Name: Bueno, Joshua C.	Date Performed: 09/05/25
Course/Section: CPE31S2	Date Submitted: 09/12/25
Instructor: Engr. Robin Valenzuela	Semester and SY: 2025-2026

Activity 4: Running Elevated Ad hoc Commands

1. **Objectives:**

1.1 Use commands that makes changes to remote machines

1.2 Use playbook in automating ansible commands

2. **Discussion:**

Provide screenshots for each task.

Elevated Ad hoc commands

So far, we have not performed ansible commands that makes changes to the remote servers. We manage to gather facts and connect to the remote machines, but we still did not make changes on those machines. In this activity, we will learn to use commands that would install, update, and upgrade packages in the remote machines. We will also create a playbook that will be used for automations.

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

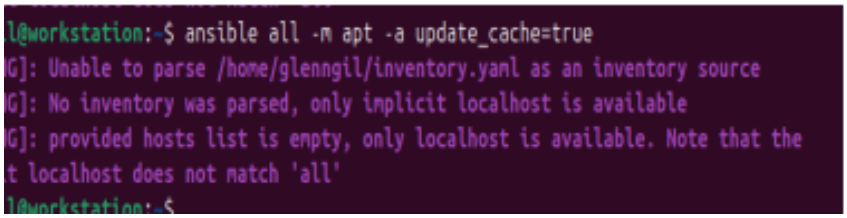
Task 1: Run elevated ad hoc commands

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

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

ansible all -m apt -a update_cache=true

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



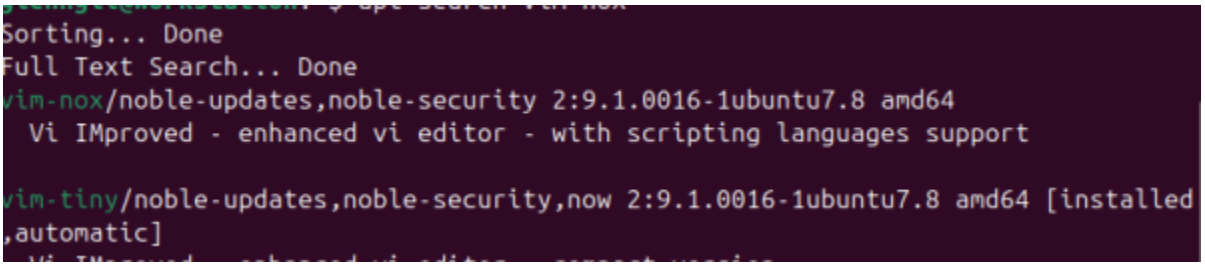
It is not successful , meaning the remote server could not execute the command without proper privileges.

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.

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.

   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?



   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.

```
alternatives.log        dmesg              openvpn
alternatives.log.1      dmesg.0            private
apport.log              dmesg.1.gz         README
apt                     dmesg.2.gz         speech-dispatcher
auth.log                dmesg.3.gz         sssd
auth.log.1              dmesg.4.gz         syslog
auth.log.2.gz           dpkg.log           syslog.1
auth.log.3.gz           dpkg.log.1         syslog.2.gz
boot.log                faillog            syslog.3.gz
boot.log.1              fontconfig.log     sysstat
boot.log.2              gdm3               ubuntu-advantage.log
boot.log.3              gpu-manager.log    ubuntu-advantage.log.1
bootstrap.log           hp                 ufw.log
btmp                    installer          ufw.log.1
btmp.1                  journal            ufw.log.2.gz
```

```
eipp.log.xz  history.log  history.log.1.gz  term.log  term.log.1.gz
```

there are five log files: eipp.log.xz, history.log, history.log.1.gz, term.log, and term.log.1.gz.

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?

```
ass
BECOME password:
192.168.56.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757057418,
    "cache_updated": false,
    "changed": false
}
192.168.56.103 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1757053606,
    "cache_updated": false,
    "changed": false
}
```

   The result of the command successfully pinged the ip address of the two remote servers.

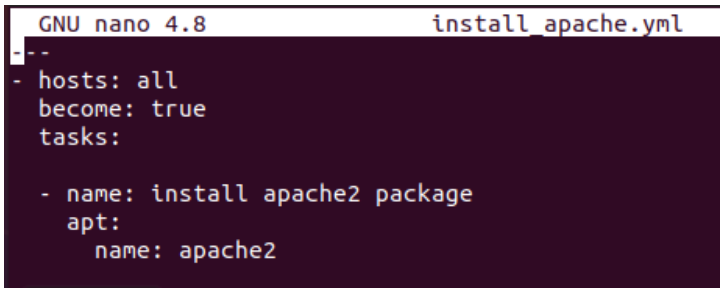
3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*
Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

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

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:



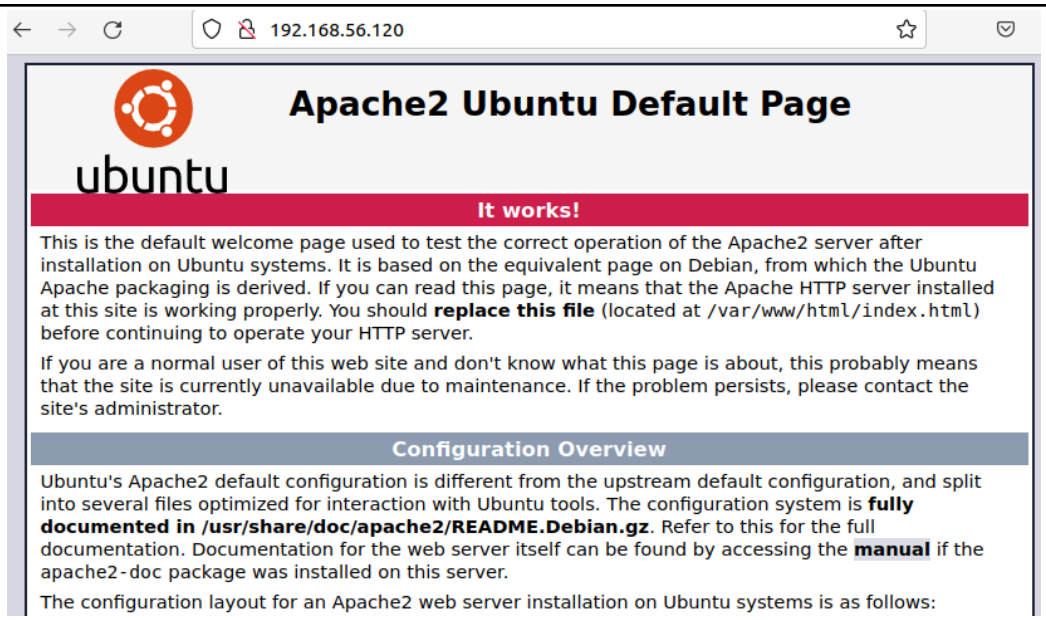
Make sure to save the file. Take note also of the alignments of the texts.

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

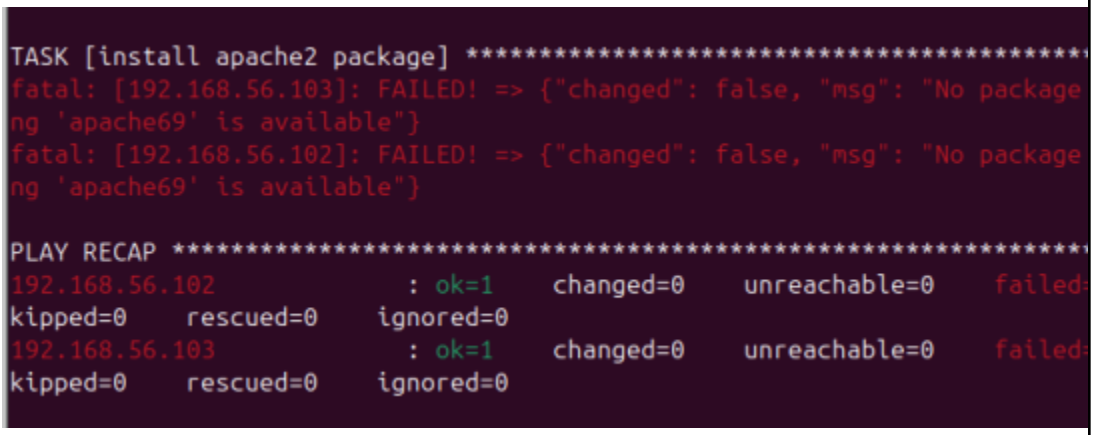
```
 install_apache.yml --ask-become-pass
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.103]
ok: [192.168.56.102]

TASK [install apache2 package] *************************************************
ok: [192.168.56.103]
ok: [192.168.56.102]

PLAY RECAP *********************************************************************
192.168.56.102             : ok=2    changed=0    unreachable=0    failed
kipped=0    rescued=0    ignored=0
192.168.56.103             : ok=2    changed=0    unreachable=0    failed
kipped=0    rescued=0    ignored=0
```

The result of the command successfully installed the apache. No error, all servers are reachable.

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.



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?



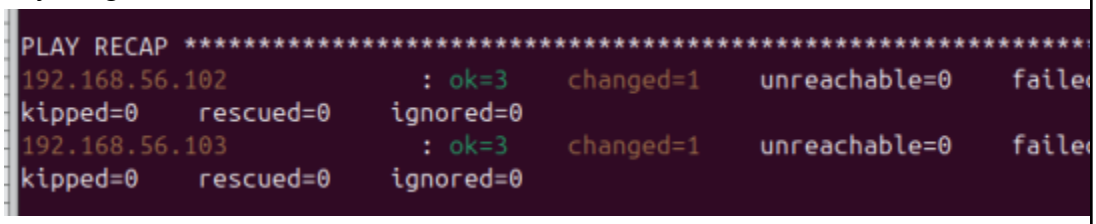
It failed because no apache69 is available.

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

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

  - name: update repository index
    apt:
      update_cache: yes

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

Save the changes to this file and exit.

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



It was ok, the output showed that the playbook executed successfully. The update_cache command refreshed the package.

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?

```
TASK [add PHP support for apache] ************************************
ok: [192.168.56.103]
ok: [192.168.56.102]

PLAY RECAP ***********************************************************
192.168.56.102             : ok=4    changed=1    unreachable=0    faile
kipped=0    rescued=0    ignored=0
192.168.56.103             : ok=4    changed=1    unreachable=0    faile
kipped=0    rescued=0    ignored=0
```

No, The output showed that PHP and its related modules were installed on the remote servers

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

```
joshuabueno@workstation:~/CpE212$ git init
Reinitialized existing Git repository in /home/joshuabueno/CpE212/.git/
joshuabueno@workstation:~/CpE212$ git remote add origin git@github.com:j
no123/CPE232_LABS.git
joshuabueno@workstation:~/CpE212$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 562 bytes | 112.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/joshuabueno123/CPE232_LABS/pull/new/mast
remote:
To github.com:joshuabueno123/CPE232_LABS.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Github Link:
joshuabueno123/CPE232_LABS

Reflections:
Answer the following:

1. What is the importance of using a playbook?
   -The importance of using a playbook is to standardize and automate tasks, ensuring consistency and efficiency in IT operations. Playbooks serve as a single source of truth for procedures, reducing the risk of human error and enabling teams to execute complex or repetitive processes reliably

2. Summarize what we have done on this activity.
   - In this activity, we ran elevated ad hoc commands. We did this to perform quick, one-off tasks that required elevated privileges on managed hosts. By executing these commands, we were able to perform administrative actions and make system-level changes without having to create a full, formal playbook. This demonstrated how to efficiently manage specific, non-routine tasks across multiple servers.