Name: Recto, Jon Jeous J. **Date Performed:** 02/05/24

Course/Section: CPE31S1 **Date Submitted:**

Instructor: Dr. Jonathan V. Taylar **Semester and SY:** 2nd Sem 23-24

Activity 4: Running Elevated Ad hoc Commands

1. Objectives:

1.1Use commands that makes changes to remote machines

1.2Use 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

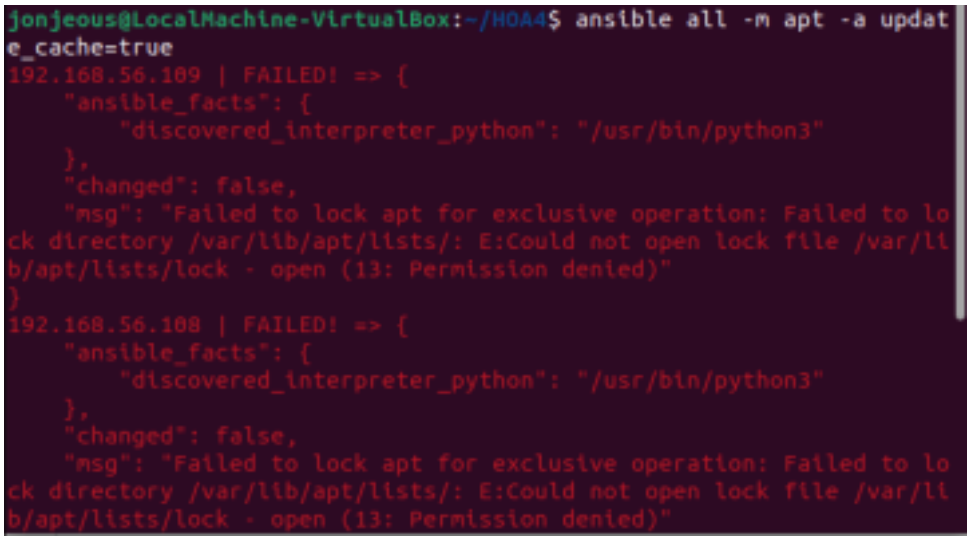
Task 1: Run elevated ad hoc commands

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?



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.

```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ ansible all -m apt -a updat
e_cache=true --become --ask-become-pass
BECOME password:
192.168.56.106 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883237,
    "cache_updated": true,
    "changed": true
}
192.168.56.108 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "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 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.

```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ ansible all -m apt -a name=
vim-nox --become --ask-become-pass
BECOME password:
192.168.56.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "cache_updated": false,
    "changed": false
}
192.168.56.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "cache_updated": false,
    "changed": false
}
```

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?

```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ which vim
/usr/bin/vim
jonjeous@LocalMachine-VirtualBox:~/HOA4$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.15 amd6
4 [installed]
  Vi IMproved - enhanced vi editor - with scripting languages suppor
t

vim-tiny/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.15 amd
64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

jonjeous@LocalMachine-VirtualBox:~/HOA4$
```

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.

```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ cd /var/log
jonjeous@LocalMachine-VirtualBox:/var/log$ ls
alternatives.log  dmesg       fontconfig.log   openvpn
apt               dmesg.0     gdm3             private
auth.log          dmesg.1.gz  gpu-manager.log  speech-dispatcher
boot.log          dmesg.2.gz  hp               syslog
bootstrap.log     dmesg.3.gz  installer        ubuntu-advantage.log
btmp              dmesg.4.gz  journal          unattended-upgrades
cups              dpkg.log    kern.log         wtmp
dist-upgrade      faillog     lastlog
jonjeous@LocalMachine-VirtualBox:/var/log$ cd apt
jonjeous@LocalMachine-VirtualBox:/var/log/apt$ cat history.log

Start-Date: 2023-08-07  22:53:16
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes --f
orce-yes upgrade
Upgrade: dpkg:amd64 (1.21.1ubuntu2, 1.21.1ubuntu2.2), libxtables12:a
md64 (1.8.7-1ubuntu5, 1.8.7-1ubuntu5.1), networkd-dispatcher:amd64 (
2.1-2, 2.1-2ubuntu0.22.04.2), libpam-runtime:amd64 (1.4.0-11ubuntu2,
 1.4.0-11ubuntu2.3), python3.10:amd64 (3.10.4-3, 3.10.12-1~22.04.2),
 python3-gi:amd64 (3.42.0-3build1, 3.42.1-0ubuntu1), libext2fs2:amd6
```

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?

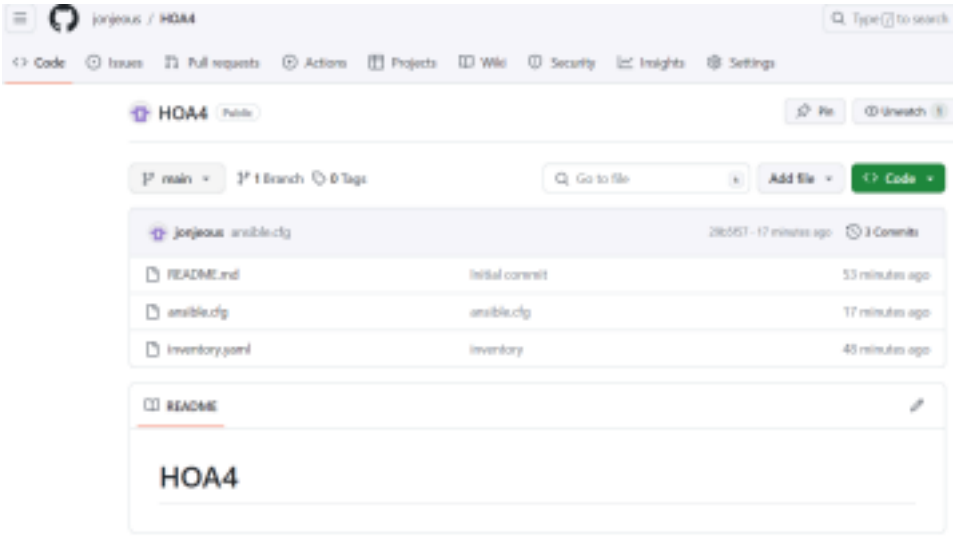
```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ ansible all -m apt -a name=
snapd --become --ask-become-pass
BECOME password:
192.168.56.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "cache_updated": false,
    "changed": false
}
192.168.56.106 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883521,
    "cache_updated": false,
    "changed": false
}
```

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 p laced them in double quotations.

```
jonjeous@LocalMachine-VirtualBox:~/HOA4$ ansible all -m apt -a "name
=snapd state=latest" --become --ask-become-pass
BECOME password:
192.168.56.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "cache_updated": false,
    "changed": false
}
192.168.56.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1707883239,
    "cache_updated": false,
    "changed": false
}
192.168.56.106 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
```

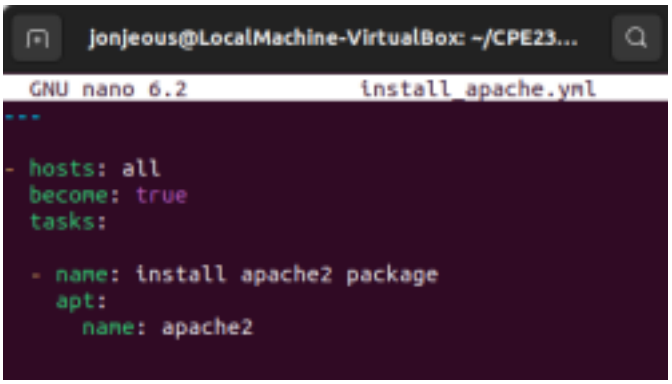
4. At this point, make sure to commit all changes to GitHub.



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:

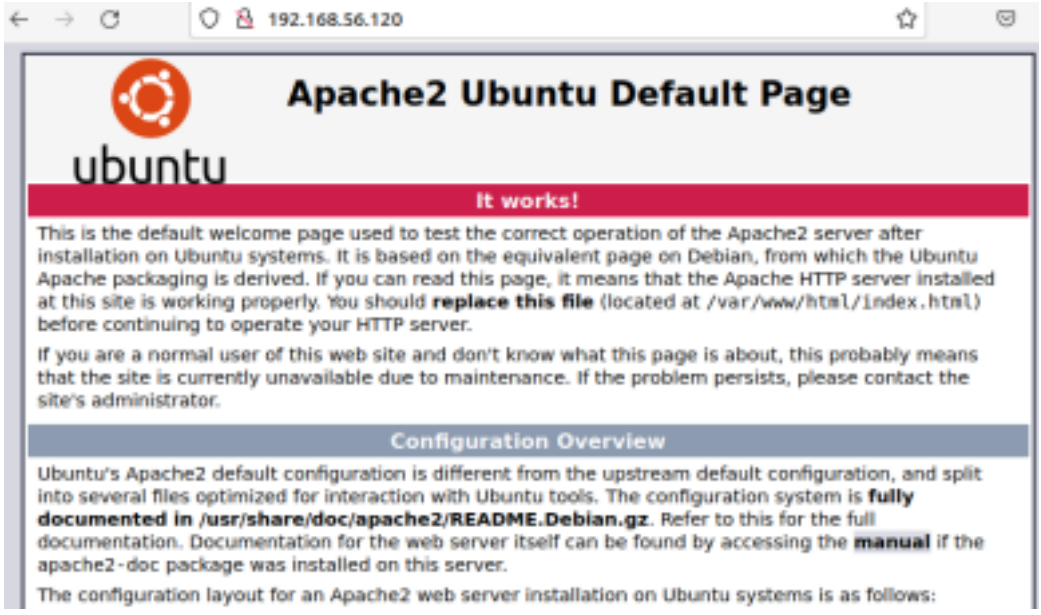


   Make sure to save the file. Take note also of the alignments of the texts.

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

```
jonjeous@LocalMachine-VirtualBox:~/CPE232_Recto$ ansible-playbook --
ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************
************
ok: [192.168.56.106]
ok: [192.168.56.109]
ok: [192.168.56.108]

TASK [install apache2 package] *************************************
************
changed: [192.168.56.106]
changed: [192.168.56.109]
changed: [192.168.56.108]

PLAY RECAP *********************************************************
************
```

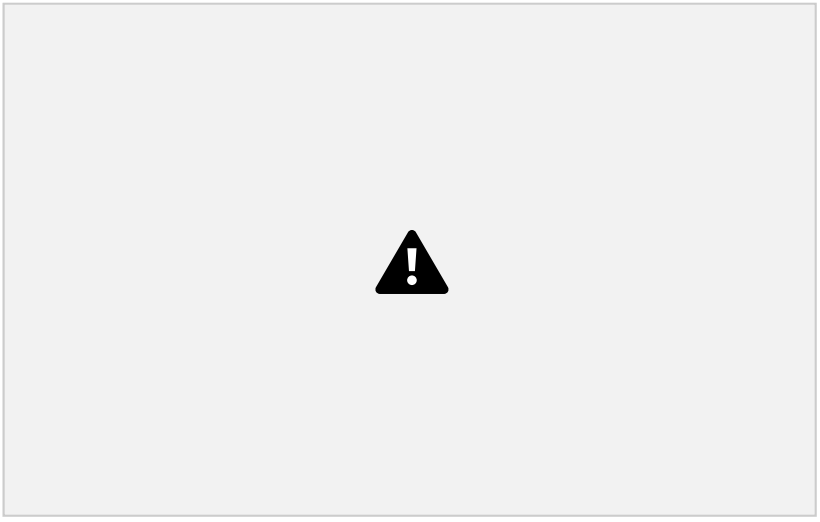
3. To verify that apache2 was installed automatically in the remote servers, go to the web browsers on each server and type its IP address. You should see something like this.



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?

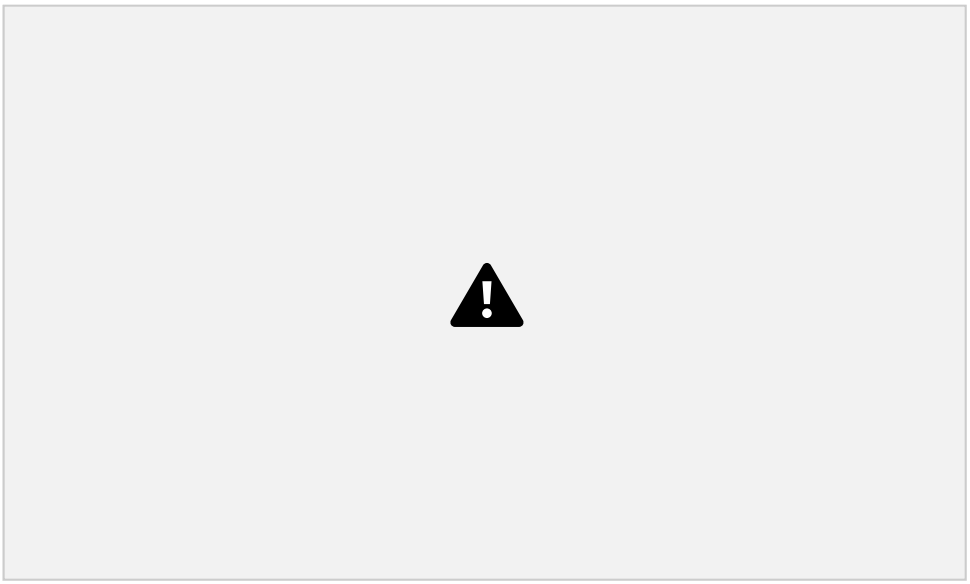
```
TASK [install apache2 package] *************************************
************
fatal: [192.168.56.106]: FAILED! => {"changed": false, "msg": "No pa
ckage matching 'jonjeous' is available"}
fatal: [192.168.56.109]: FAILED! => {"changed": false, "msg": "No pa
ckage matching 'jonjeous' is available"}
fatal: [192.168.56.108]: FAILED! => {"changed": false, "msg": "No pa
ckage matching 'jonjeous' is available"}

PLAY RECAP *********************************************************
************
192.168.56.106             : ok=1    changed=0    unreachable=0    f
ailed=1    skipped=0    rescued=0    ignored=0
192.168.56.108             : ok=1    changed=0    unreachable=0    f
ailed=1    skipped=0    rescued=0    ignored=0
192.168.56.109             : ok=1    changed=0    unreachable=0    f
ailed=1    skipped=0    rescued=0    ignored=0

jonjeous@LocalMachine-VirtualBox:~/CPE232_Recto$
```

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.

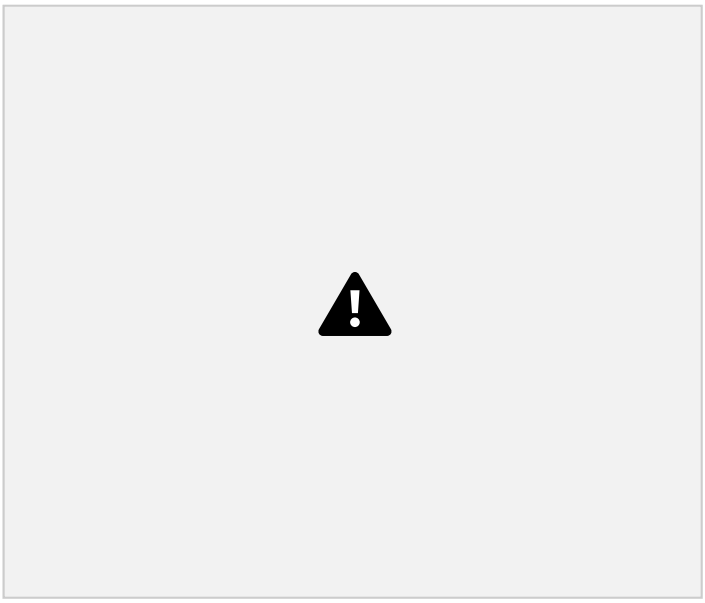


Save the changes to this file and exit.

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

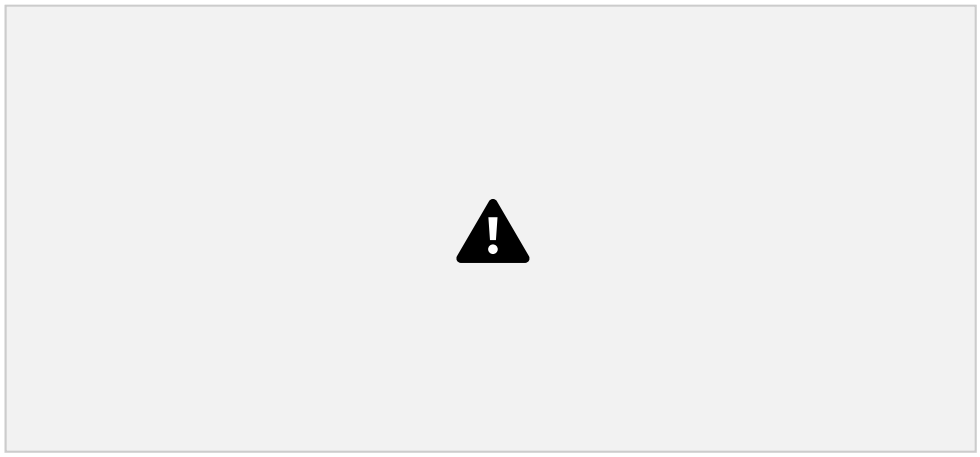


7. Edit again the *install_apache.yml*. This time, we are going to add a PHP support for the apache package we installed earlier.

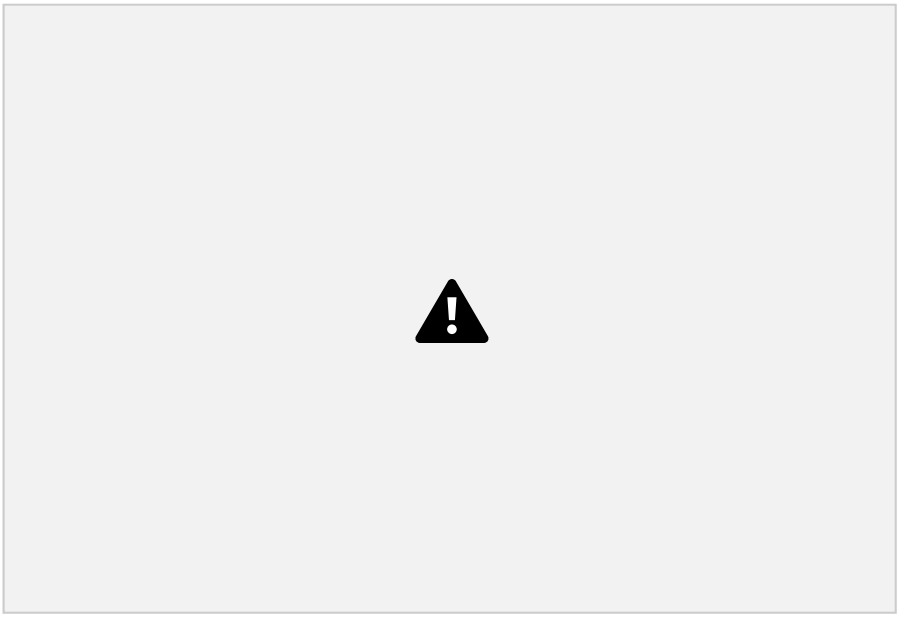


Save the changes to this file and exit.

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



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



https://github.com/jonjeous/CPE232_Recto

Reflections:

Answer the following:

1. What is the importance of using a playbook?

In Ansible, using a playbook is like having a helpful assistant that automates tasks on multiple systems, making sure everything is consistent and easy to reuse.

2. Summarize what we have done on this activity.

To sum it up, I successfully completed the task of making changes to remote machines using Ansible. Ansible commands helped me easily interact with these systems for tasks like configuration updates. I also utilized Ansible playbooks to automate these processes across multiple machines. This combination of commands and playbooks provided an efficient and organized approach to configuring and managing remote machines, ensuring consistency and scalability in my operations.