Name: Planta, Calvin Earl L.	**Date Performed: Aug 27, 2025**
Course/Section: CPE 212 - CPE31S2	**Date Submitted: Aug 27, 2025**
Instructor: Engr. Robin Valenzuela	**Semester and SY: 1st Sem 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 make changes to the remote servers. We managed to gather facts and connect to the remote machines, but we still did not make changes on those machines. In this activity, we will learn to use commands that would install, update, and upgrade packages in the remote machines. We will also create a playbook that will be used for automations.

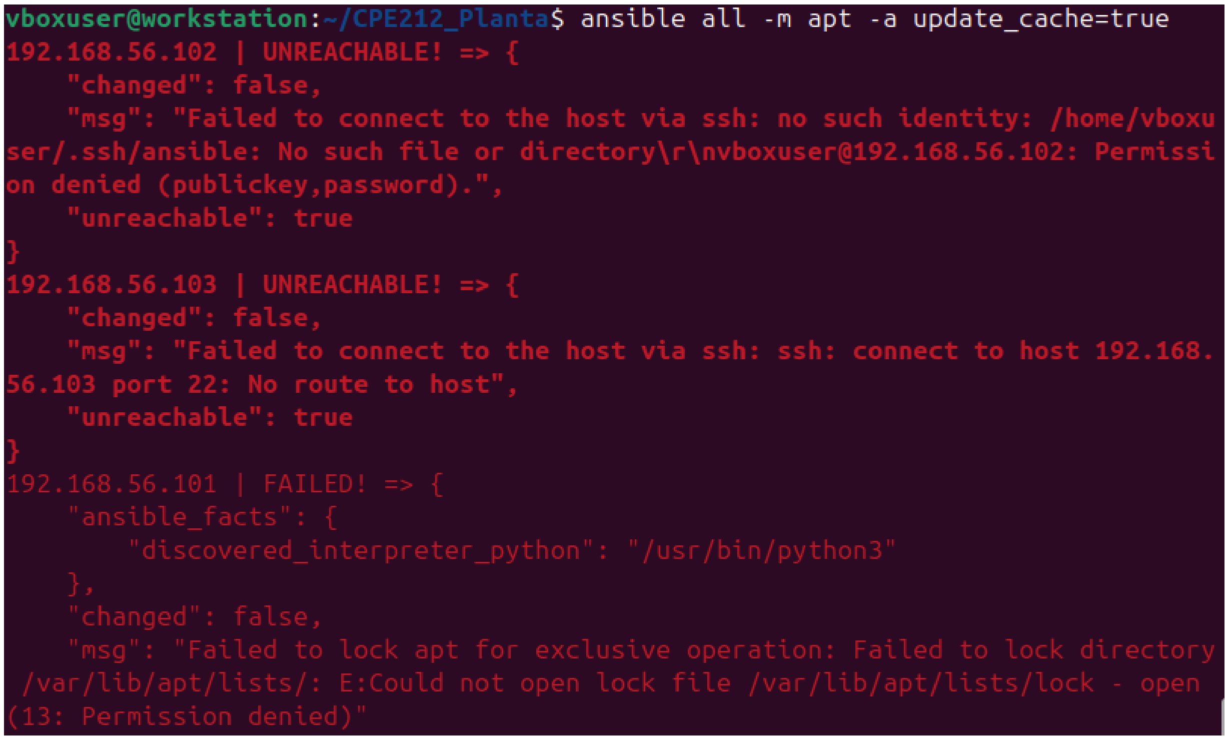
Playbooks record and execute **Ansible**'s configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material. At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way. You can check this documentation if you want to learn more about playbooks. Working with playbooks — Ansible Documentation

Task 1: Run elevated ad hoc commands

1. Locally, we use the command *sudo apt update* when we want to download package information from all configured resources. The sources are often defined in /etc/apt/sources.list file and other files located in /etc/apt/sources.list.d/ directory. So, when you run the update command, it downloads the package information from the Internet. It is useful to get info on an updated version of packages or their dependencies. We can only run

an apt update command in a remote machine. Issue the following command:

ansible all -m apt -a update_cache=true

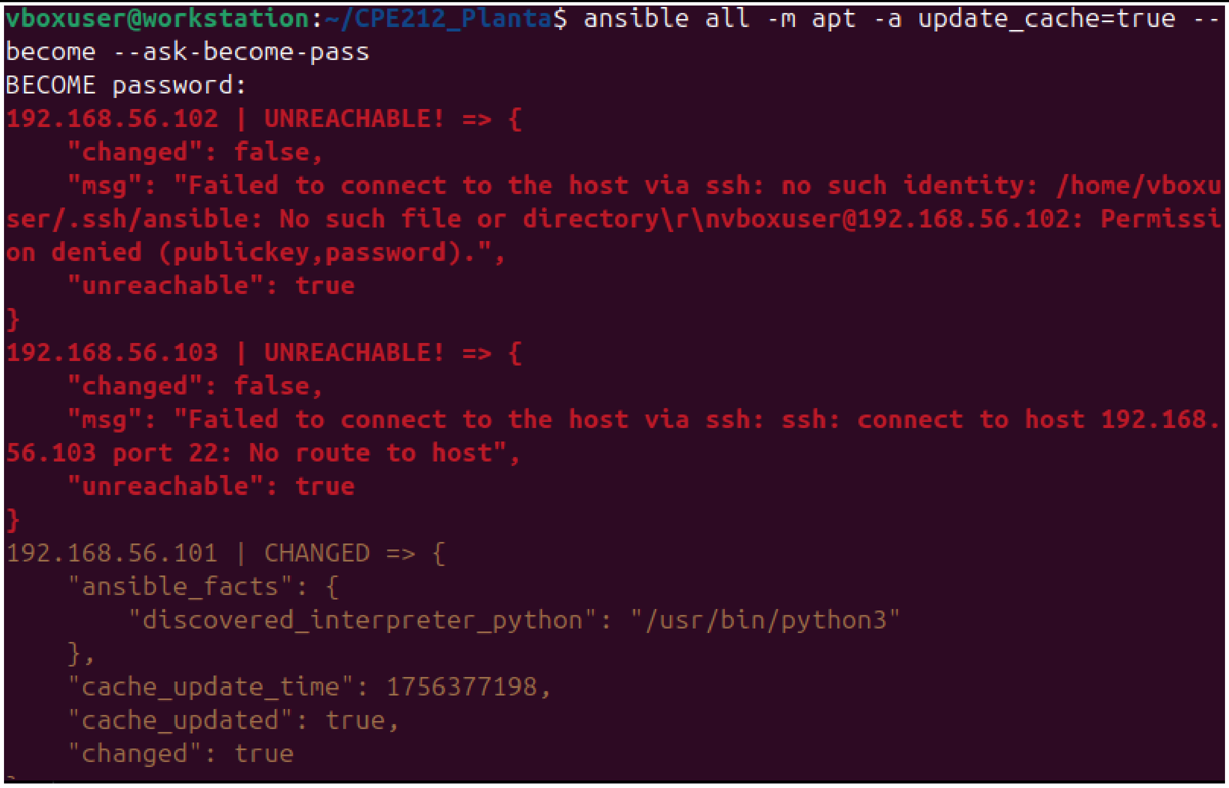


```
vboxuser@workstation:~/CPE212_Planta$ ansible all -m apt -a update_cache=true
192.168.56.102 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: no such identity: /home/vboxu
ser/.ssh/ansible: No such file or directory\r\nvboxuser@192.168.56.102: Permissi
on denied (publickey,password).",
    "unreachable": true
}
192.168.56.103 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168.
56.103 port 22: No route to host",
    "unreachable": true
}
192.168.56.101 | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock directory
 /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)"
```

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

- The command resulted in a failure, with an error message that reads “failed to lock apt for exclusive operation”.

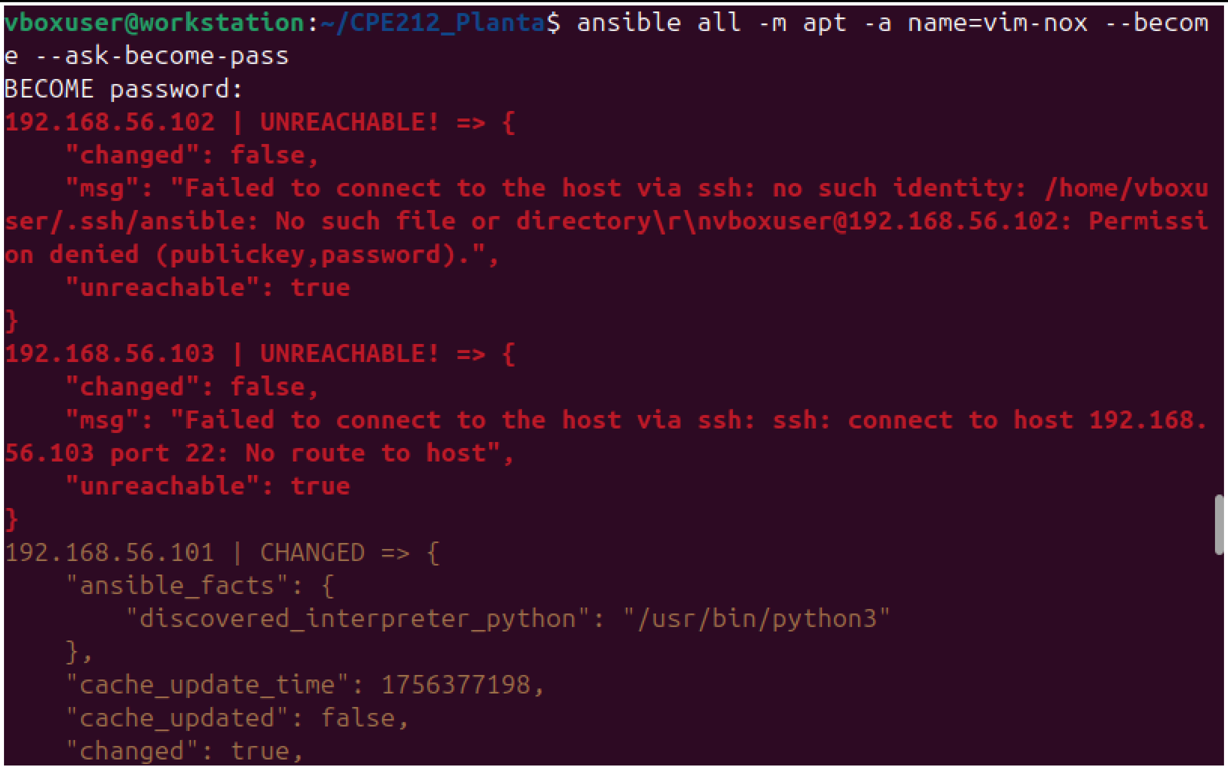
Try editing the command and add something that would elevate the privilege. Issue the command *ansible all -m apt -a update_cache=true --become --ask-become-pass.* Enter the sudo password when prompted. You will notice now that the output of this command is a success. The *update_cache=true* is the same thing as running *sudo apt update*. The --become command elevates the privileges and the *--ask-become-pass* asks for the password. For now, even if we only changed the packaged index, we were able to change something on the remote server.



```
vboxuser@workstation:~/CPE212_Planta$ ansible all -m apt -a update_cache=true --
become --ask-become-pass
BECOME password:
192.168.56.102 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: no such identity: /home/vboxu
ser/.ssh/ansible: No such file or directory\r\nvboxuser@192.168.56.102: Permissi
on denied (publickey,password).",
    "unreachable": true
}
192.168.56.103 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168.
56.103 port 22: No route to host",
    "unreachable": true
}
192.168.56.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756377198,
    "cache_updated": true,
    "changed": true
```

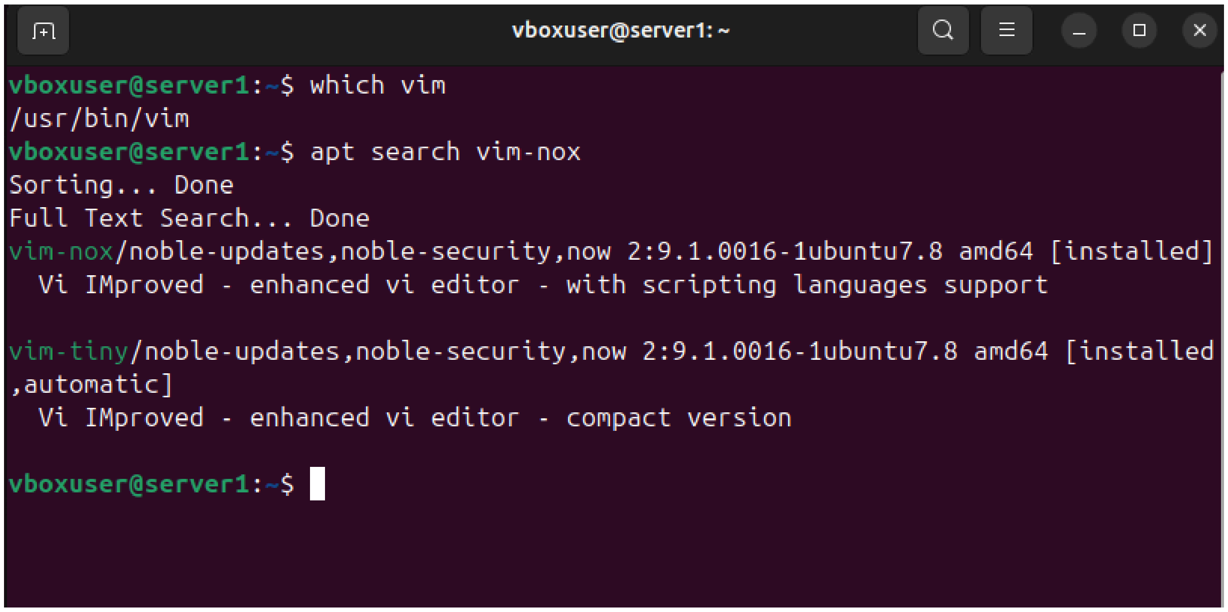
You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

2. Let's try to install VIM, which is an almost compatible version of the UNIX editor Vi. To do this, we will just 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.



```
vboxuser@workstation:~/CPE212_Planta$ ansible all -m apt -a name=vim-nox --becom
e --ask-become-pass
BECOME password:
192.168.56.102 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: no such identity: /home/vboxu
ser/.ssh/ansible: No such file or directory\r\nvboxuser@192.168.56.102: Permissi
on denied (publickey,password).",
    "unreachable": true
}
192.168.56.103 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168.
56.103 port 22: No route to host",
    "unreachable": true
}
192.168.56.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756377198,
    "cache_updated": false,
    "changed": true,
```

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?

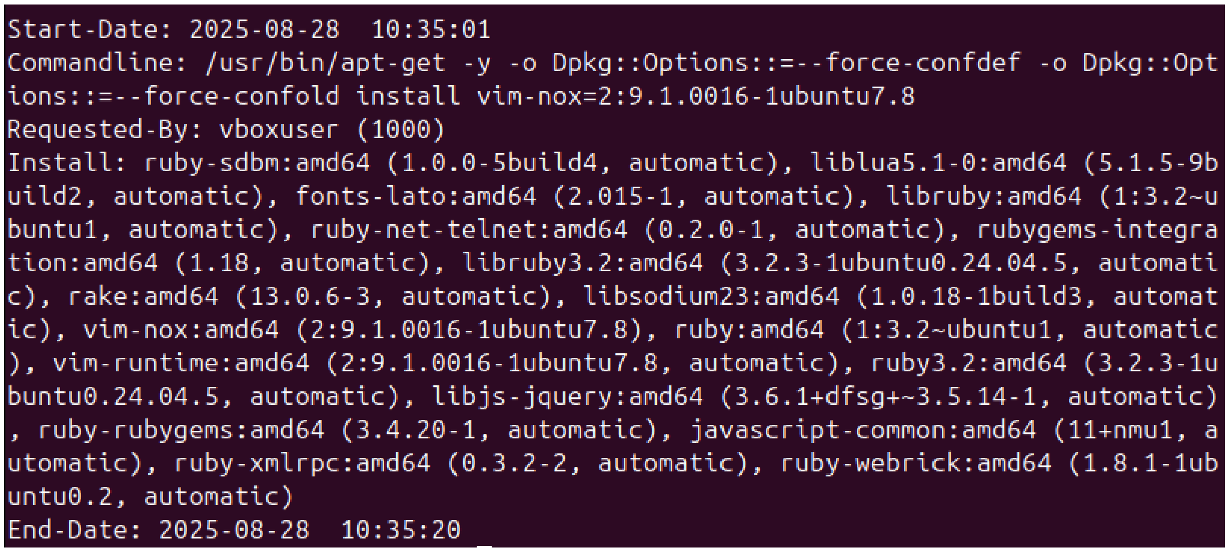


vboxuser@server1: ~

```
vboxuser@server1:~$ which vim
/usr/bin/vim
vboxuser@server1:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/noble-updates,noble-security,now 2:9.1.0016-1ubuntu7.8 amd64 [installed]
  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

vboxuser@server1:~$ 
```

-The command was successful, and this confirms that vim-nox was successfully installed in the remote server

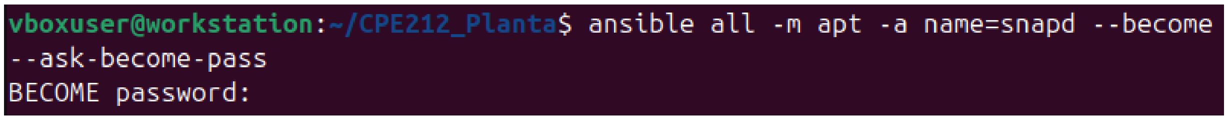
2.2 Check the logs in the servers using the following commands: *cd /var/log*. After this, issue the command *ls,* go to the folder *apt* and open history.log. Describe what you see in the history.log.



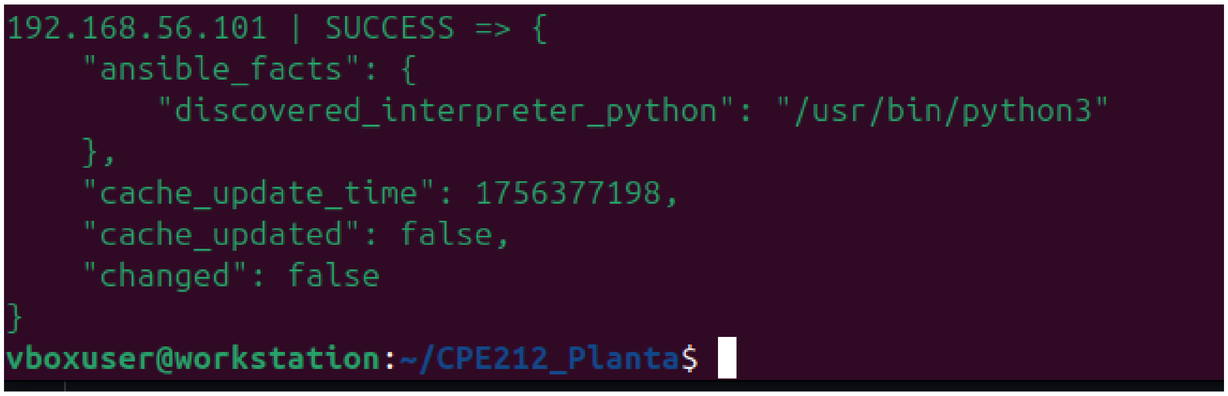
```
Start-Date: 2025-08-28  10:35:01
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Opt
ions::=--force-confold install vim-nox=2:9.1.0016-1ubuntu7.8
Requested-By: vboxuser (1000)
Install: ruby-sdbm:amd64 (1.0.0-5build4, automatic), liblua5.1-0:amd64 (5.1.5-9b
uild2, automatic), fonts-lato:amd64 (2.015-1, automatic), libruby:amd64 (1:3.2~u
buntu1, automatic), ruby-net-telnet:amd64 (0.2.0-1, automatic), rubygems-integra
tion:amd64 (1.18, automatic), libruby3.2:amd64 (3.2.3-1ubuntu0.24.04.5, automati
c), rake:amd64 (13.0.6-3, automatic), libsodium23:amd64 (1.0.18-1build3, automat
ic), vim-nox:amd64 (2:9.1.0016-1ubuntu7.8), ruby:amd64 (1:3.2~ubuntu1, automatic
), vim-runtime:amd64 (2:9.1.0016-1ubuntu7.8, automatic), ruby3.2:amd64 (3.2.3-1u
buntu0.24.04.5, automatic), libjs-jquery:amd64 (3.6.1+dfsg+~3.5.14-1, automatic)
, ruby-rubygems:amd64 (3.4.20-1, automatic), javascript-common:amd64 (11+nmu1, a
utomatic), ruby-xmlrpc:amd64 (0.3.2-2, automatic), ruby-webrick:amd64 (1.8.1-1ub
untu0.2, automatic)
End-Date: 2025-08-28  10:35:20
```

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

   3.1 Issue the command: *ansible all -m apt -a name=snapd --become --ask-become-pass*



```
vboxuser@workstation:~/CPE212_Planta$ ansible all -m apt -a name=snapd --become
--ask-become-pass
BECOME password:
```

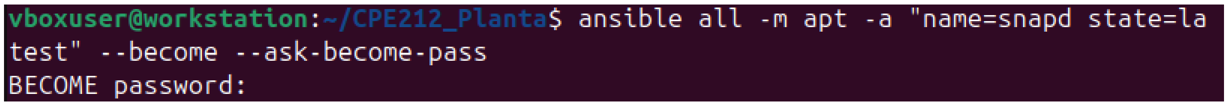


```
192.168.56.101 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756377198,
    "cache_updated": false,
    "changed": false
}
vboxuser@workstation:~/CPE212_Planta$ 
```

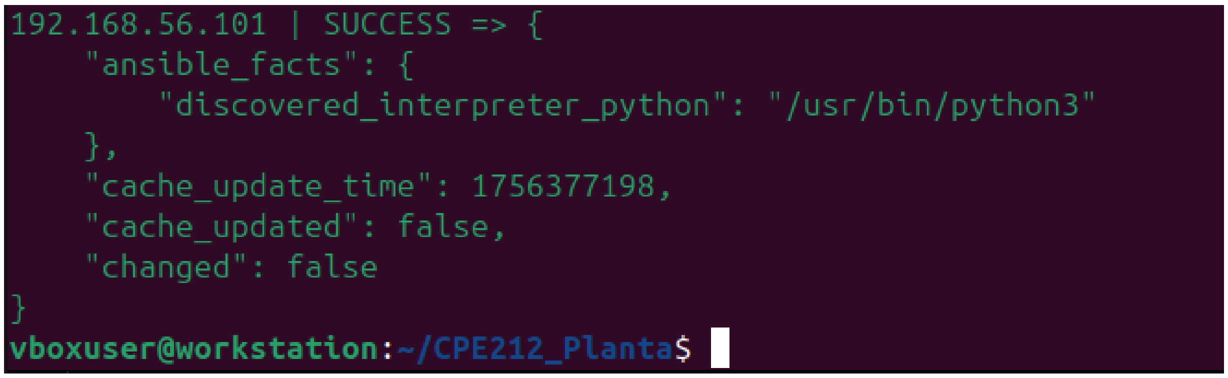
Can you describe the result of this command? Is it a success? Did it change anything in the remote servers?

- The result was SUCCESS, which indicates that the snapd package was successfully installed in the remote server.

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



```
vboxuser@workstation:~/CPE212_Planta$ ansible all -m apt -a "name=snapd state=la
test" --become --ask-become-pass
BECOME password:
```

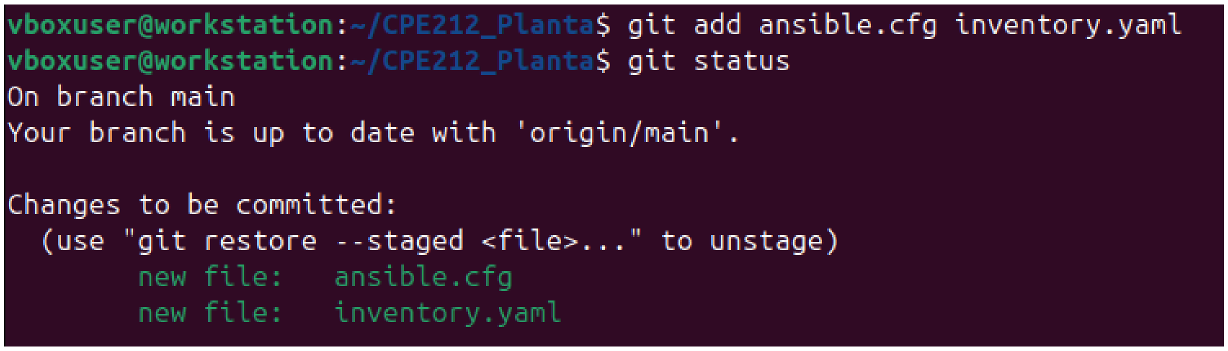


```
192.168.56.101 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1756377198,
    "cache_updated": false,
    "changed": false
}
vboxuser@workstation:~/CPE212_Planta$
```

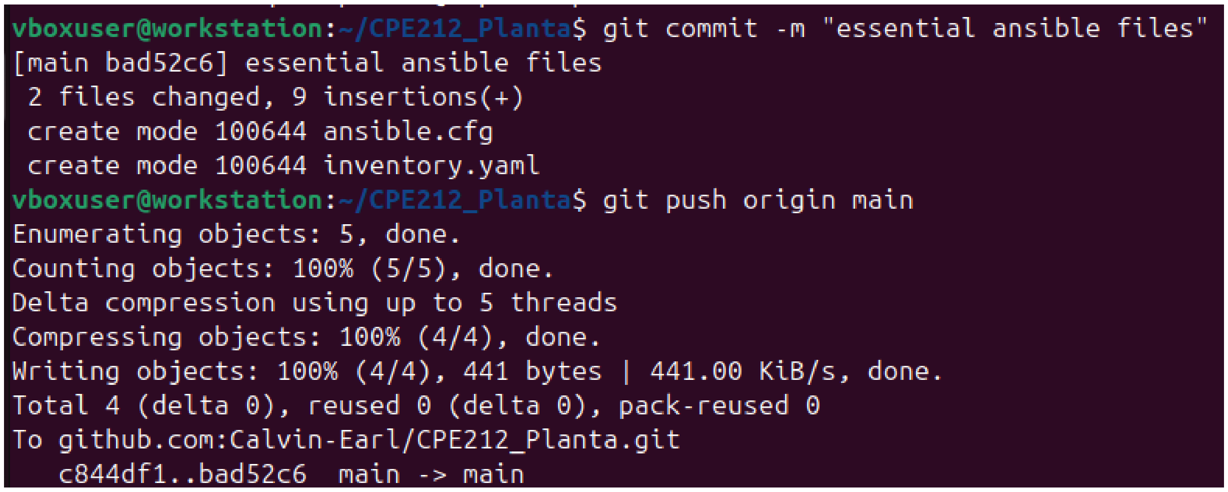
Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

- The output also resulted in a success.

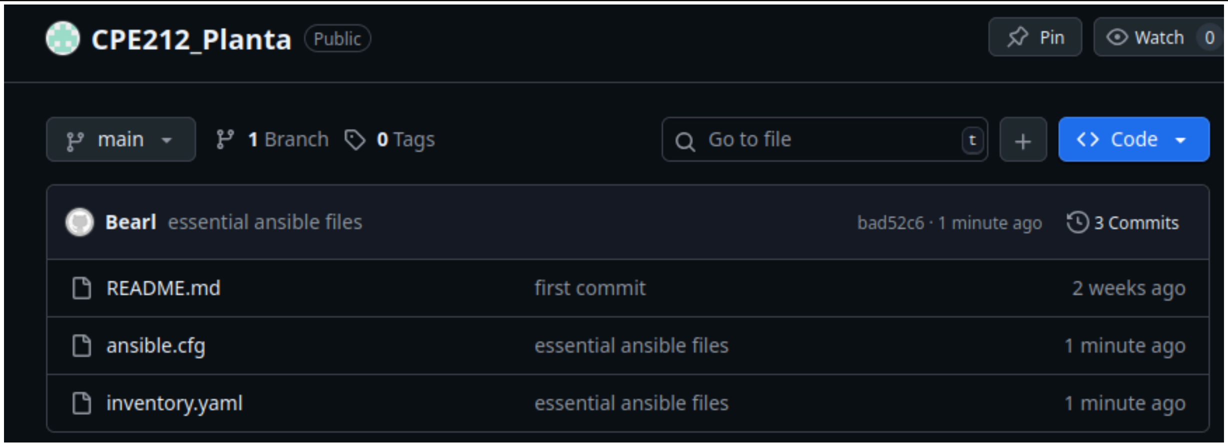
4. At this point, make sure to commit all changes to GitHub.



```
vboxuser@workstation:~/CPE212_Planta$ git add ansible.cfg inventory.yaml
vboxuser@workstation:~/CPE212_Planta$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ansible.cfg
        new file:   inventory.yaml
```



```
vboxuser@workstation:~/CPE212_Planta$ git commit -m "essential ansible files"
[main bad52c6] essential ansible files
 2 files changed, 9 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 inventory.yaml
vboxuser@workstation:~/CPE212_Planta$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 5 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 441 bytes | 441.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Calvin-Earl/CPE212_Planta.git
   c844df1..bad52c6  main -> main
```

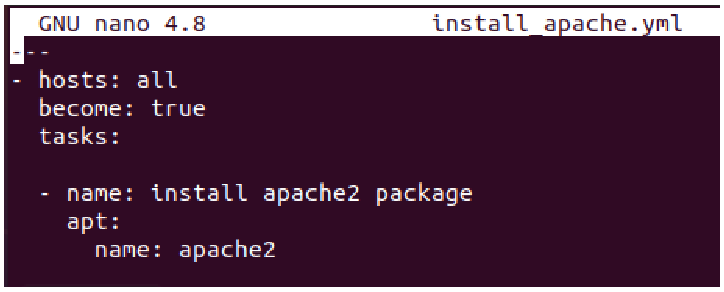


```
Bearl  essential ansible files          bad52c6 · 1 minute ago   3 Commits
README.md        first commit             2 weeks ago
ansible.cfg      essential ansible files  1 minute ago
inventory.yaml   essential ansible files  1 minute ago
```

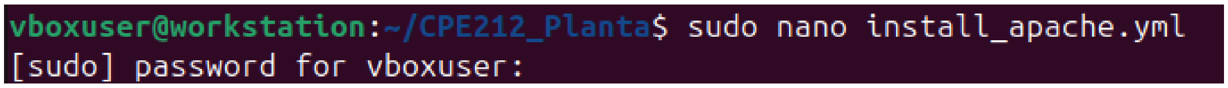
Task 2: Writing our First Playbook

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.

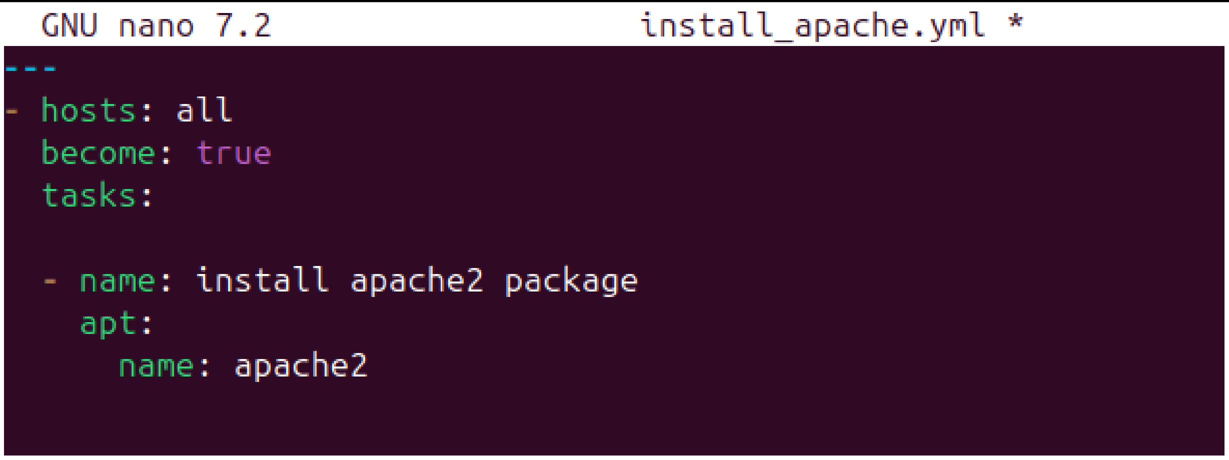
   When the editor appears, type the following:



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

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



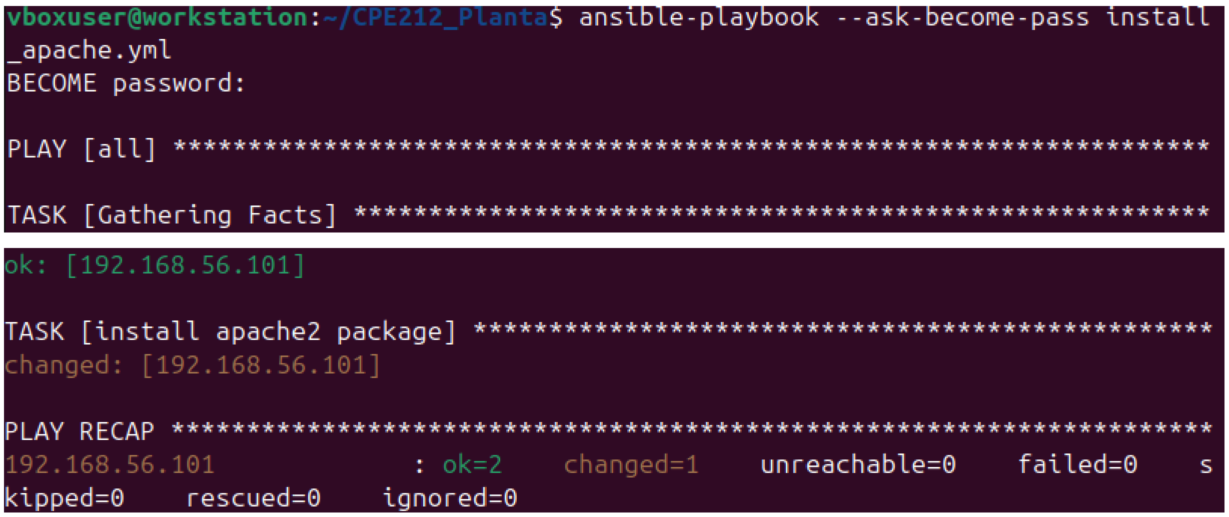
```
vboxuser@workstation:~/CPE212_Planta$ sudo nano install_apache.yml
[sudo] password for vboxuser:
```



```
  GNU nano 7.2                        install_apache.yml *
---
- hosts: all
  become: true
  tasks:

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

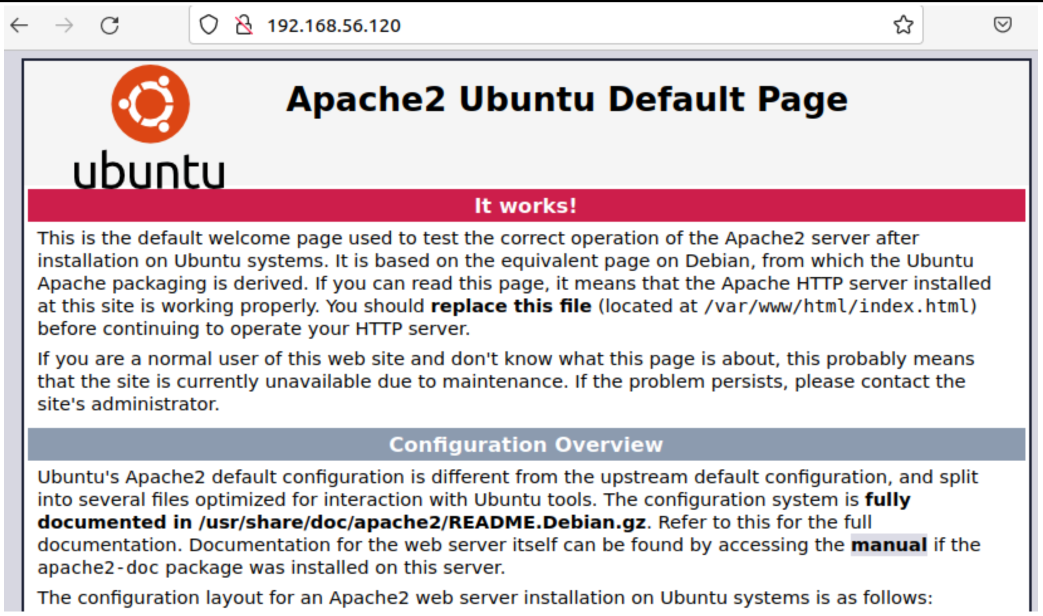
Make sure to save the file. Take note also of the alignments of the texts.

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_Planta$ ansible-playbook --ask-become-pass install
_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.101]

TASK [install apache2 package] *************************************************
changed: [192.168.56.101]

PLAY RECAP *********************************************************************
192.168.56.101             : ok=2    changed=1    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.



192.168.56.120

Apache2 Ubuntu Default Page

It works!

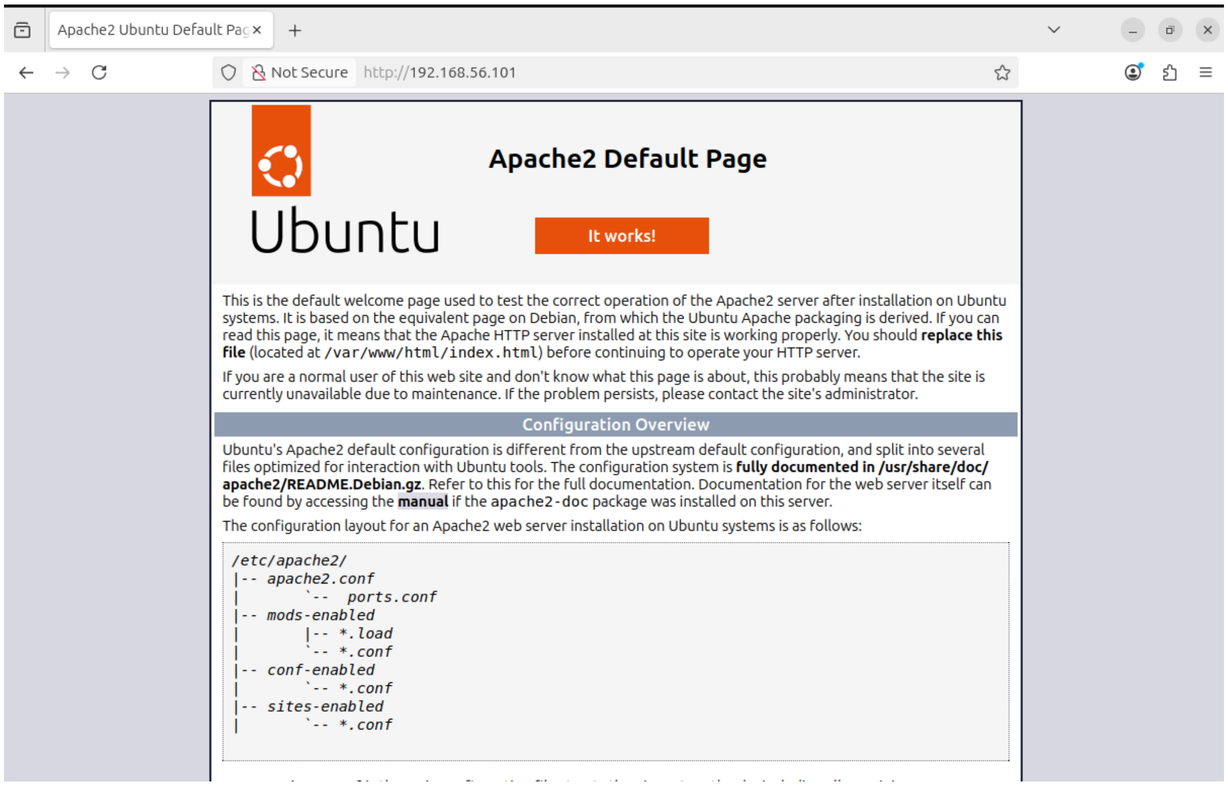
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at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

Not Secure http://192.168.56.101

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.

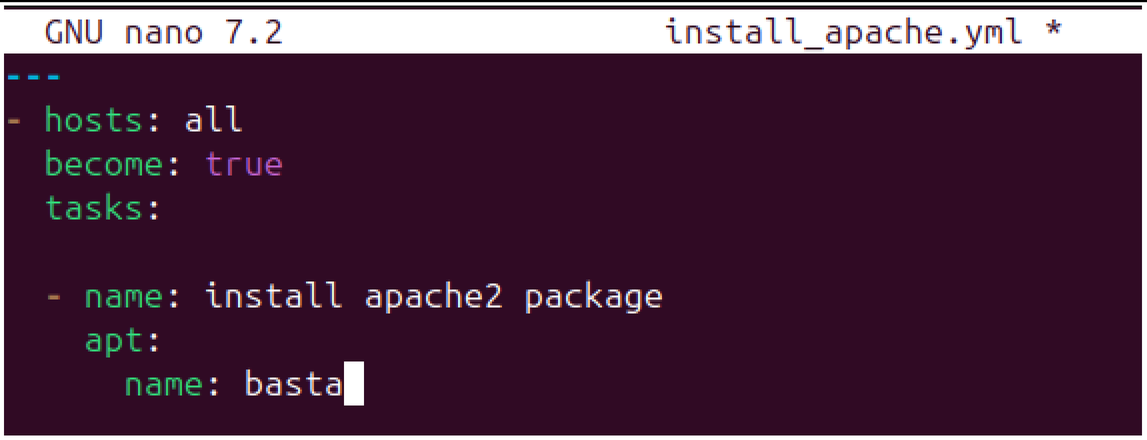
Configuration Overview

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.

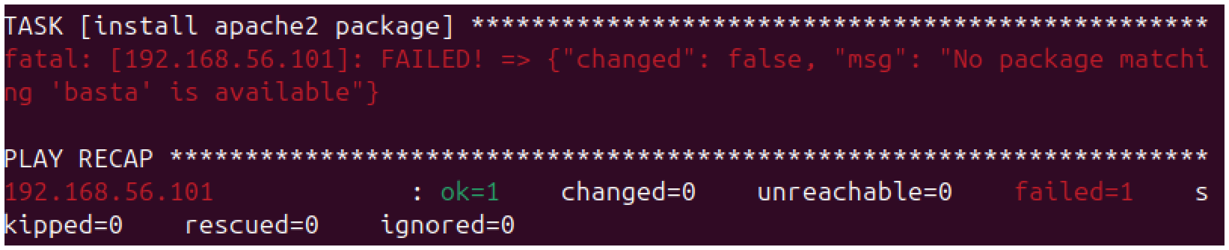
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

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

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?



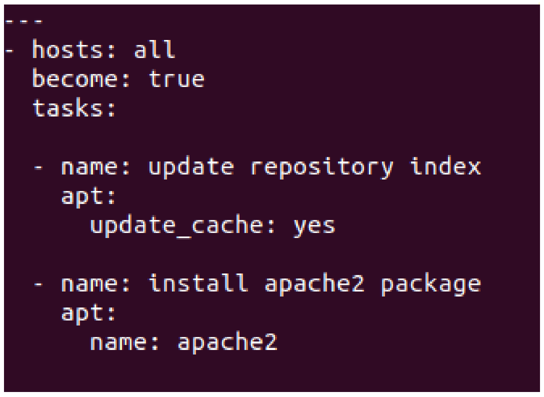
```
  GNU nano 7.2                    install_apache.yml *
---
- hosts: all
  become: true
  tasks:

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



```
TASK [install apache2 package] *************************************************
fatal: [192.168.56.101]: FAILED! => {"changed": false, "msg": "No package matchi
ng 'basta' is available"}

PLAY RECAP *********************************************************************
192.168.56.101             : ok=1    changed=0    unreachable=0    failed=1    s
kipped=0    rescued=0    ignored=0
```

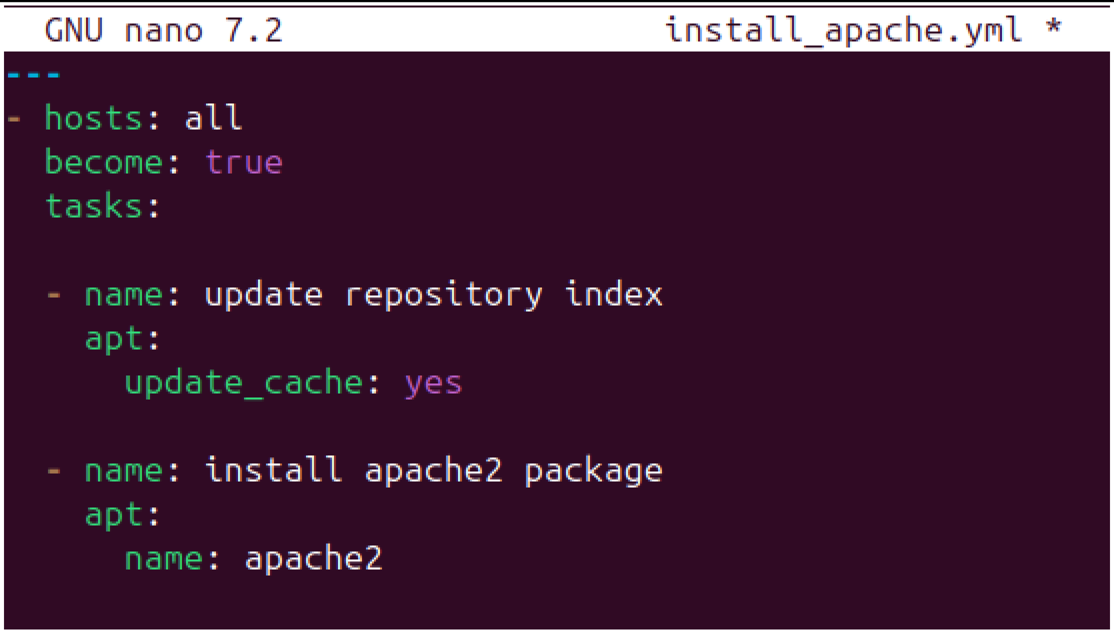
- The output resulted in a failure because the system did not recognize the package to be installed.

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.



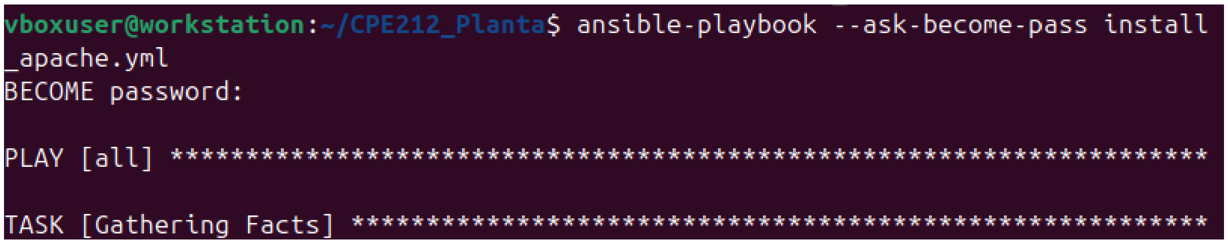
```
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

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

Save the changes to this file and exit.

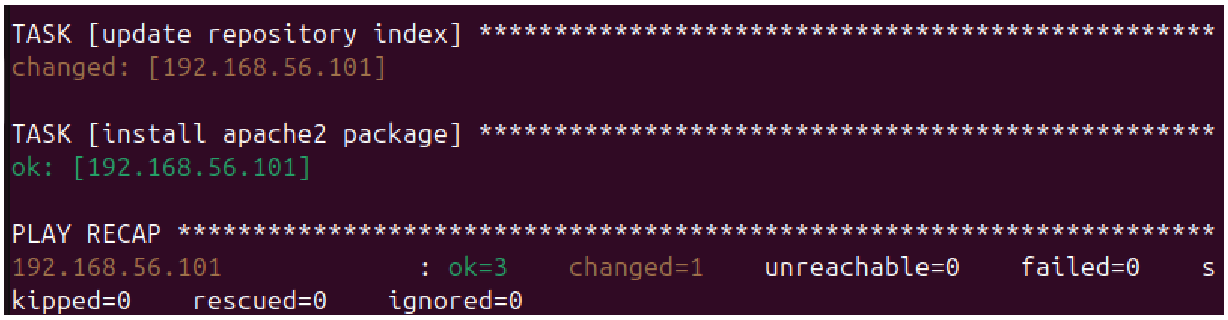


```
  GNU nano 7.2                         install_apache.yml *
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

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

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



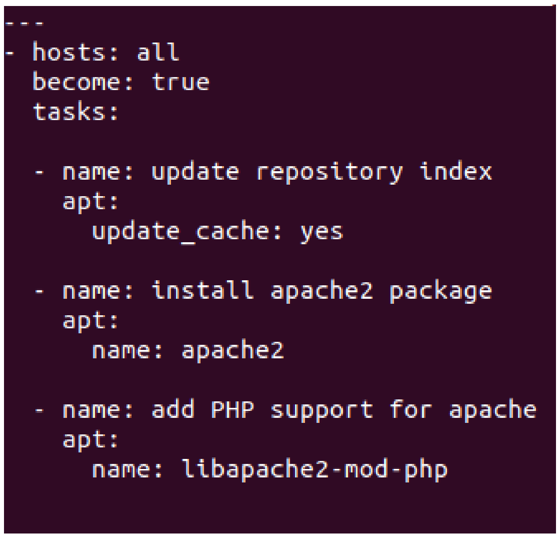
```
vboxuser@workstation:~/CPE212_Planta$ ansible-playbook --ask-become-pass install
_apache.yml
BECOME password:

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

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



```
TASK [update repository index] *************************************************
changed: [192.168.56.101]

TASK [install apache2 package] *************************************************
ok: [192.168.56.101]

PLAY RECAP *********************************************************************
192.168.56.101             : ok=3    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

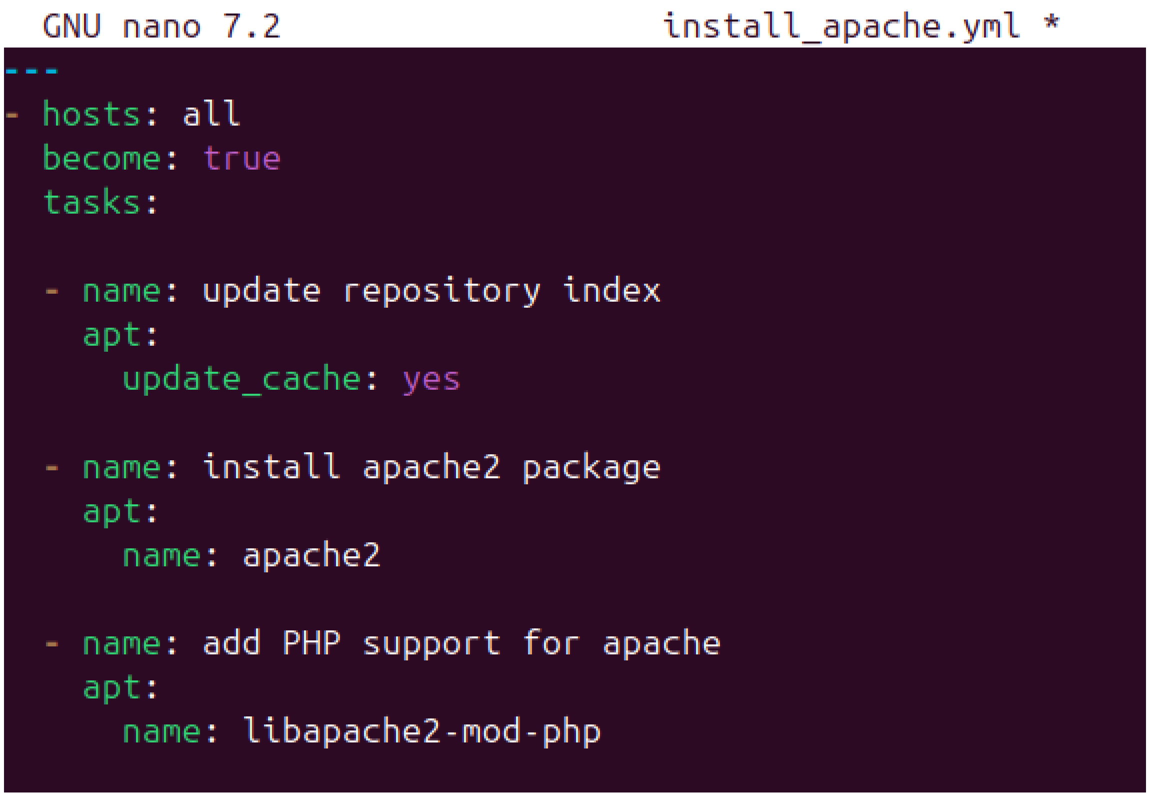
- apt update and installation of the apache2 package was executed on the remote servers.

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



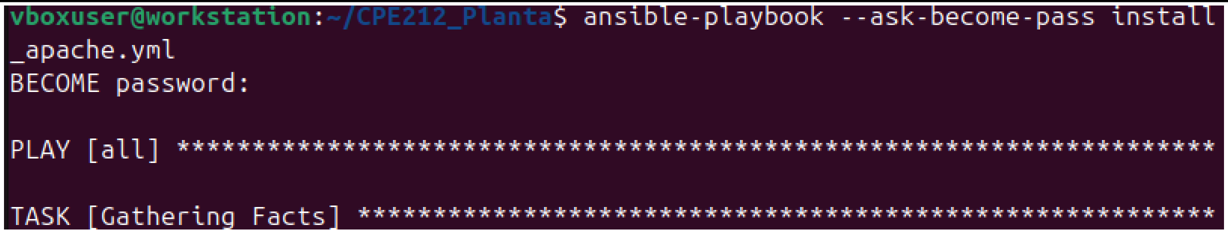
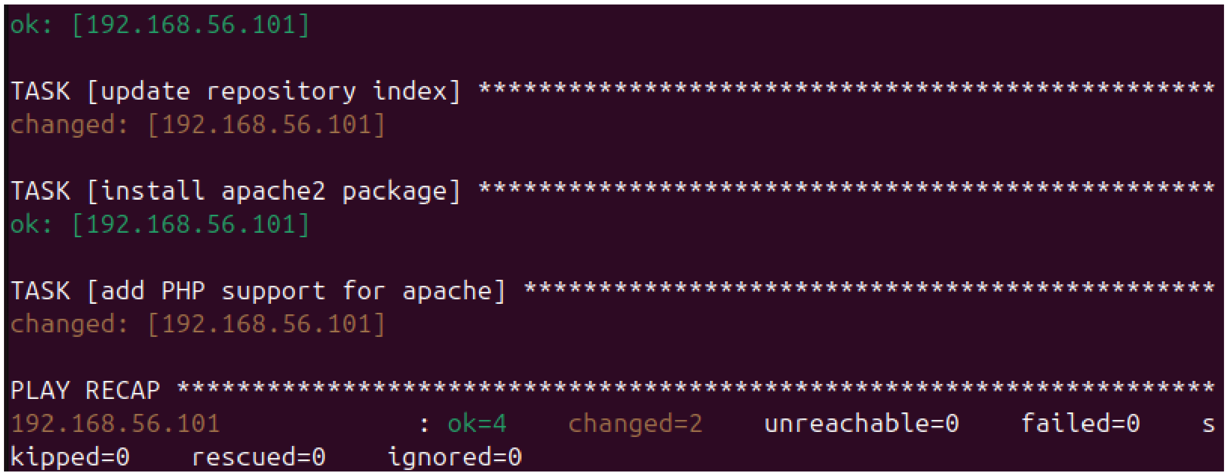
```
---
- hosts: all
  become: true
  tasks:

  - 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.



```
  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
```

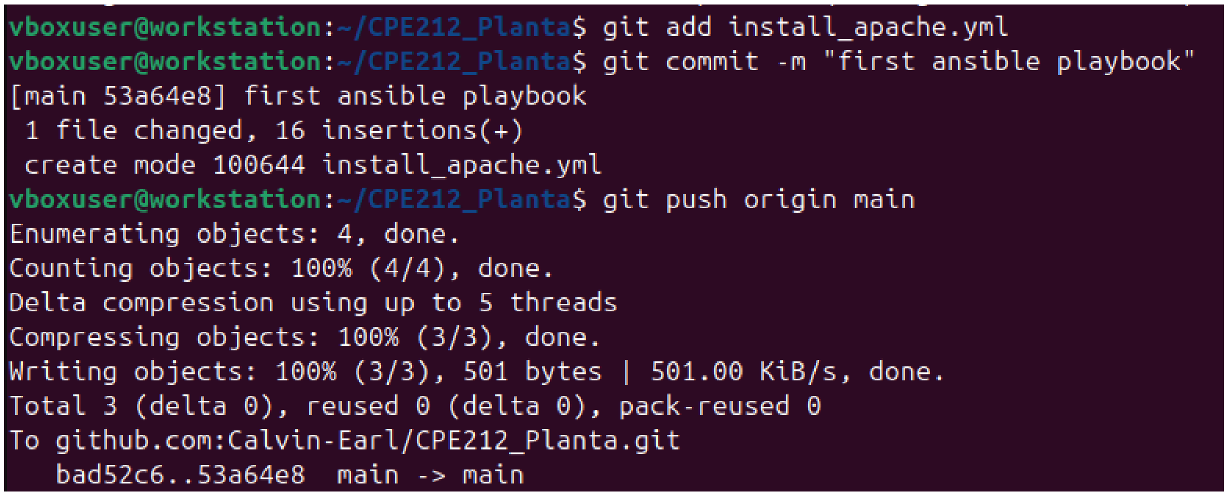
8. Run the playbook and describe the output. Did the new command change anything on the remote servers?

```
vboxuser@workstation:~/CPE212_Planta$ ansible-playbook --ask-become-pass install
_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.101]

TASK [update repository index] *************************************************
changed: [192.168.56.101]

TASK [install apache2 package] *************************************************
ok: [192.168.56.101]

TASK [add PHP support for apache] **********************************************
changed: [192.168.56.101]

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

- The PHP support for apache2 was successfully installed on the remote server.

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



```
vboxuser@workstation:~/CPE212_Planta$ git add install_apache.yml
vboxuser@workstation:~/CPE212_Planta$ git commit -m "first ansible playbook"
[main 53a64e8] first ansible playbook
 1 file changed, 16 insertions(+)
 create mode 100644 install_apache.yml
vboxuser@workstation:~/CPE212_Planta$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 5 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 501 bytes | 501.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Calvin-Earl/CPE212_Planta.git
   bad52c6..53a64e8  main -> main
```

git@github.com:Calvin-Earl/CPE212_Planta.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?
   - Playbook allows us to simplify and automate the tasks needed to execute when managing servers. As seen in this activity, manually typing the commands in

the terminal can be too time-consuming, as it is an inefficient way of managing servers. With playbooks, we can write tasks, and even include specific conditions to be executed along with the task, as well as specify the hosts targeted by the playbook. This helps us manage servers easily, as it makes server management a lot more efficient.

2. Summarize what we have done on this activity.

   - In this activity, I prepared my nodes for ansible automation by including the necessary files, such as ansible.cfg and inventory.yaml. After configuring the ansible file and listing the necessary IP addresses to the inventory file, we first tried installing packages to the remote servers by manually running the commands in the terminal. After that, I tried an easier method for executing tasks directed to the remote servers, which is by writing a playbook. In my playbook, I was able to write multiple tasks and specify which hosts should the playbook be directed to, which demonstrates the advantages of using .yml files for server management and automation. After completing the activity, I committed and pushed all the created files into my github repository.