Name: Potestades, North Nygel G.	**Date Performed:** 9/5/25
Course/Section: CPE31S2	**Date Submitted:** 9/5/25
Instructor: Engr. Robin Valenzuela	**Semester and SY:** 1st Semester 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

```
north@workstation:~/Potestades_PrelimExam$ ansible all -m apt -a update_cache=t
rue
192.168.56.106 | FAILED! => {
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation"
}
192.168.56.105 | FAILED! => {
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation"
}
192.168.56.107 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168
.56.107 port 22: No route to host\r\n",
    "unreachable": true
}
```

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

It is not successful, coming up with the same error for both. The unreachable IP address is the CentOS managed node which was set-up in the previous HOA.

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.

```
north@workstation:~/Potestades_PrelimExam$ ansible all -m apt -a update_cache=t
rue --become --ask-become-pass
SUDO password:
192.168.56.107 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168
.56.107 port 22: No route to host\r\n",
    "unreachable": true
}
```

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

Yes, it is. The request was taking a long time to process, so the screenshot is of the command in progress.

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.

```
north@workstation:~/Potestades_PrelimExam$ ansible all -m apt -a name=vim-nox -
-become --ask-become-pass
SUDO password:
192.168.56.106 | SUCCESS => {
    "cache_update_time": 1757063192,
    "cache_updated": false,
    "changed": false
}
192.168.56.105 | SUCCESS => {
    "cache_update_time": 1757063130,
    "cache_updated": false,
    "changed": false
}
192.168.56.107 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168
.56.107 port 22: No route to host\r\n",
    "unreachable": 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?

```
north@server1:~$ which vim
/usr/bin/vim
north@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
```

```
north@server2:~$ which vim
/usr/bin/vim
north@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 [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
```

Yes, it was 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.

```
  GNU nano 2.9.3                    history.log


Start-Date: 2025-09-05  17:03:13
Commandline: apt install vim-nox
Requested-By: north (1000)
Install: javascript-common:amd64 (11, automatic), ruby2.5:amd64 (2.5.1-1ubuntu$
End-Date: 2025-09-05  17:03:21
```

```
  GNU nano 2.9.3                    history.log


Start-Date: 2025-09-05  17:01:28
Commandline: apt install vim-nox
Requested-By: north (1000)
Install: javascript-common:amd64 (11, automatic), ruby2.5:amd64 (2.5.1-1ubuntu$
End-Date: 2025-09-05  17:01:36
```

What you can see in the history.log file is the history of commands which used apt, showing the apt install requests and the apt update requests delivered by the control node.

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?

```
north@workstation:~/CPE212_Potestades$ ansible all -m apt -a name=snapd --becom
e --ask-become-pass
SUDO password:
192.168.56.106 | SUCCESS => {
    "cache_update_time": 1757063192,
    "cache_updated": false,
    "changed": false
}
192.168.56.105 | SUCCESS => {
    "cache_update_time": 1757063130,
    "cache_updated": false,
    "changed": false
}
```

Yes, it is a success. It checked it the latest version of snapd was installed, then installed it if it wasn't.

   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.

```
north@workstation:~/CPE212_Potestades$ ansible all -m apt -a "name=snapd state=
latest" --become --ask-become-pass
SUDO password:
192.168.56.105 | SUCCESS => {
    "cache_update_time": 1757063130,
    "cache_updated": false,
    "changed": false
}
192.168.56.106 | SUCCESS => {
    "cache_update_time": 1757063192,
    "cache_updated": false,
    "changed": false
}
```

The output is the same, with no changes other than the time being different. This is because the package is already updated to the latest version.

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

```
north@workstation:~/CPE212_Potestades$ git commit -m "Update 1 HOA 4.1"
[main 32e3ca4] Update 1 HOA 4.1
 2 files changed, 13 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 inventory.yaml
north@workstation:~/CPE212_Potestades$ git push origin main
Warning: Permanently added the ECDSA host key for IP address '20.205.243.166' t
o the list of known hosts.
Counting objects: 4, done.
```

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.

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

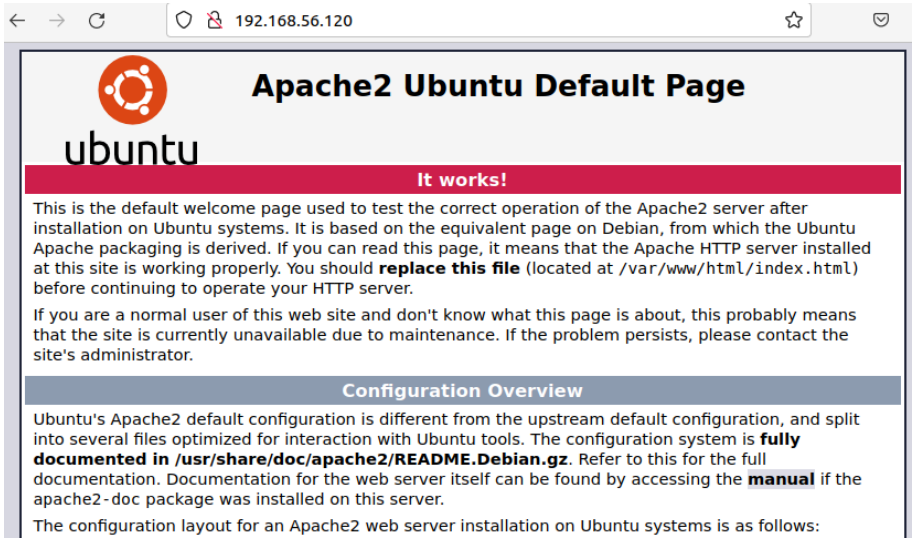
```
TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.106]
ok: [192.168.56.105]
fatal: [192.168.56.107]: UNREACHABLE! => {"changed": false, "msg": "Failed to c
onnect to the host via ssh: ssh: connect to host 192.168.56.107 port 22: No rou
te to host\r\n", "unreachable": true}

TASK [install apache2 package] *************************************************
*
ok: [192.168.56.106]
ok: [192.168.56.105]
        to retry, use: --limit @/home/north/CPE212_Potestades/install_apache.re
try

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=2    changed=0    unreachable=0    failed=0
192.168.56.106             : ok=2    changed=0    unreachable=0    failed=0
```

It took a little while, but I was able to install apache2 onto the remote 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. 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?

```
TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]
ok: [192.168.56.106]
fatal: [192.168.56.107]: UNREACHABLE! => {"changed": false, "msg": "Failed to c
onnect to the host via ssh: ssh: connect to host 192.168.56.107 port 22: No rou
te to host\r\n", "unreachable": true}

TASK [install apache2 package] *************************************************
*
fatal: [192.168.56.106]: FAILED! => {"changed": false, "msg": "No package match
ing 'asd' is available"}
fatal: [192.168.56.105]: FAILED! => {"changed": false, "msg": "No package match
ing 'asd' is available"}
```

The output is an error which says that there is no matching package.

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?

```
PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=2    changed=1    unreachable=0    failed=1
192.168.56.106             : ok=2    changed=1    unreachable=0    failed=1
```

Yes, it updated the package repository index on the remote hosts. The changed is the new command, while the failed command is the package name that will not be recognized.

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?

```
PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=4    changed=0    unreachable=0    failed=0
192.168.56.106             : ok=4    changed=0    unreachable=0    failed=0
```

Yes, it added the new package libapache2-mod-php which supports PHP for the apache package.

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

northpotestades/CPE212_Potestades

Reflections:

Answer the following:

1. What is the importance of using a playbook?

Playbooks are so important because they let you automate the process of running commands on many different remote hosts. Because of this, you do not need to manually SSH into each host and execute the commands you need to update the system, saving you a lot of valuable time.

2. Summarize what we have done on this activity.

In this activity, we first saw the power which an elevated command has, letting you run sudo level commands on the remote host, which was followed by installing vim-nox and the latest version of snapd onto the remote hosts. Lastly, we wrote a playbook which updated the repository index (sudo apt update), installed apache2, and installed libapache2-mod-php on the remote hosts. All of this was then pushed to my GitHub repository for the course.