

Name: ESCOSIA, JERICO	Date Performed: 9/15/24
Course/Section: CPE31S21	Date Submitted: 9/15/24
Instructor: Engr. Valenzuela	Semester and SY:
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: <i>Provide screenshots for each task.</i> 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 <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. Issued the following command:

```
workstation@workstation:~/HOA4$ ansible all -m apt -a update_cache=true
server1 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ubuntu@192.168.56.108: Permission denied (publickey,password).",
  "unreachable": true
}
server2 | 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",
  "unreachable": true
}
```

ansible all -m apt -a update_cache=true

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

No, it requires higher permission to perform the task.

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.

```
workstation@workstation:~/HOA4$ ansible all -m apt -a update_cache=true --become --ask-become-pass
BECOME password:
server1 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329524,
  "cache_updated": true,
  "changed": true
}
server2 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329524,
  "cache_updated": true,
  "changed": true
}
```

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.

```
workstation@workstation:~/H0A4$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
server1 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329704,
  "cache_updated": false,
  "changed": true,
  "stderr": "",
  "stderr_lines": [],
  "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\n\nThe following
additional packages will be installed:\n fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby libruby3.1
\n libsodium23 rake ruby ruby-net-telnet ruby-rubygems ruby-sdbm ruby-webrick\n ruby-xmllrpc ruby3.1 rubygems-int
egration vim-runtime\n\nSuggested packages:\n apache2 | lighttpd | httpd ri ruby-dev bundler cscope vim-doc\n\nThe fo
llowing NEW packages will be installed:\n fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby libruby3.
1\n libsodium23 rake ruby ruby-net-telnet ruby-rubygems ruby-sdbm ruby-webrick\n ruby-xmllrpc ruby3.1 rubygems-int
egration vim-nox vim-runtime\n0 upgraded, 18 newly installed, 0 to remove and 0 not upgraded.\nNeed to get 18.1 M
B of archives.\nAfter this operation, 81.8 MB of additional disk space will be used.\nGet:1 http://ph.archive.ubun
tu.com/ubuntu mantic/main amd64 fonts-lato all 2.0-2.1 [2696 kB]\nGet:2 http://ph.archive.ubuntu.com/ubuntu mantic
/main amd64 javascript-common all 11+nmu1 [5936 B]\nGet:3 http://ph.archive.ubuntu.com/ubuntu mantic/main amd64 li
bjs-jquery all 3.6.1+dfsg+~3.5.14-1 [328 kB]\nGet:4 http://ph.archive.ubuntu.com/ubuntu mantic/universe amd64 libl
ua5.2-0 amd64 5.2.4-3 [122 kB]\nGet:5 http://ph.archive.ubuntu.com/ubuntu mantic/main amd64 rubygems-integration a
ll 1.18 [5336 B]\nGet:6 http://ph.archive.ubuntu.com/ubuntu mantic-updates/main amd64 ruby3.1 amd64 3.1.2-7ubuntu3
.3 [49.0 kB]\nGet:7 http://ph.archive.ubuntu.com/ubuntu mantic/main amd64 ruby-rubygems all 3.3.15-2 [231 kB]\nGet
:8 http://ph.archive.ubuntu.com/ubuntu mantic/main amd64 ruby amd64 1:3.1 [3470 B]\nGet:9 http://ph.archive.ubuntu
```

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**

```
workstation@server1:~$ which vim
/usr/bin/vim
workstation@server1:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/mantic-updates,now 2:9.0.1672-1ubuntu2.4 amd64 [installed]
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/mantic-updates,now 2:9.0.1672-1ubuntu2.4 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

workstation@server1:~$
```

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: 2024-09-15 00:02:44
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=-force-confdef -o Dpkg::Options::=-force-confold install vim-nox=2:9.0.1672-1ubuntu2.4
Requested-By: workstation (1000)
Install: ruby-sdbm:amd64 (1.0.0-5build2, automatic), fonts-lato:amd64 (2.0-2.1, automatic), liblua5.2-0:amd64 (5.2.4-3, automatic), libruby:amd64 (1:3.1, automatic), ruby-net-telnet:amd64 (0.2.0-1, automatic), rubygems-integration:amd64 (1.18, automatic), libruby3.1:amd64 (3.1.2-7ubuntu3.3, automatic), rake:amd64 (13.0.6-3, automatic), libsodium23:amd64 (1.0.18-1build2, automatic), vim-nox:amd64 (2:9.0.1672-1ubuntu2.4), ruby:amd64 (1:3.1, automatic), vim-runtime:amd64 (2:9.0.1672-1ubuntu2.4, automatic), ruby3.1:amd64 (3.1.2-7ubuntu3.3, automatic), libjs-jquery:amd64 (3.6.1+dfsg+~3.5.14-1, automatic), ruby-rubygems:amd64 (3.3.15-2, automatic), javascript-common:amd64 (11+nmu1, automatic), ruby-xmllrpc:amd64 (0.3.2-2, automatic), ruby-webrick:amd64 (1.8.1-1, automatic)
End-Date: 2024-09-15 00:02:52
workstation@server1:/var/log/apt$
```

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*

```
workstation@workstation:~/HOA4$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
server1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329704,
  "cache_updated": false,
  "changed": false
}
server2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329704,
  "cache_updated": false,
  "changed": false
}
```

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

The installation of snapd is successful but the change was not yet on the 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.

```
workstation@workstation:~/HOA4$ ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
BECOME password:
server1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329704,
  "cache_updated": false,
  "changed": false
}
server2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1726329704,
  "cache_updated": false,
  "changed": false
}
```

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

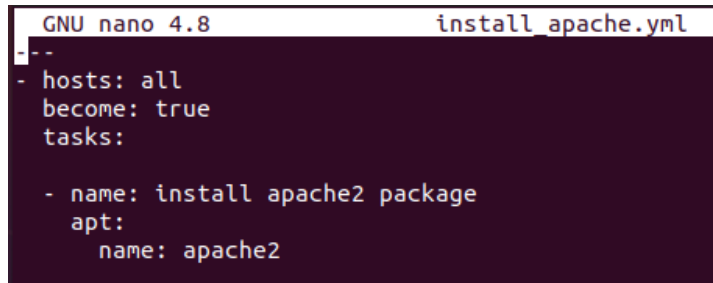
```
workstation@workstation:~/HOA4$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
workstation@workstation:~/HOA4$
```

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.

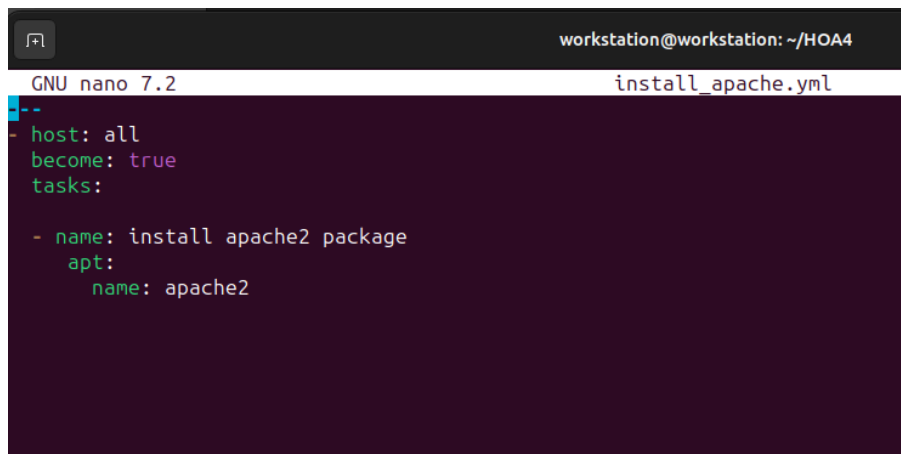
When the editor appears, type the following:



```
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.



```
workstation@workstation: ~/HOA4
GNU nano 7.2      install_apache.yml
--
- host: all
  become: true
  tasks:

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

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

```
workstation@workstation:~/H0A4$ nano install_apache.yml
workstation@workstation:~/H0A4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

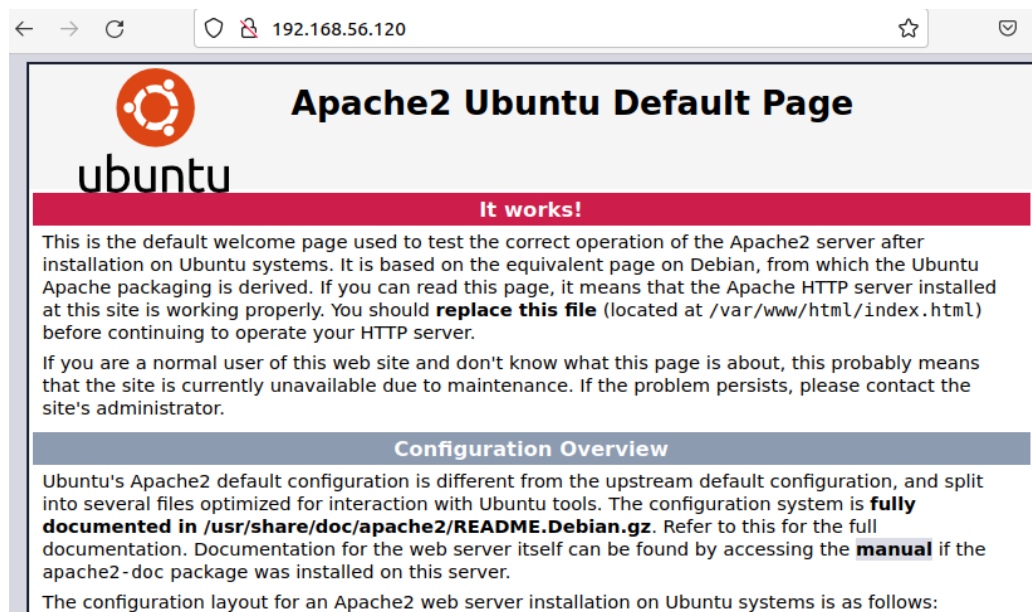
PLAY [all] *****

TASK [Gathering Facts] *****
ok: [server2]
ok: [server1]

TASK [install apache2 package] *****
changed: [server1]
changed: [server2]

PLAY RECAP *****
server1      : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
server2      : ok=2    changed=1    unreachable=0    failed=0    skipped=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.





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?

The installation failed because the package name does not exist.

```

workstation@workstation: ~/HOA4
GNU nano 7.2                                install_apache.yml
...
hosts: all
become: true
tasks:
  - name: install apachwfsdfe2 package
    apt:
      name: apachechwfsdfe2

workstation@workstation: ~/HOA4$ nano install_apache.yml
workstation@workstation: ~/HOA4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [server2]
ok: [server1]

TASK [install apachwfsdfe2 package] *****
fatal: [server1]: FAILED! => {changed: false, msg: "No package matching 'apachechwfsdfe2' is available"}
fatal: [server2]: FAILED! => {changed: false, msg: "No package matching 'apachechwfsdfe2' is available"}

PLAY RECAP *****
server1                : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
server2                : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
  
```

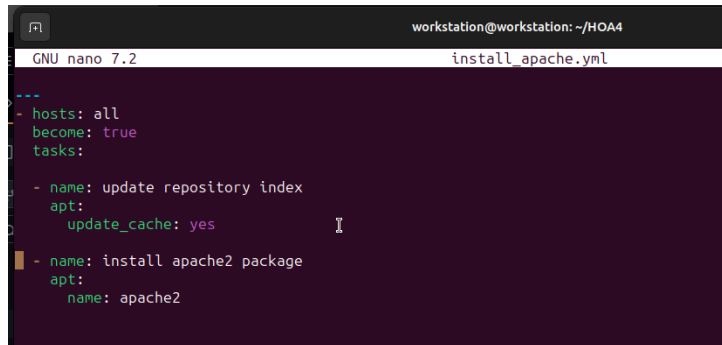
5. This time, we are going to put additional task to our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional command, which is the *update_cache*. This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities)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



```
workstation@workstation: ~/HOA4
GNU nano 7.2      install_apache.yml

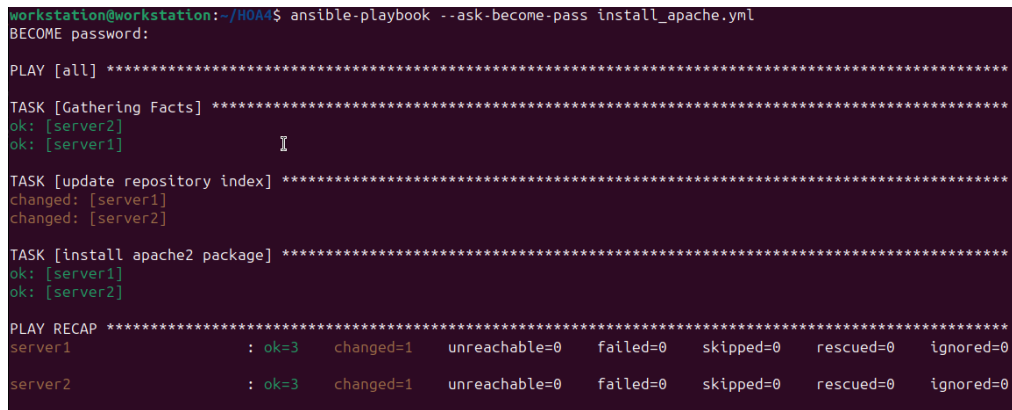
---
- hosts: all
  become: true
  tasks:

    - 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?

Yes, the new command updated the repository index on the remote servers.



```
workstation@workstation:~/HOA4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [server2]
ok: [server1]

TASK [update repository index] *****
changed: [server1]
changed: [server2]

TASK [install apache2 package] *****
ok: [server1]
ok: [server2]

PLAY RECAP *****
server1      : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
server2      : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```


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

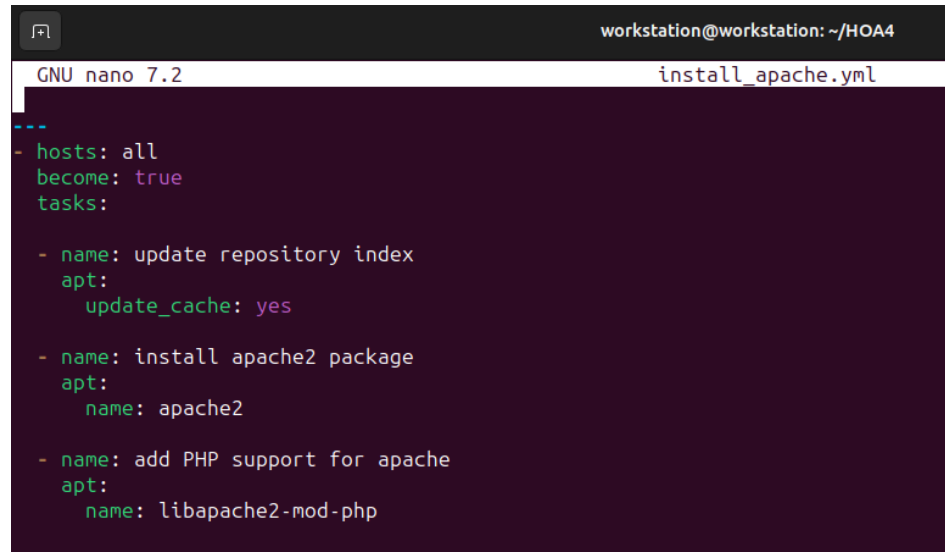
```
---
- hosts: all
  become: true
  tasks:

    - 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.



```
workstation@workstation: ~/HOA4
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?

The last command added PHP support on the apache package that was installed earlier.

```
workstation@workstation:~/HOA4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [server2]
ok: [server1]

TASK [update repository index] *****
changed: [server1]
changed: [server2]

TASK [install apache2 package] *****
ok: [server1]
ok: [server2]

TASK [apt] *****
changed: [server1]
changed: [server2]

PLAY RECAP *****
server1      : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
server2      : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

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

```
workstation@workstation:~/HOA4$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 488 bytes | 488.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:emrys66/HOA4.git
   dec4447..cfc2768  main -> main
```

Repo link: <https://github.com/emrys66/HOA4.git>

Reflections:

Answer the following:

1. What is the importance of using a playbook?

The ansible playbook is important because this is where we issue that command we want to perform in our remote servers. Using a playbook we can configure the remote server using automation.

2. Summarize what we have done on this activity.

In this activity we learned what is ansible, how it is used in automating configuring from workstation to the remote servers. We set up the configurations in github and use git commands in the terminal to upload the files we created to our repositories.

