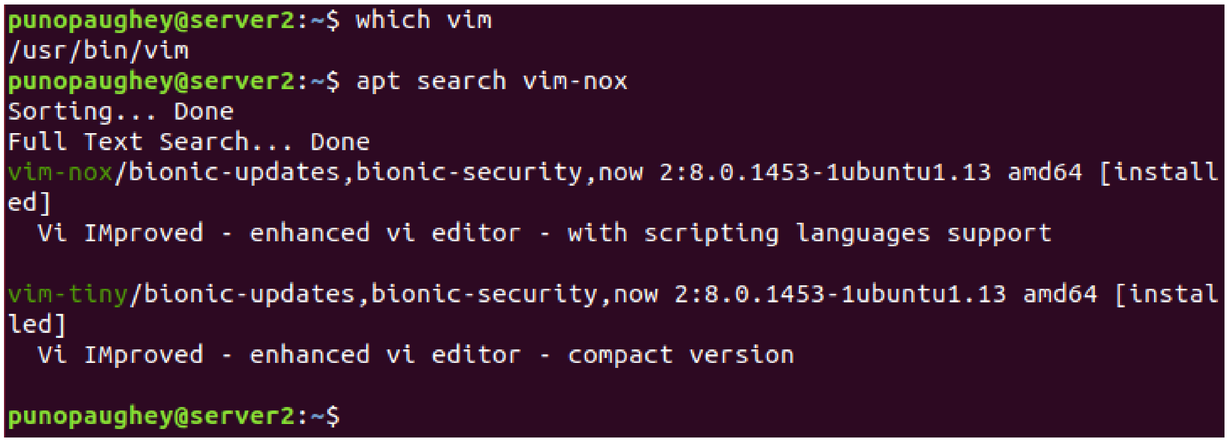


```
punopaughey@workstation:~/CPE-212-Activity4$ ansible all -m apt -a name
=vim-nox --become --ask-become-pass
SUDO password:
server1 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nR
```

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?

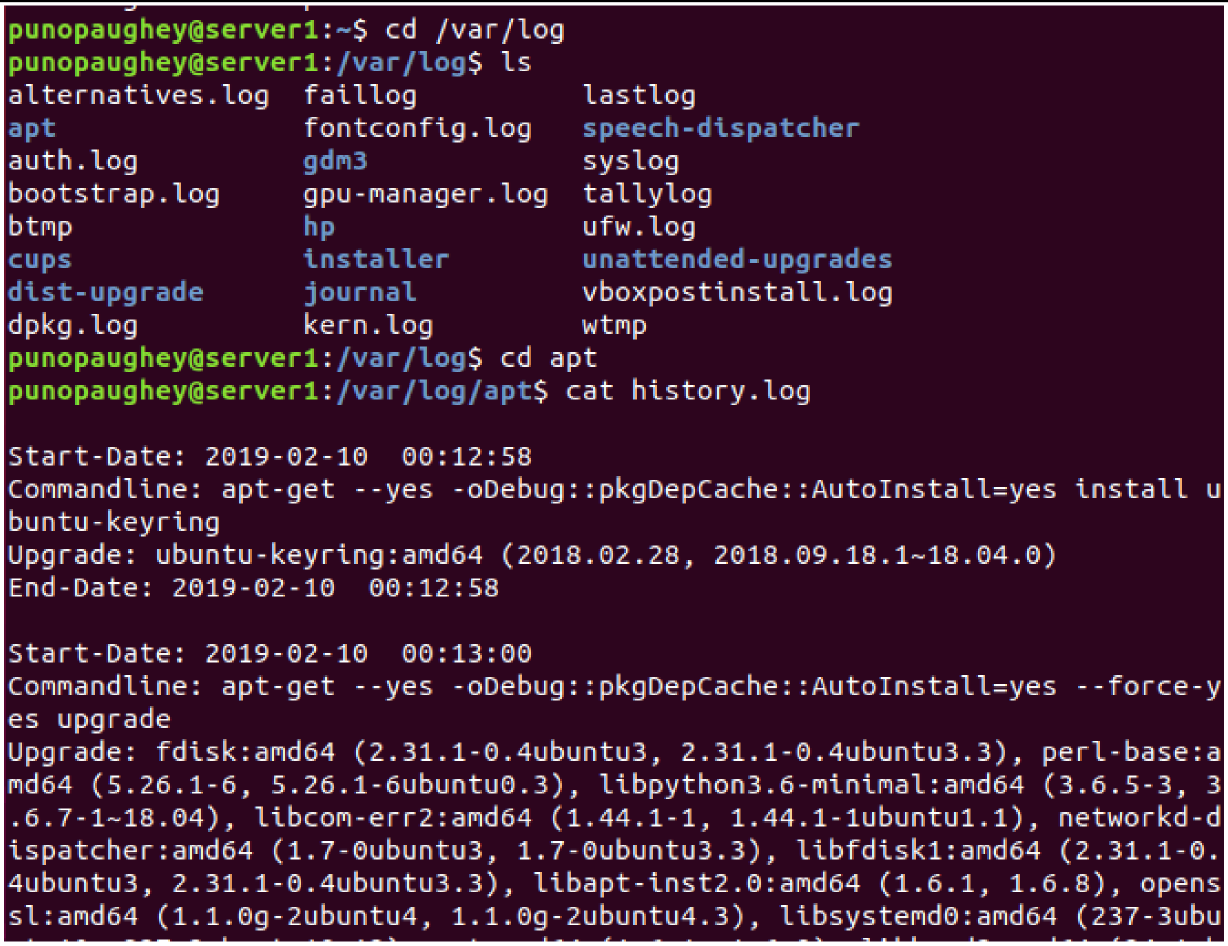


```
punopaughey@server1:~$ which vim
/usr/bin/vim
punopaughey@server1:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/bionic-updates,bionic-security,now 2:8.0.1453-1ubuntu1.13 amd64 [instal
led]
  Vi IMproved - enhanced vi editor - with scripting languages support

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

punopaughey@server1:~$
```



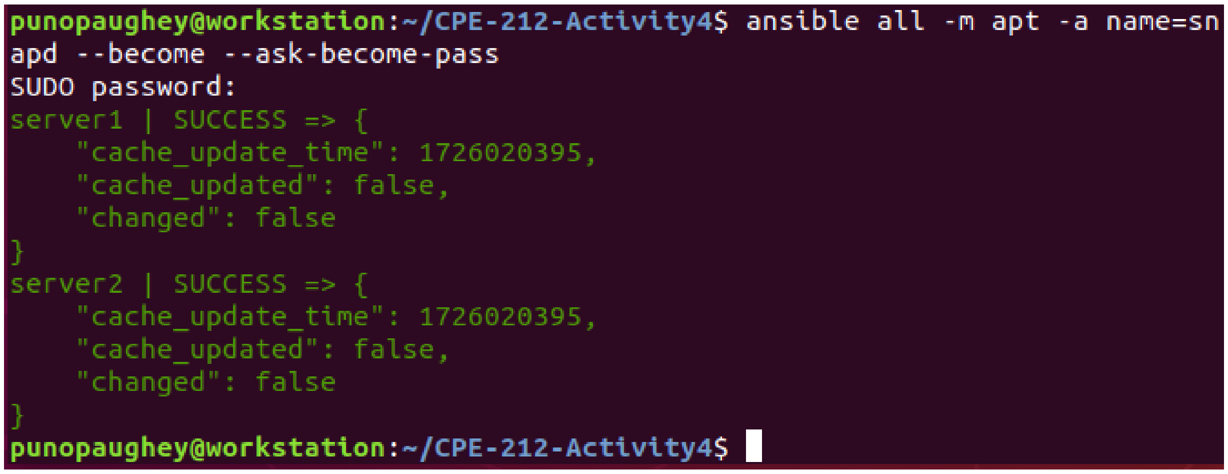
```
punopaughey@server2:~$ which vim
/usr/bin/vim
punopaughey@server2:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/bionic-updates,bionic-security,now 2:8.0.1453-1ubuntu1.13 amd64 [install
ed]
  Vi IMproved - enhanced vi editor - with scripting languages support

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

punopaughey@server2:~$
```

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.



```
punopaughey@server1:~$ cd /var/log
punopaughey@server1:/var/log$ ls
alternatives.log  faillog           lastlog
apt               fontconfig.log    speech-dispatcher
auth.log          gdm3              syslog
bootstrap.log     gpu-manager.log   tallylog
btmp              hp                ufw.log
cups              installer         unattended-upgrades
dist-upgrade      journal           vboxpostinstall.log
dpkg.log          kern.log          wtmp
punopaughey@server1:/var/log$ cd apt
punopaughey@server1:/var/log/apt$ cat history.log

Start-Date: 2019-02-10  00:12:58
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes install u
buntu-keyring
Upgrade: ubuntu-keyring:amd64 (2018.02.28, 2018.09.18.1~18.04.0)
End-Date: 2019-02-10  00:12:58

Start-Date: 2019-02-10  00:13:00
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes --force-y
es upgrade
Upgrade: fdisk:amd64 (2.31.1-0.4ubuntu3, 2.31.1-0.4ubuntu3.3), perl-base:a
md64 (5.26.1-6, 5.26.1-6ubuntu0.3), libpython3.6-minimal:amd64 (3.6.5-3, 3
.6.7-1~18.04), libcom-err2:amd64 (1.44.1-1, 1.44.1-1ubuntu1.1), networkd-d
ispatcher:amd64 (1.7-0ubuntu3, 1.7-0ubuntu3.3), libfdisk1:amd64 (2.31.1-0.
4ubuntu3, 2.31.1-0.4ubuntu3.3), libapt-inst2.0:amd64 (1.6.1, 1.6.8), opens
sl:amd64 (1.1.0g-2ubuntu4, 1.1.0g-2ubuntu4.3), libsystemd0:amd64 (237-3ubu
```

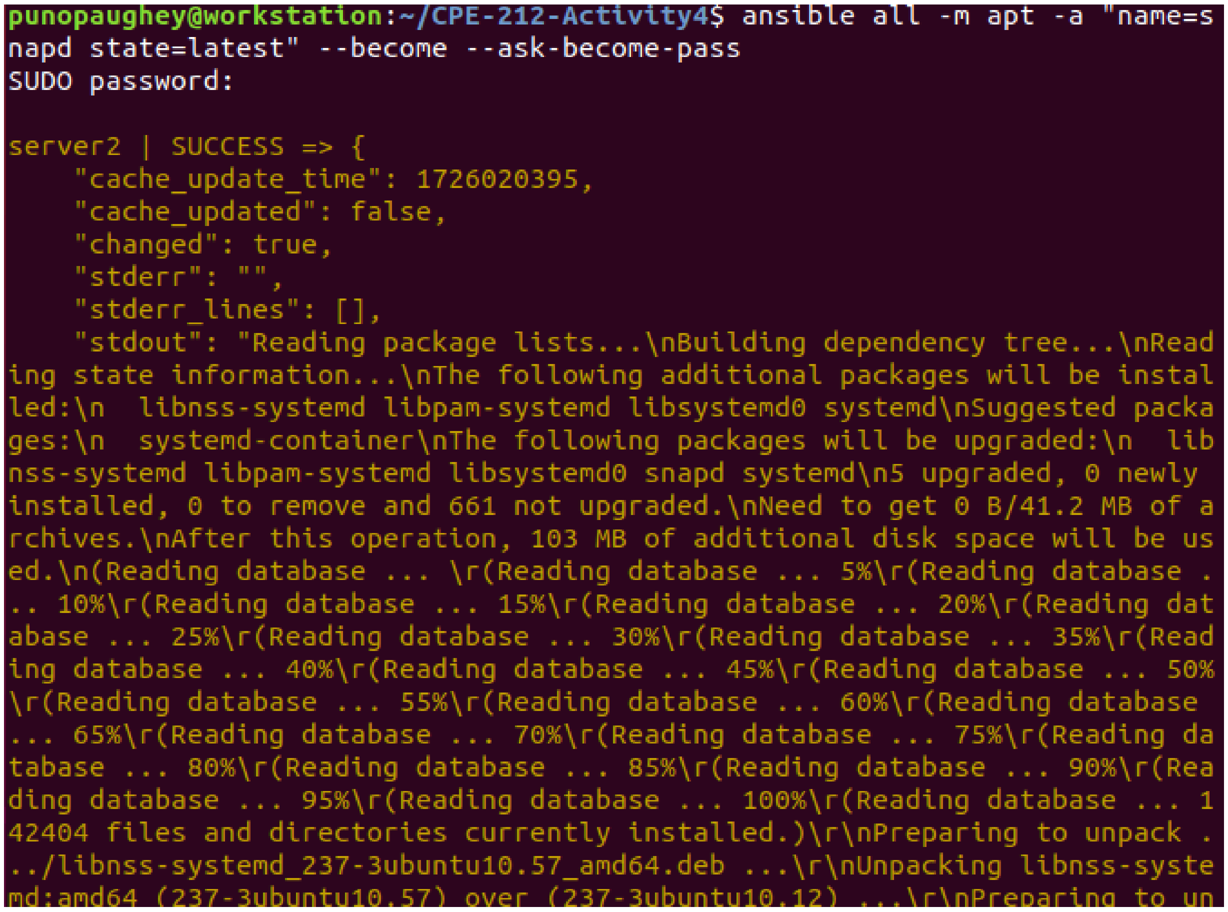
3. This time, we will install a package called snapd. Snap is pre-installed in Ubuntu system. However, our goal is to create a command that checks for the latest installation package.



```
punopaughey@workstation:~/CPE-212-Activity4$ ansible all -m apt -a name=sn
apd --become --ask-become-pass
SUDO password:
server1 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": false,
    "changed": false
}
punopaughey@workstation:~/CPE-212-Activity4$
```

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?



```
punopaughey@workstation:~/CPE-212-Activity4$ ansible all -m apt -a "name=s
napd state=latest" --become --ask-become-pass
SUDO password:

server2 | SUCCESS => {
    "cache_update_time": 1726020395,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nRead
ing state information...\nThe following additional packages will be instal
led:\n  libnss-systemd libpam-systemd libsystemd0 systemd\nSuggested packa
ges:\n  systemd-container\nThe following packages will be upgraded:\n  lib
nss-systemd libpam-systemd libsystemd0 snapd systemd\n5 upgraded, 0 newly
installed, 0 to remove and 661 not upgraded.\nNeed to get 0 B/41.2 MB of a
rchives.\nAfter this operation, 103 MB of additional disk space will be us
ed.\n(Reading database ... \r(Reading database ... 5%\r(Reading database .
.. 10%\r(Reading database ... 15%\r(Reading database ... 20%\r(Reading dat
abase ... 25%\r(Reading database ... 30%\r(Reading database ... 35%\r(Read
ing database ... 40%\r(Reading database ... 45%\r(Reading database ... 50%
\r(Reading database ... 55%\r(Reading database ... 60%\r(Reading database
... 65%\r(Reading database ... 70%\r(Reading database ... 75%\r(Reading da
tabase ... 80%\r(Reading database ... 85%\r(Reading database ... 90%\r(Rea
ding database ... 95%\r(Reading database ... 100%\r(Reading database ... 1
42404 files and directories currently installed.)\r\nPreparing to unpack .
../libnss-systemd_237-3ubuntu10.57_amd64.deb ...\r\nUnpacking libnss-syste
md:amd64 (237-3ubuntu10.57) over (237-3ubuntu10.12) ...\r\nPreparing to un
```

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.

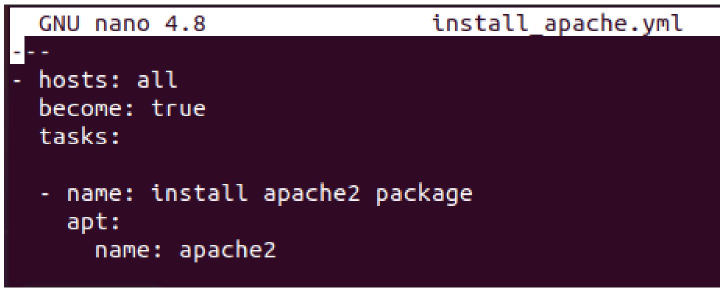
4. At this point, make sure to commit all changes to GitHub.

Note: Task 1 was done during face to face classes, Task 2 will be taken at home.

Task 2: Writing our First Playbook

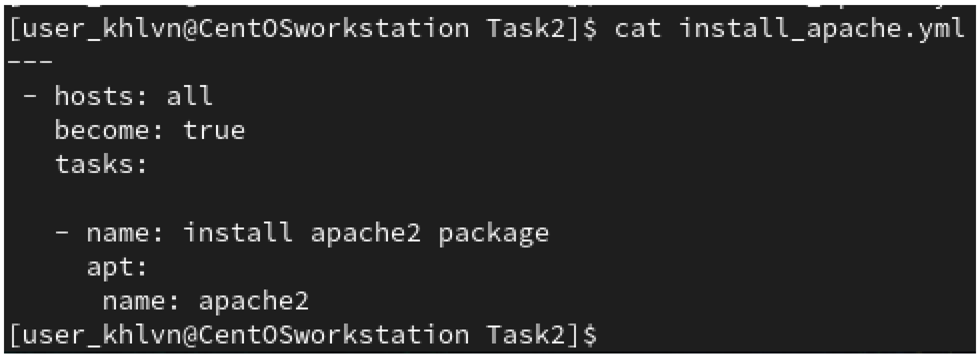
1. With ad hoc commands, we can simplify the administration of remote servers. For example, we can install updates, packages, and applications, etc. However, the real strength of Ansible comes from its playbooks. When we write a playbook, we can define the state that we want our servers to be in and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

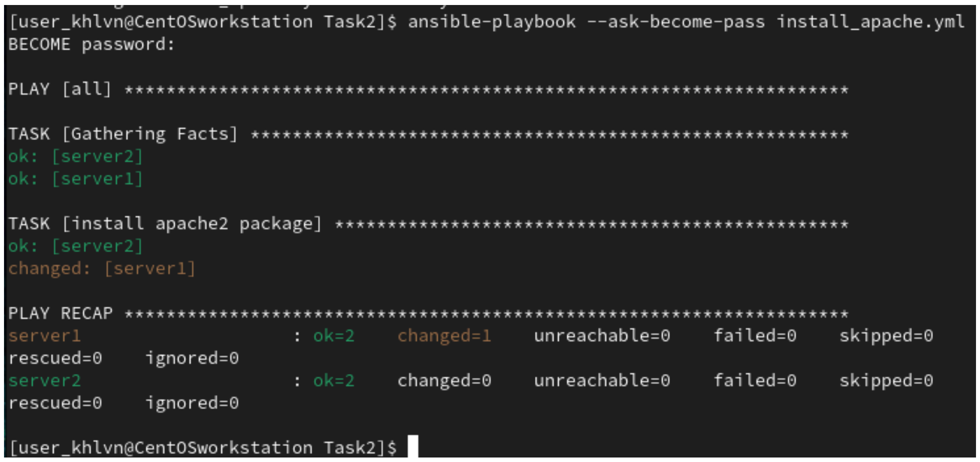


```
[user_khlvn@CentOSworkstation Task2]$ cat install_apache.yml
---
 - hosts: all
   become: true
   tasks:

   - name: install apache2 package
     apt:
      name: apache2
[user_khlvn@CentOSworkstation Task2]$
```

figure 2.1: writing the first ansible playbook

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

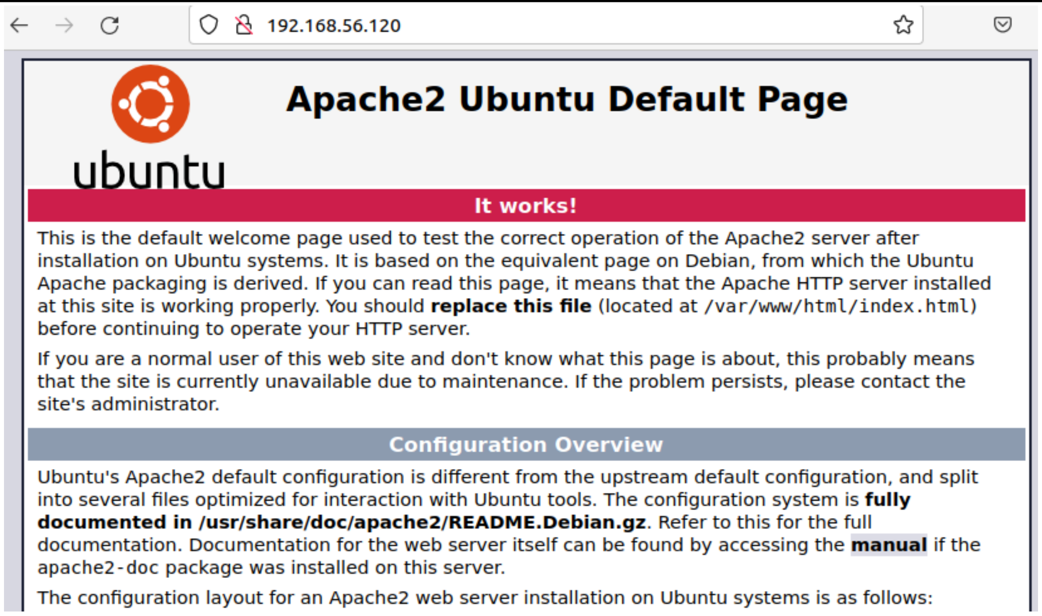
   - **The command asked first for the password, then pings the remote servers to check the connectivity, and executes the task written in the .yml file.**



```
[user_khlvn@CentOSworkstation Task2]$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

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

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

[user_khlvn@CentOSworkstation Task2]$
```

figure 2.2: using first playbook that installs apache2 package

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.



Apache2 Ubuntu Default Page

It works!

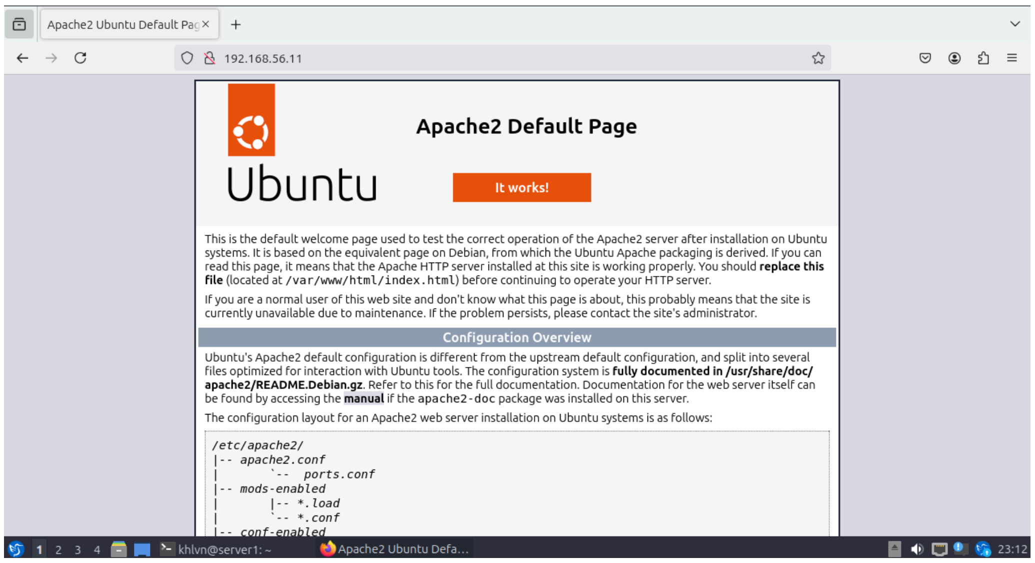
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at 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

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.

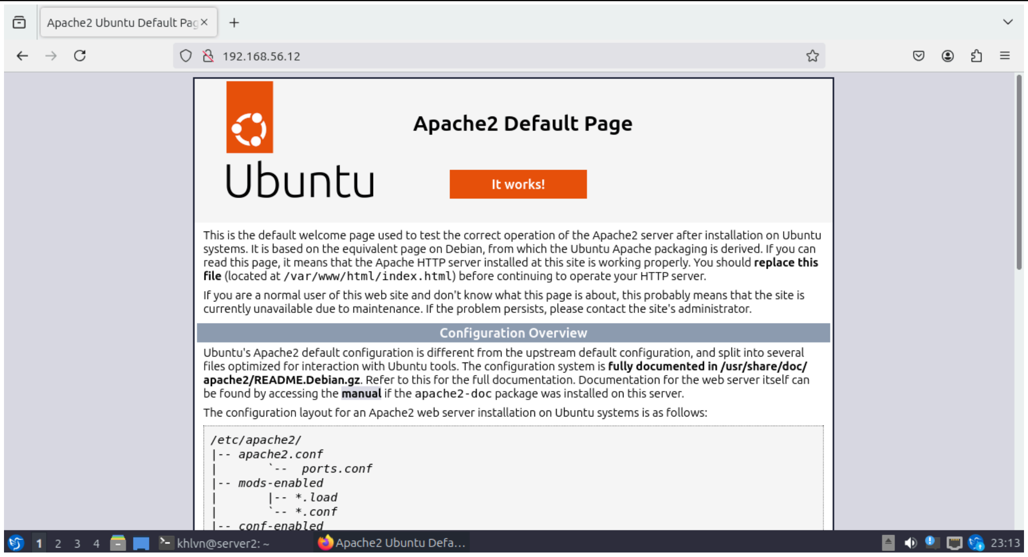
Configuration Overview

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
```

figure 2.3.1: Installed Apache2 package on server1



Apache2 Ubuntu Default Page

192.168.56.12

Ubuntu

Apache2 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.

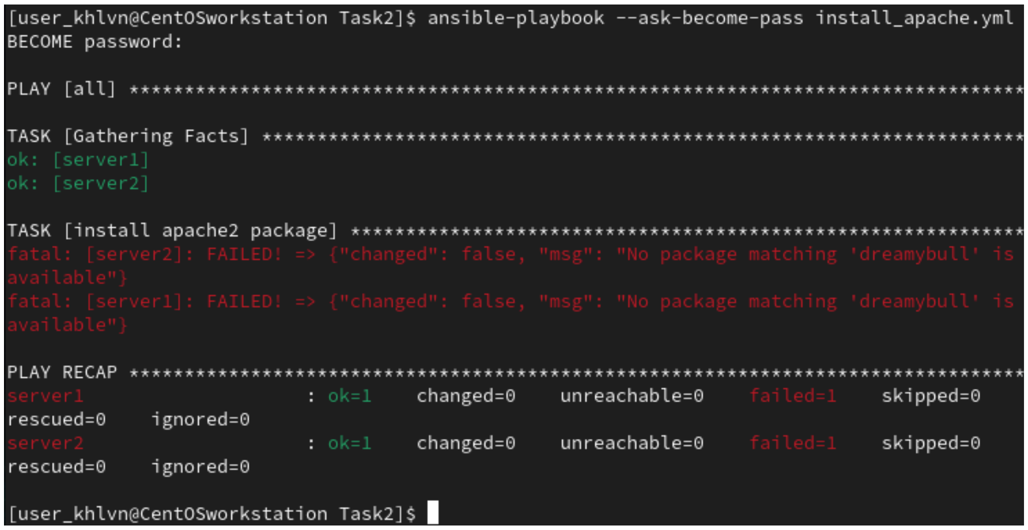
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

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

1 2 3 4 khlvn@server2: ~ Apache2 Ubuntu Defa... 23:13

figure 2.3.2: Installed Apache2 package on server2

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?



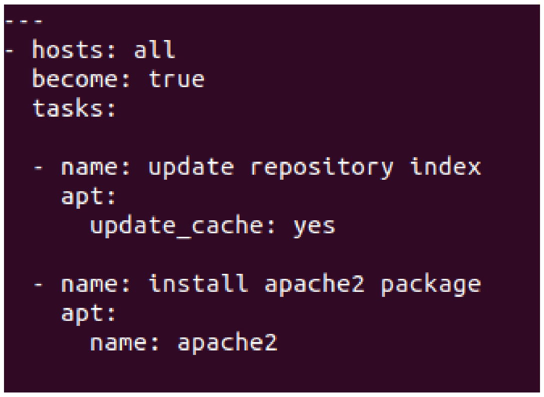
```
[user_khlvn@CentOSworkstation Task2]$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

TASK [install apache2 package] ********************************************************
fatal: [server2]: FAILED! => {"changed": false, "msg": "No package matching 'dreamybull' is
available"}
fatal: [server1]: FAILED! => {"changed": false, "msg": "No package matching 'dreamybull' is
available"}

PLAY RECAP ****************************************************************************
server1                    : ok=1    changed=0    unreachable=0    failed=1    skipped=0
rescued=0    ignored=0
server2                    : ok=1    changed=0    unreachable=0    failed=1    skipped=0
rescued=0    ignored=0

[user_khlvn@CentOSworkstation Task2]$
```

figure 2.4: changing the name of the package to any unrecognized name.

- **The output failed because I changed the package into something that isn't recognized.**

5. This time, we are going to put additional tasks into our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional

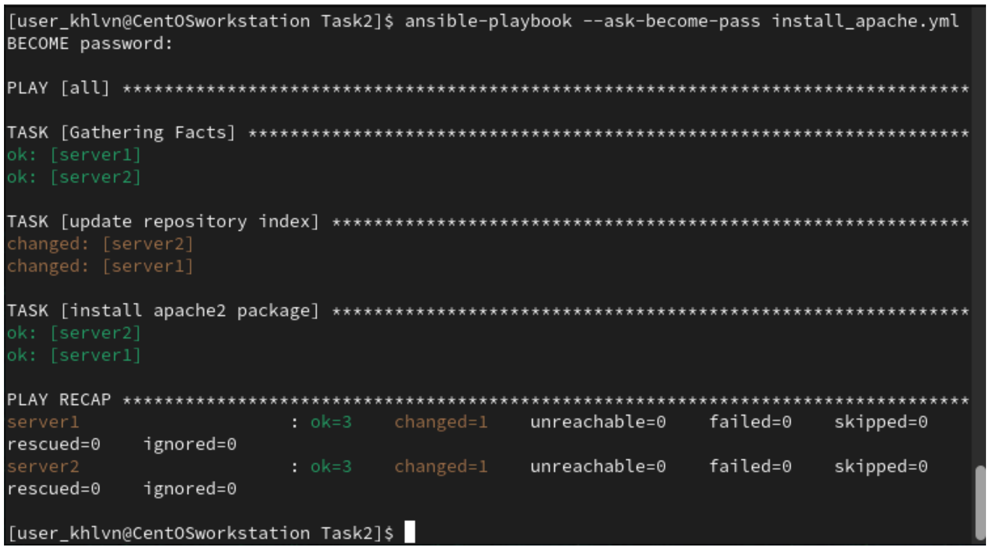
command, which is the *update_cache*. This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities) that were being installed.



```
---
- 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?

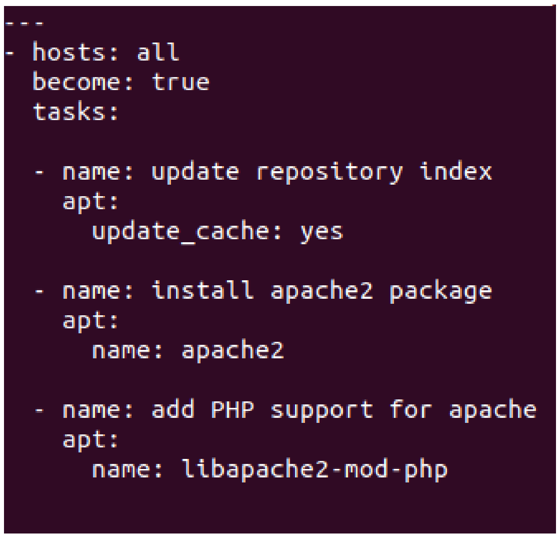


```
[user_khlvn@CentOSworkstation Task2]$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

TASK [update repository index] *************************************************
changed: [server2]
changed: [server1]

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

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

[user_khlvn@CentOSworkstation Task2]$
```

figure 2.5: added "update repository index" task

- **Yes, it updated the cache of the repository index.**

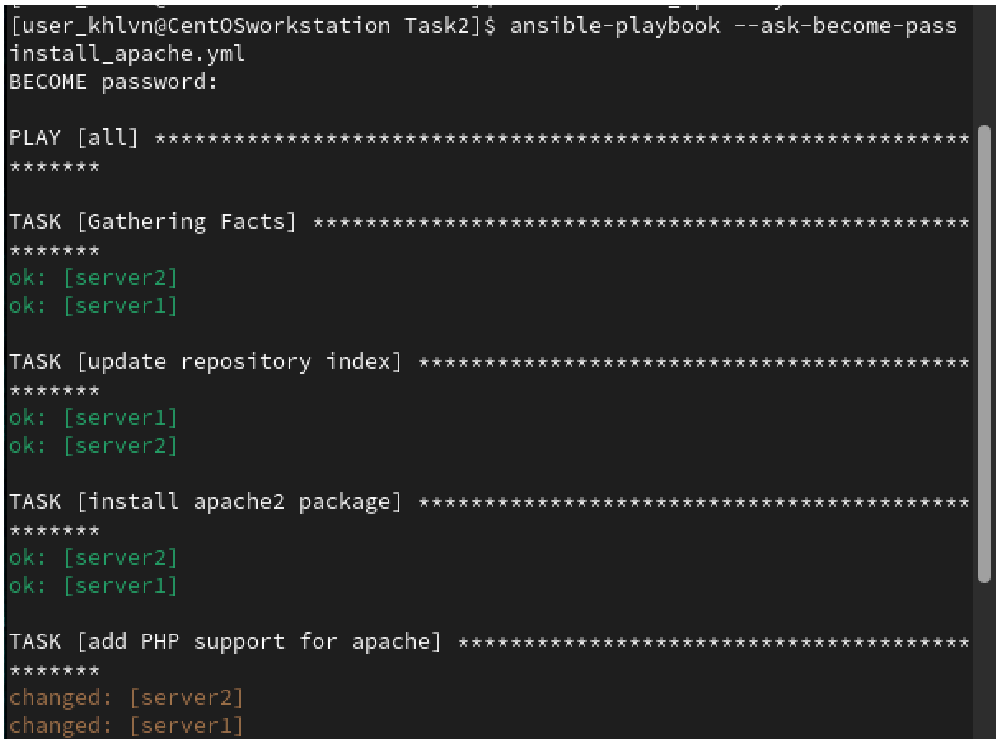
7. Edit again the *install_apache.yml*. This time, we are going to add a PHP support for the apache package we installed earlier.



```
---
- 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?



```
[user_khlvn@CentOSworkstation Task2]$ ansible-playbook --ask-become-pass
install_apache.yml
BECOME password:

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

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

TASK [update repository index] *************************************************
*******
ok: [server1]
ok: [server2]

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

TASK [add PHP support for apache] **********************************************
*******
changed: [server2]
changed: [server1]
```

figure 2.6: added "add PHP support for apache" task

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

   Github Link: https://github.com/KHLVN/CPE-212-Activity4

Reflections:

Answer the following:

1. **What is the importance of using a playbook?**
   - Playbooks are important because they automate tasks that need to be done on servers. They allow for consistent, streamline deployment of packages, and management of multiple servers using just one local machine. It uses YAML format which is a convenient tool since it is not a strongly typed language like Java, C++, and Python, and is usually used as configuration files in other tools.
2. **Summarize what we have done on this activity.**
   - We created our first playbook, how it executes to manage servers, and how to write them using YAML. Using these learnings, we are able to install apache2 package and added PHP support on the remote servers using playbook through Ansible. We also updated the repository index of the OS installed in the remote servers.