

Name: Peñas, Issa Victoria H.	Date Performed: 09/06/2022
Course/Section: CPE 232 - CPE31S22	Date Submitted: 09/08/2022
Instructor: Dr. Jonathan Taylar	Semester and SY: 1st Semester (SY: 2022 - 2023)
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 Elevated Ad hoc commands <p>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.</p> <p>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</p>	
Task 1: Run elevated ad hoc commands <ol style="list-style-type: none"> 1. Locally, we use the command <i>sudo apt update</i> when we want to download package information from all configured resources. The sources often defined in <i>/etc/apt/sources.list</i> file and other files located in <i>/etc/apt/sources.list.d/</i> 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: <i>ansible all -m apt -a update_cache=true</i> 	

```

penas@penas-workstation-VirtualBox:~/Penas/ansible$ ansible all -m apt -a update_cache=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)"
}
192.168.56.102 | 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)"
}

```

Figure 1.1. Entering the given command gives an unsuccessful output

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

- It was unsuccessful due to the lack of privilege to run the command.

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.

```

penas@penas-workstation-VirtualBox:~/Penas/ansible$ 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": 1662431572,
  "cache_updated": true,
  "changed": true
}
192.168.56.101 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431573,
  "cache_updated": true,
  "changed": true
}

```

Figure 1.2. Appending a privileged command on the given command in which its output changed

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 actually to install the package.

```
penas@penas-workstation-VirtualBox:~/Penas/ansible$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431572,
  "cache_updated": false,
  "changed": false
}
192.168.56.101 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431573,
  "cache_updated": false,
  "changed": false
}
penas@penas-workstation-VirtualBox:~/Penas/ansible$
```

Figure 1.3. Installing a VIM which is a text editor which outputted a successful installation in green

- 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?

- Yes, It sorted and Searched the vim-nox along with its version mode as the command *which vim* allows to locate the vim's executable files and its destination

```
penas@penas-workstation-VirtualBox:~/Penas/ansible$ which vim
penas@penas-workstation-VirtualBox:~/Penas/ansible$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy 2:8.2.3995-1ubuntu2 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

penas@penas-workstation-VirtualBox:~/Penas/ansible$
```

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`.

- In the `history.log` it shows the previous commands that have been executed specifically commands of the installations of software packages, library and modules.

```
penas@penas-workstation-VirtualBox:~/Penas/ansible$ cd /var/log
penas@penas-workstation-VirtualBox:/var/log$ ls
alternatives.log      cups                  faillog              openvpn
alternatives.log.1    dist-upgrade         fontconfig.log       private
apt                   dmesg                gdm3                 speech-dispatcher
auth.log              dmesg.0              gpu-manager.log      syslog
auth.log.1            dmesg.1.gz           hp                   syslog.1
boot.log              dmesg.2.gz           installer            ubuntu-advantage.log
boot.log.1            dmesg.3.gz           journal              ubuntu-advantage-timer.log
bootstrap.log         dmesg.4.gz           kern.log              ubuntu-advantage-timer.log.1
btmtp                 dpkg.log             kern.log.1            unattended-upgrades
btmtp.1               dpkg.log.1           lastlog              wtmp
penas@penas-workstation-VirtualBox:/var/log$
```

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

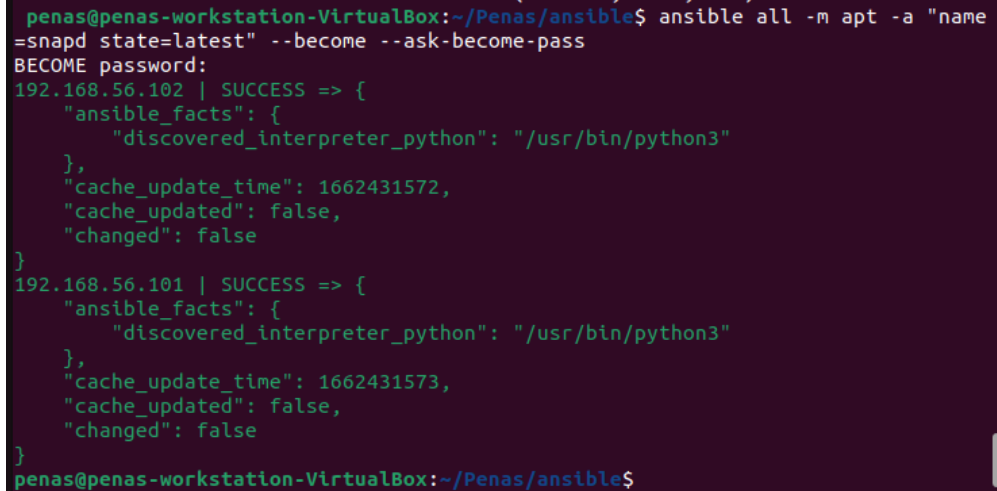
3.1 Issue the command: `ansible all -m apt -a name=snapt --become --ask-become-pass`

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

```
penas@penas-workstation-VirtualBox:~/Penas/ansible$ ansible all -m apt -a name=s
napd --become --ask-become-pass
BECOME password:
192.168.56.101 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431573,
  "cache_updated": false,
  "changed": false
}
192.168.56.102 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431572,
  "cache_updated": false,
  "changed": false
}
penas@penas-workstation-VirtualBox:~/Penas/ansible$
```

Figure 1.5. Installing the latest version of `snapt` which is a success

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



```
penas@penas-workstation-VirtualBox:~/Penas/ansible$ 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": 1662431572,
  "cache_updated": false,
  "changed": false
}
192.168.56.101 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1662431573,
  "cache_updated": false,
  "changed": false
}
penas@penas-workstation-VirtualBox:~/Penas/ansible$
```

Figure 1.6. Entering the given command

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

- Between the two outputs it shows the same installation details, since the snapd was already pre-installed within the linux ubuntu system the first command that only contains *name=snapd* will only install the old version that was pre-installed within the system compared to the command that contains the *state=latest* which will re-install the software package up to date or its latest version.

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.

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.

```

GNU nano 6.2                                install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - name: install apache2 package
    apt:
      name: apache2

```

Figure 2.1. Creating a Playbook which consists of given commands that allows the user to install apache2

2. Run the yml file using the command: *ansible-playbook --ask-become-pass install_apache.yml*.
 - Using Playbook allows the user to manually run the commands that were written on the playbook itself in one go, in which running the command *ansible-playbook --ask-become-pass install_apache.yml* will apply also on the connected servers that are stored within the /etc/hosts which is efficient for the administrator to update and install softwares, and debug issues in multiple devices.

```

penas@penas-workstation-VirtualBox:~/Penas/ansible$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [192.168.56.102]
ok: [192.168.56.101]

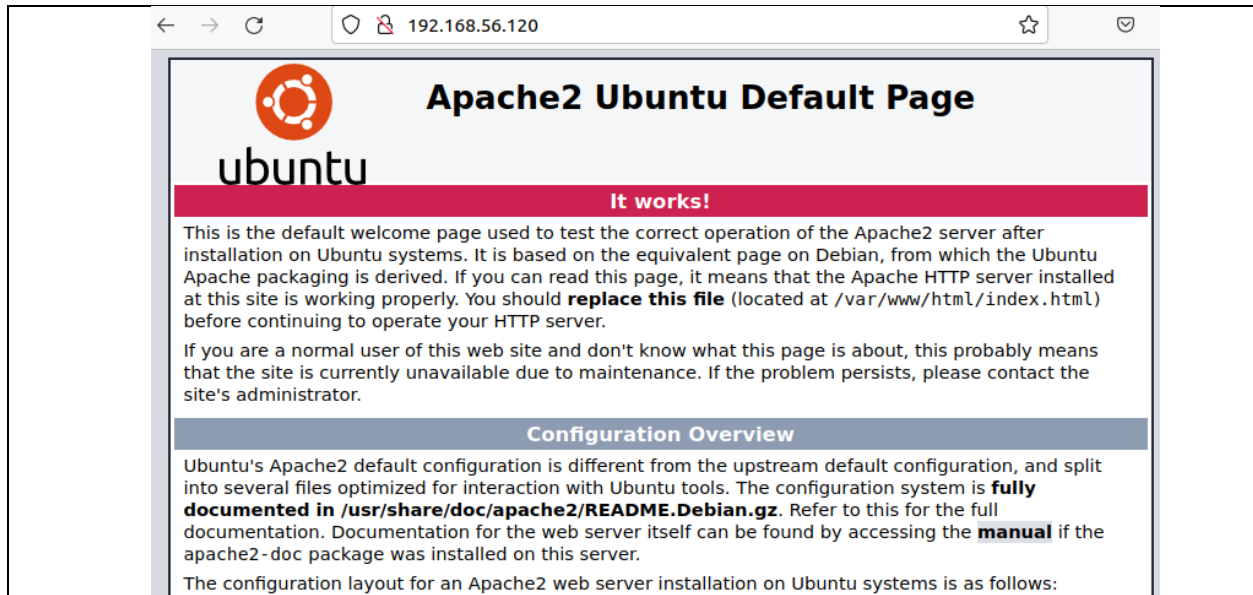
TASK [install apache2 package] *****
changed: [192.168.56.102]
changed: [192.168.56.101]

PLAY RECAP *****
192.168.56.101      : ok=2    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.56.102      : ok=2    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
penas@penas-workstation-VirtualBox:~/Penas/ansible$

```

Figure 2.2. Entering the given command allows to install the apache2 in one go

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.



Server 1:

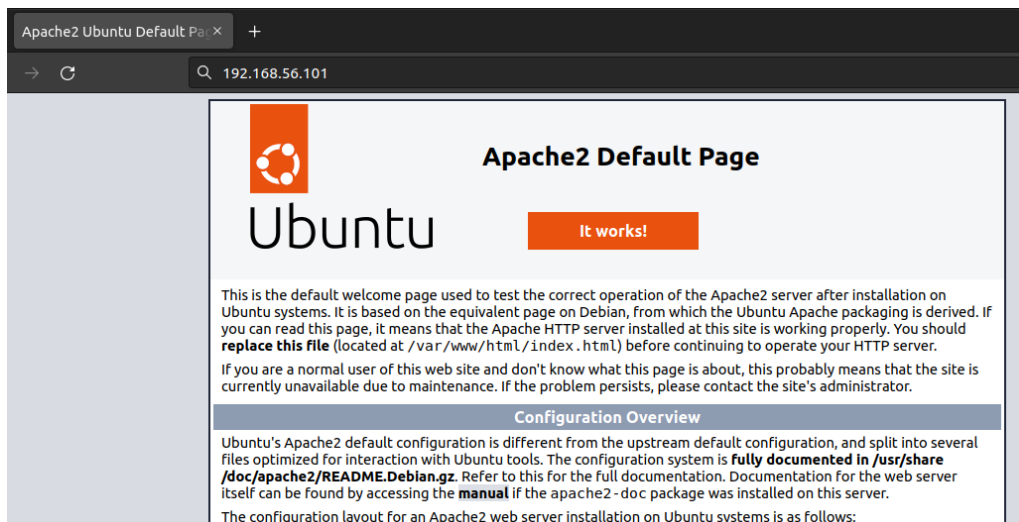


Figure 2.3. Verified the IIP address of 192.168.56.101 (Server 1) that the Apache2 was installed

Server 2:

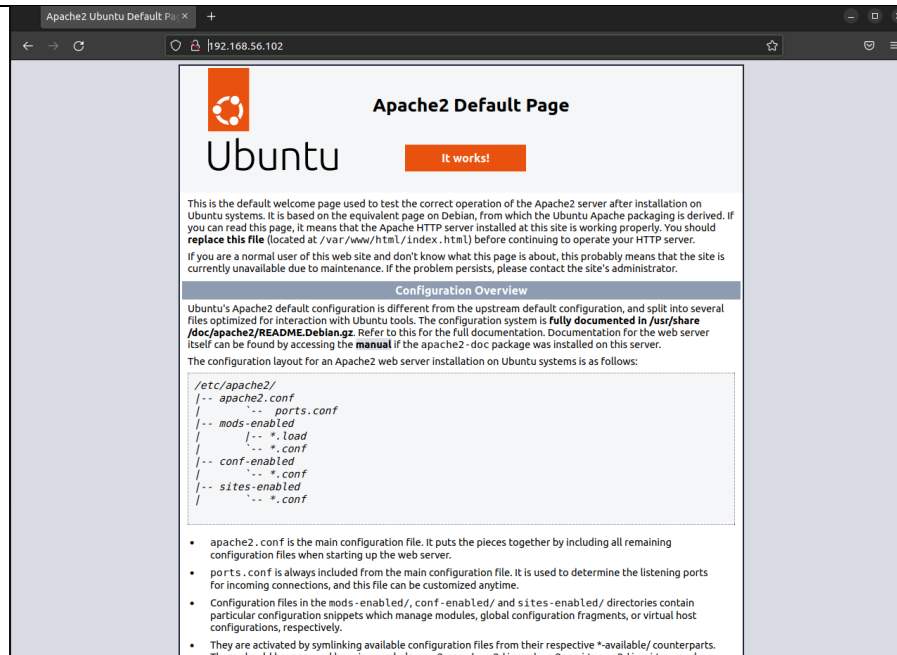


Figure 2.3. Verified the IIP address of 192.168.56.102 (Server 2) that the Apache2 was installed

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?
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.


```

GNU nano 6.2                                install_apache.yml
---
- hosts: all
  become: true
  tasks:

    - name: update repository index
      apt:
        update_cache: yes

    - name: install apache2 package
      apt:
        name: apache2

```

Figure 2.4. Appending the created Playbook which consists of given commands that allows the user to install apache2 and update the repository index

6. Run the playbook and describe the output. Did the new command change anything on the remote servers?
 - Since apache2 was already installed, its output indicated a successful installation faster than updating the appended command which is the repository index that showed the state 'changed' in which the repository was successfully updated.

```

e-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [192.168.56.101]
ok: [192.168.56.102]

TASK [update repository index] *****
changed: [192.168.56.102]
changed: [192.168.56.101]

TASK [install apache2 package] *****
ok: [192.168.56.102]
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
192.168.56.102      : ok=3    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

penas@penas-workstation-VirtualBox: ~/Penas/ansible$

```

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:

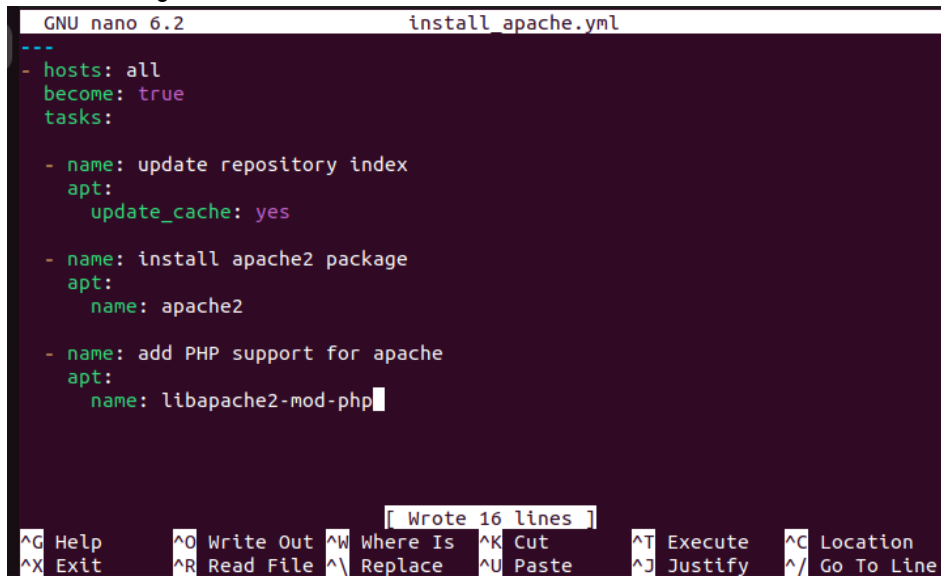
    - name: update repository index
      apt:
        update_cache: yes

    - 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.



```
GNU nano 6.2                                install_apache.yml
--
- hosts: all
  become: true
  tasks:

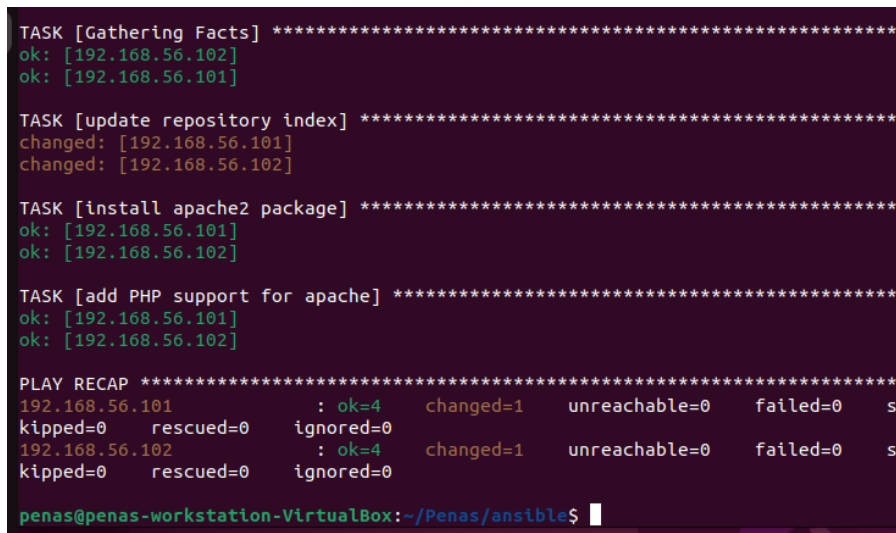
    - name: update repository index
      apt:
        update_cache: yes

    - name: install apache2 package
      apt:
        name: apache2

    - name: add PHP support for apache
      apt:
        name: libapache2-mod-php
```

Figure 2.5. Appending the created Playbook which consists of given commands that allows the user to install apache2, update the repository index, and add PHP support to apache

8. Run the playbook and describe the output. Did the new command change anything on the remote servers?
 - All of the existing commands in the past Playbook will re-run quickly compared to the new appended command that allows the user to add a PHP Support for the apache which is successfully executed within all the servers connected.



```
TASK [Gathering Facts] *****
ok: [192.168.56.102]
ok: [192.168.56.101]

TASK [update repository index] *****
changed: [192.168.56.101]
changed: [192.168.56.102]

TASK [install apache2 package] *****
ok: [192.168.56.101]
ok: [192.168.56.102]

TASK [add PHP support for apache] *****
ok: [192.168.56.101]
ok: [192.168.56.102]

PLAY RECAP *****
192.168.56.101      : ok=4    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.56.102      : ok=4    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

penas@penas-workstation-VirtualBox:~/Penas/ansible$
```

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

```

penas@penas-workstation-VirtualBox:~$ cp -r Penas CPE232_penas
penas@penas-workstation-VirtualBox:~$ cd CPE232_penas
penas@penas-workstation-VirtualBox:~/CPE232_penas$ ls
Penas  README.md
penas@penas-workstation-VirtualBox:~/CPE232_penas$ git add *
penas@penas-workstation-VirtualBox:~/CPE232_penas$ git commit -n
Aborting commit due to empty commit message.
penas@penas-workstation-VirtualBox:~/CPE232_penas$ git commit -m
error: switch `m` requires a value
penas@penas-workstation-VirtualBox:~/CPE232_penas$ git commit -m "Ansible Files"
[main e7dd002] Ansible Files
 3 files changed, 26 insertions(+)
 create mode 100644 Penas/ansible/ansible.cfg
 create mode 100644 Penas/ansible/install_apache.yml
 create mode 100644 Penas/ansible/inventory
penas@penas-workstation-VirtualBox:~/CPE232_penas$ git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 733 bytes | 733.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:IVPENAS/CPE232_penas.git
 254c241..e7dd002  main -> main
penas@penas-workstation-VirtualBox:~/CPE232_penas$

```

Figure 2.6. Syncing and commit the run ansible commands and configurations within the systems into the GitHub

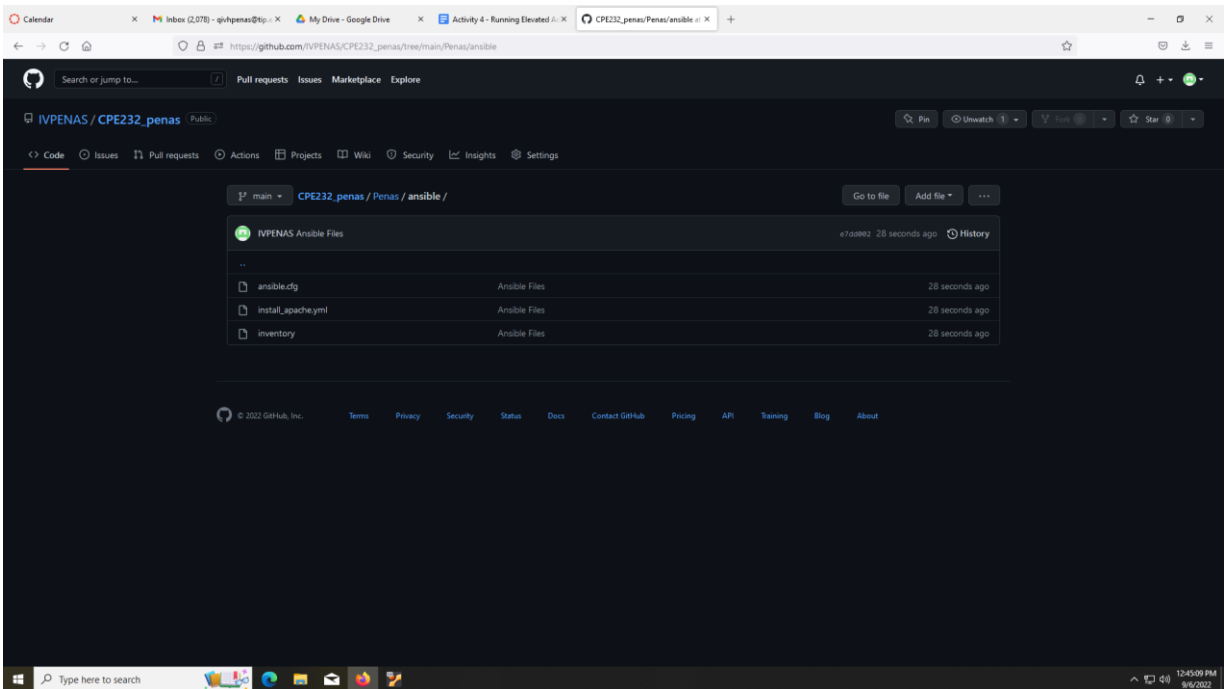


Figure 2.7. All the created files and configurations have been pushed in the GitHub

GitHub Link: https://github.com/IVPENAS/CPE232_penas.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?

- Playbook plays an important role in managing servers in which the Admin was handling multiple devices such as servers and computers in which updating, upgrading, and installing new software will take time if done manually one-by-one on each device. Implementing an Ansible Playbook will help the Administrator to manage time efficiently as it functions as a blueprint for multiple devices by inputting the needed commands in the playbook and executing it in one go will be implemented not only in one device but also to the connected devices or servers.

2. Summarize what we have done on this activity.

- Ansible commands were already introduced in past activities, where in this activity combined **Ad hoc commands** that allow the user to perform single linear commands that can be either run individually depending on its configurations to execute the commands in a more efficient way making the **Ansible Commands** more flexible and useful resulting for the Admin to execute commands in one-go pertaining the **Playbook** where it is a blueprint of commands that can be executed simultaneously within the selected stored IP address of servers or computer inside the Ansible Inventory, one mis-entered detail inside the Inventory will result for the specific computer to negate the Playbook's command and will return a fail output.