

Name: Seruelas, Ronn Kristoper H.	Date Performed: 09-11-2023
Course/Section: CPE232-CPE31S4	Date Submitted: 09-12-2023
Instructor: Dr. Jonathan V. Taylor	Semester and SY: 1st Sem. 2023-2024
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 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:	

```
seruelas@workstation: ~  
seruelas@workstation:~$ sudo apt update  
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease  
Hit:2 http://ph.archive.ubuntu.com/ubuntu jammy InRelease  
Hit:3 http://ph.archive.ubuntu.com/ubuntu jammy-updates InRelease  
Hit:4 http://ph.archive.ubuntu.com/ubuntu jammy-backports InRelease  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Figure 1.1.1 - Updating the workstation.

ansible all -m apt -a update_cache=true

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

```
seruelas@workstation:~/Ansible$ ansible all -m apt -a update_cache=true  
192.168.56.112 | FAILED! => {  
  "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)"  
}  
192.168.56.113 | FAILED! => {  
  "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 1.1.2 - Running the following command, it produced a failed output and was not able to update the ansible.

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.

```

seruelas@workstation:~/AnsibleS4$ ansible all -m apt -a update_cache=true --become
me --ask-become-pass
BECOME password:
192.168.56.112 | CHANGED => {
    "cache_update_time": 1694406987,
    "cache_updated": true,
    "changed": true
}
192.168.56.113 | CHANGED => {
    "cache_update_time": 1694406988,
    "cache_updated": true,
    "changed": true
}

```

Figure 1.1.3 - By including the new arguments, it allowed the user to update the ansible's, as it elevated the permission of the user.

- 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.

```

seruelas@workstation:~/AnsibleS4$ ansible all -m apt -a name=vim-nox --become --
ask-become-pass
BECOME password:
192.168.56.113 | CHANGED => {
    "cache_update_time": 1694406988,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading st
ate information...\nThe following additional packages will be installed:\n font
s-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n ruby-n
et-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n rubygems-integration
vim-common vim-runtime vim-tiny\nSuggested packages:\n apache2 | lighttpd | ht
tpd rl ruby-dev bundler cscope vim-doc indent\nThe following NEW packages will b
e installed:\n fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0
rake ruby\n ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n
rubygems-integration vim-nox vim-runtime\nThe following packages will be upgrade

```

Figure 1.2.1 - Installation of the application, VIM to the ansible

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?

```
seruelas@server1:~$ which vim
/usr/bin/vim
seruelas@server1:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.11 amd64 [installed]
]
Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.11 amd64 [install
d,automatic]
Vi IMproved - enhanced vi editor - compact version
```

```
seruelas@server2:~$ which vim
/usr/bin/vim
seruelas@server2:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.11 amd64 [installed]
]
Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.11 amd64 [install
d,automatic]
Vi IMproved - enhanced vi editor - compact version
```

Figure 1.2.2-1.2.3 - By issuing the following commands to server1 and server 2, we were able to verify the successful installation to both servers from 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**.

```
seruelas@workstation:~/Ansible$ cd /var/log
seruelas@workstation:/var/log$ ls
alternatives.log  btmptmp  faillog  openvpn
alternatives.log.1  btmptmp.1  fontconfig.log  private
apt  cups  gdm3  speech-dispatcher
auth.log  dist-upgrade  gpu-manager.log  syslog
auth.log.1  dmesg  hp  syslog.1
auth.log.2.gz  dmesg.0  installer  syslog.2.gz
auth.log.3.gz  dmesg.1.gz  journal  syslog.3.gz
boot.log  dmesg.2.gz  kern.log  ubuntu-advantage.log
boot.log.1  dmesg.3.gz  kern.log.1  ubuntu-advantage.log.1
boot.log.2  dmesg.4.gz  kern.log.2.gz  unattended-upgrades
boot.log.3  dpkg.log  kern.log.3.gz  wtmp
bootstrap.log  dpkg.log.1  lastlog
seruelas@workstation:/var/log$ cd apt
seruelas@workstation:/var/log/apt$ ls
eipp.log.xz  history.log  history.log.1.gz  term.log  term.log.1.gz
```

Figure 1.2.4 - Changing directory and verifying existence of **history.log**

```

seruelas@workstation:/var/log/apt$ cat history.log

Start-Date: 2023-09-11 11:01:43
Commandline: apt install pip
Requested-By: seruelas (1000)
Install: libalgorithm-merge-perl:amd64 (0.08-3, automatic), manpages-dev:amd64 (
5.10-1ubuntu1, automatic), g++-11:amd64 (11.4.0-1ubuntu1~22.04, automatic), gcc-
11:amd64 (11.4.0-1ubuntu1~22.04, automatic), build-essential:amd64 (12.9ubuntu3,
automatic), libc6:amd64 (2.38-4ubuntu2.3, automatic), libstdc++6:amd64 (1
1.4.0-1ubuntu1~22.04, automatic), libpython3-dev:amd64 (3.10.6-1~22.04, automati
c), g++:amd64 (4:11.2.0-1ubuntu1, automatic), gcc:amd64 (4:11.2.0-1ubuntu1, auto
matic), libalgorithm-diff-perl:amd64 (1.201-1, automatic), libbinutils:amd64 (2.
38-4ubuntu2.3, automatic), libfakeroot:amd64 (1.28-1ubuntu1, automatic), binutil
s-x86-64-linux-gnu:amd64 (2.38-4ubuntu2.3, automatic), libcc1-0:amd64 (12.3.0-1u
buntu1~22.04, automatic), zlib1g-dev:amd64 (1:1.2.11.dfsg-2ubuntu9.2, automatic)
, python3-wheel:amd64 (0.37.1-2ubuntu0.22.04.1, automatic), libfile-fcntllock-pe
rl:amd64 (0.22-3build7, automatic), dpkg-dev:amd64 (1.21.1ubuntu2.2, automatic),
libasan6:amd64 (11.4.0-1ubuntu1~22.04, automatic), libnsl-dev:amd64 (1.3.0-2bui
ld2, automatic), rpcsvc-proto:amd64 (1.4.2-0ubuntu6, automatic), make:amd64 (4.3
-4.1build1, automatic), libc6-dev:amd64 (2.38-4ubuntu2.3, automatic), lto-disabled
-list:amd64 (24, automatic), python3-dev:amd64 (3.10.6-1~22.04, automatic), libc
rypt-dev:amd64 (1:4.4.27-1, automatic), libpython3.10-dev:amd64 (3.10.12-1~22.04
.2, automatic), binutils-common:amd64 (2.38-4ubuntu2.3, automatic), libitm1:amd6
4 (12.3.0-1ubuntu1~22.04, automatic), python3-pip:amd64 (22.0.2+dfsg-1ubuntu0.3)

```

Figure 1.2.5 - Inside of the history.log, can be seen the installations and the commands executed by the user.

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?

```

seruelas@workstation:~/Ansible$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
192.168.56.112 | SUCCESS => {
  "cache_update_time": 1694406987,
  "cache_updated": false,
  "changed": false
}
192.168.56.113 | SUCCESS => {
  "cache_update_time": 1694406988,
  "cache_updated": false,
  "changed": false
}

```

Figure 1.3.1 - By executing the installation command for snapd, we were able to successfully install the snapd package by using ansible.

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.

```

seruelas@workstation:~/AnsibleS4$ ansible all -m apt -a "name=snapd state=latest"
" --become --ask-become-pass
BECOME password:
192.168.56.112 | SUCCESS => {
  "cache_update_time": 1694406987,
  "cache_updated": false,
  "changed": false
}
192.168.56.113 | SUCCESS => {
  "cache_update_time": 1694406988,
  "cache_updated": false,
  "changed": false
}

```

Figure 1.3.2 - By issuing the command with additional arguments, we were able to install the snapd package in its latest version.

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

```

seruelas@workstation: ~/AnsibleS4
seruelas@workstation:~/AnsibleS4$ git add *
seruelas@workstation:~/AnsibleS4$ git commit -m "Latest Commit"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
seruelas@workstation:~/AnsibleS4$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
seruelas@workstation:~/AnsibleS4$ git push origin
Everything up-to-date

```

Figure 1.3.3 - Committing 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:

```
seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2                                install_apache.yml
--
- hosts: all
  become: true
  tasks:
    - name: install apache2 package
      apt:
        name: apache2

seruelas@workstation:~/CPE232_Seruelas1$ ls
ansible.cfg  inventory  README.md
```

Figures 2.1.1-2.1.2 - Creating the playbook for installation of apache2.

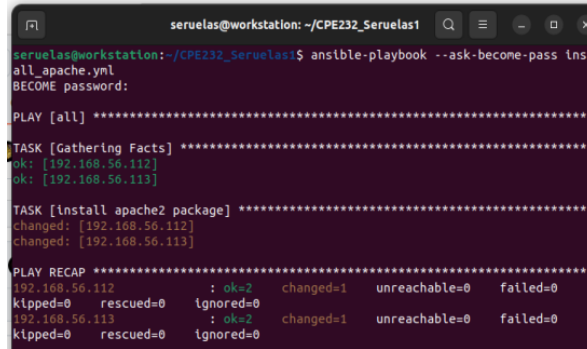
```
seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2                                ansible.cfg *
[defaults]
inventory = inventory
host_key_checking = False
deprecation_warning= False
remote_user = seruelas
private_key_file = ~/.ssh/

seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2                                inventory
[servers]
192.168.56.112 ansible_python_interpreter=/usr/bin/python3
192.168.56.113 ansible_python_interpreter=/usr/bin/python3
```

Figures 2.1.3-2.1.4 - Setting up the repository for the ansible.

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.



```
seruelas@workstation: ~/CPE232_Seruelas1
seruelas@workstation:~/CPE232_Seruelas1$ ansible-playbook --ask-become-pass inst
all_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [192.168.56.112]
ok: [192.168.56.113]

TASK [install apache2 package] *****
changed: [192.168.56.112]
changed: [192.168.56.113]

PLAY RECAP *****
192.168.56.112 : ok=2 changed=1 unreachable=0 failed=0 s
klpped=0 rescued=0 ignored=0
192.168.56.113 : ok=2 changed=1 unreachable=0 failed=0 s
klpped=0 rescued=0 ignored=0
```

Figure 2.2.1 - By executing the command, it was able to execute the playbook, in which it produced different tasks, where first it checks for the hosts for the ansible, and installing the apache2 package in each hosts.

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.

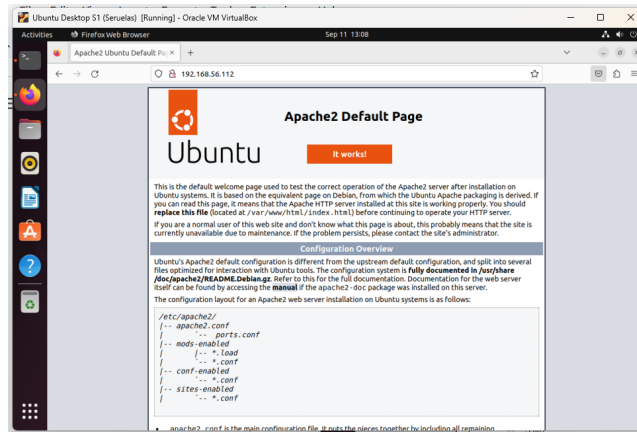


Figure 2.3.1 - Verification of installation of apache2 in server1.

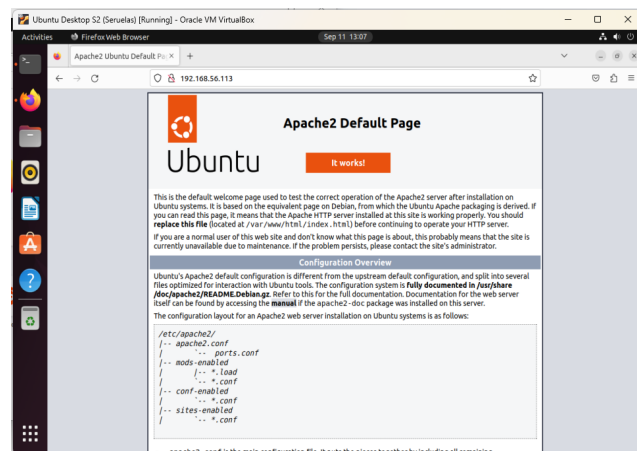
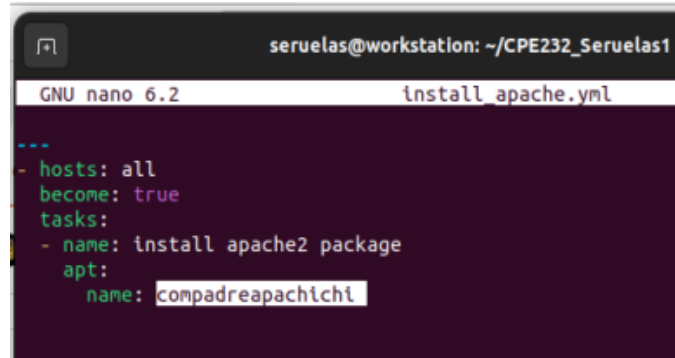


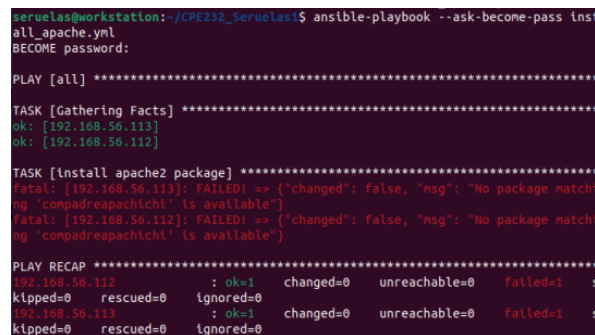
Figure 2.3.2 - Verification of installation of apache2 in server1.

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?



```
seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2 install_apache.yml
---
- hosts: all
  become: true
  tasks:
  - name: install apache2 package
    apt:
      name: compadreapachichi
```

Figure 2.4.1 - Changing the name of the package to another name.



```
seruelas@workstation:~/CPE232_Seruelas1$ ansible-playbook --ask-become-pass inst
all_apache.yml
BECOME password:

PLAY [all] *****

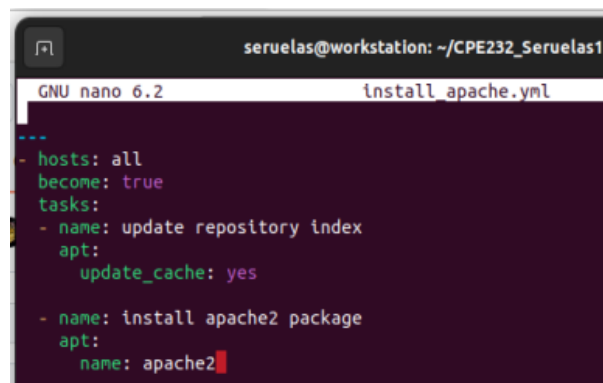
TASK [Gathering Facts] *****
ok: [192.168.56.113]
ok: [192.168.56.112]

TASK [install apache2 package] *****
fatal: [192.168.56.112]: FAILED! => [{"changed": false, "msg": "No package matchi
ng 'compadreapachichi' is available"}]
fatal: [192.168.56.113]: FAILED! => [{"changed": false, "msg": "No package matchi
ng 'compadreapachichi' is available"}]

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

Figure 2.4.1 - Executing the playbook with a new package name, resulting to an error of installation.

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.

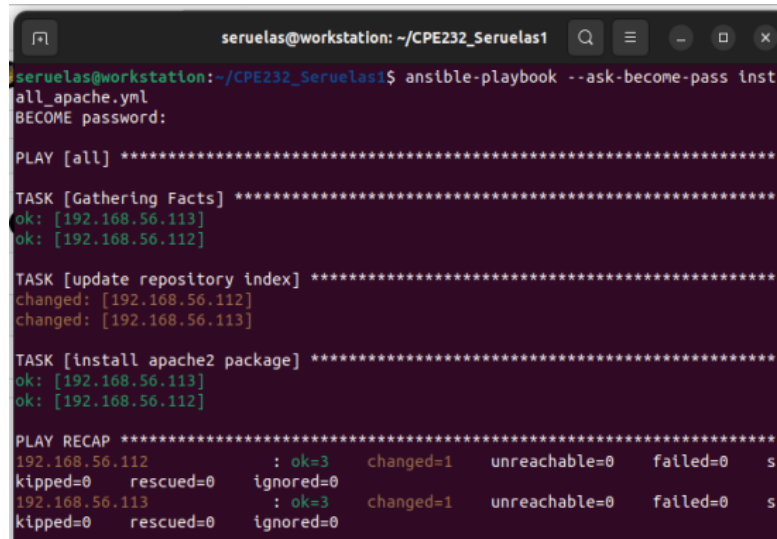


```
seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2 install_apache.yml
---
- hosts: all
  become: true
  tasks:
  - name: update repository index
    apt:
      update_cache: yes
  - name: install apache2 package
    apt:
      name: apache2
```

Figure 2.5.1 - Including the update_cache index in the playbook.

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?



```
seruelas@workstation: ~/CPE232_Seruelas1
seruelas@workstation:~/CPE232_Seruelas1$ ansible-playbook --ask-become-pass inst
all_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [192.168.56.113]
ok: [192.168.56.112]

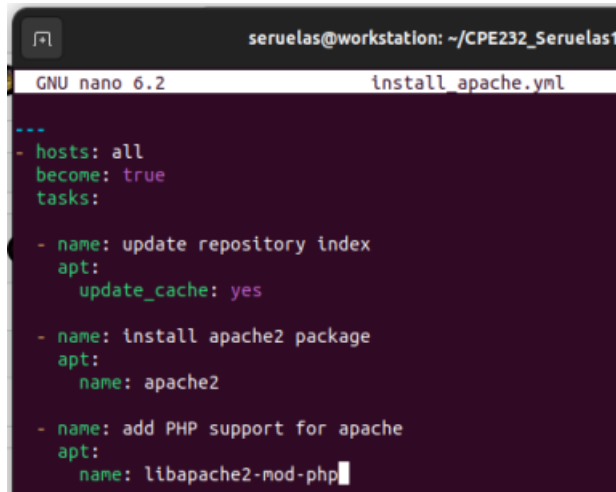
TASK [update repository index] *****
changed: [192.168.56.112]
changed: [192.168.56.113]

TASK [install apache2 package] *****
ok: [192.168.56.113]
ok: [192.168.56.112]

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

Figure 2.6.1 - By executing or running the modified playbook, a new task has been executed for the hosts, in which is to update the repository index of the git.

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



```
seruelas@workstation: ~/CPE232_Seruelas1
GNU nano 6.2      install_apache.yml

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

  - name: update repository index
    apt:
      update_cache: yes

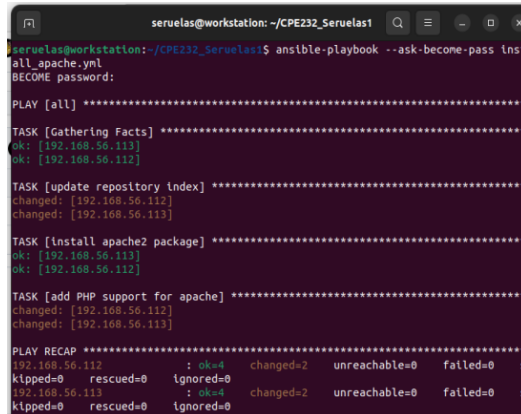
  - name: install apache2 package
    apt:
      name: apache2

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

Figure 2.7.1 - Including the PHP package installation in the playbook.

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?



```
seruelas@workstation: ~/CPE232_Seruelas1
seruelas@workstation:~/CPE232_Seruelas1$ ansible-playbook --ask-become-pass inst
all_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [192.168.56.113]
ok: [192.168.56.112]

TASK [update repository index] *****
changed: [192.168.56.112]
changed: [192.168.56.113]

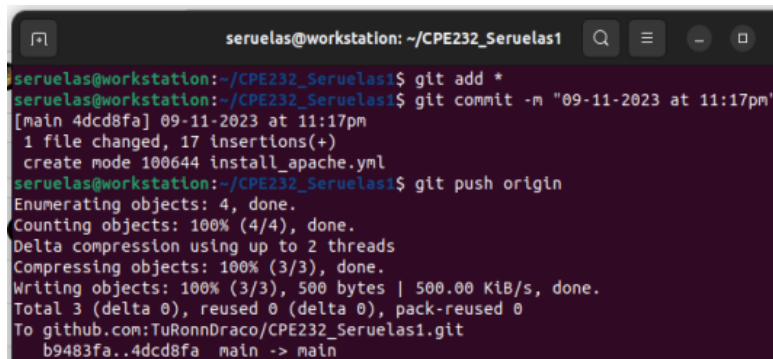
TASK [install apache2 package] *****
ok: [192.168.56.113]
ok: [192.168.56.112]

TASK [add PHP support for apache] *****
changed: [192.168.56.112]
changed: [192.168.56.113]

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

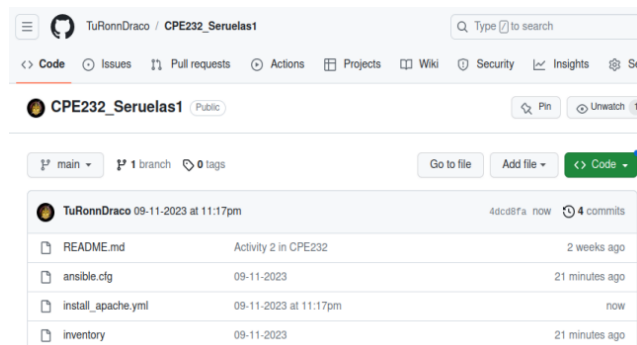
Figure 2.8.1 - Executing the modified playbook, a new task has been executed, in which the playbook successfully installed the PHP package via ansible.

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



```
seruelas@workstation: ~/CPE232_Seruelas1
seruelas@workstation:~/CPE232_Seruelas1$ git add *
seruelas@workstation:~/CPE232_Seruelas1$ git commit -m "09-11-2023 at 11:17pm"
[main 4dcd8fa] 09-11-2023 at 11:17pm
1 file changed, 17 insertions(+)
create mode 100644 install_apache.yml
seruelas@workstation:~/CPE232_Seruelas1$ git push origin
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 500 bytes | 500.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:TuRonnDraco/CPE232_Seruelas1.git
b9483fa..4dcd8fa main -> main
```

Figure 2.9.1 - Committing all changes to the github repository.



https://github.com/TuRonnDraco/CPE232_Seruelas1.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?
 - The importance of a playbook in managing enterprise servers is that it allows the users to create a list of commands that allows the user to execute them efficiently over different devices or hosts, with the use of ansible.
2. Summarize what we have done on this activity.
 - In this activity, we are able to set up our repositories for ansible, in which we are able to execute and create playbooks, that allowed us to execute a list of commands by using a workstation to remotely execute it to other connected hosts. We also used github as the platform for ansible, in which we are able to connect to other hosts, allowing us to perform numerous tasks such as pinging other hosts, and installing packages with them.

Conclusion:

In conclusion, we were able to learn on the basics of ansible, by running through its configurations and preparations on how a user can use it. We ran through its preparations by installing the required packages for our workstations and servers, such as python3, pip, and ansible, and updating our softwares or kernels to the latest versions. We also used a github repository as our platform as it allows us to connect our servers to the workstation. Finally, we were able to learn about playbooks, in which we are educated that playbooks are essentially used as a more efficient script where we can use the playbook as to execute multiple commands to another host with only using the workstation or remote host.

“I affirm that I have not received or given any unauthorized help on this activity, and that all work is my own.”

