Name: Kazuki A. Ogata	**Date Performed:** September 12, 2023
Course/Section: CPE 232 - CPE31S5	**Date Submitted:** September 17, 2023
Instructor: Engr. Roman Richard	**Semester and SY:** 1st semester S.Y 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:**

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:

```
kazuki@workstation:~$ python3 -m pip install --user ansible
Requirement already satisfied: ansible in ./.local/lib/python3.10/site-packages (8.4.0)
Requirement already satisfied: ansible-core~=2.15.4 in ./.local/lib/python3.10/site-packages (from ansible) (2.15.4)
Requirement already satisfied: jinja2>=3.0.0 in /usr/lib/python3/dist-packages (from ansible-core~=2.15.4->ansible) (3.0.3)
Requirement already satisfied: PyYAML>=5.1 in /usr/lib/python3/dist-packages (from ansible-core~=2.15.4->ansible) (5.4.1)
Requirement already satisfied: cryptography in /usr/lib/python3/dist-packages (from ansible-core~=2.15.4->ansible) (3.4.8)
Requirement already satisfied: packaging in /usr/lib/python3/dist-packages (from ansible-core~=2.15.4->ansible) (21.3)
Requirement already satisfied: resolvelib<1.1.0,>=0.5.3 in ./.local/lib/python3.10/site-packages (from ansible-core~=2.15.4->ansible
) (1.0.1)
```

Figure 1.1.1 - Installing Ansible

```
[defaults]
inventory = hosts
```

```
[localhost]
192.168.56.112 ansible_connection=local
192.168.56.113 ansible_connection=local
```

Figure 1.1.2 - Creating an Inventory

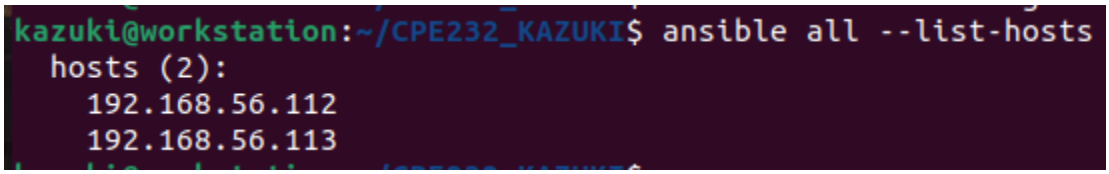

Figure 1.1.3 - Verifying Host in Inventory

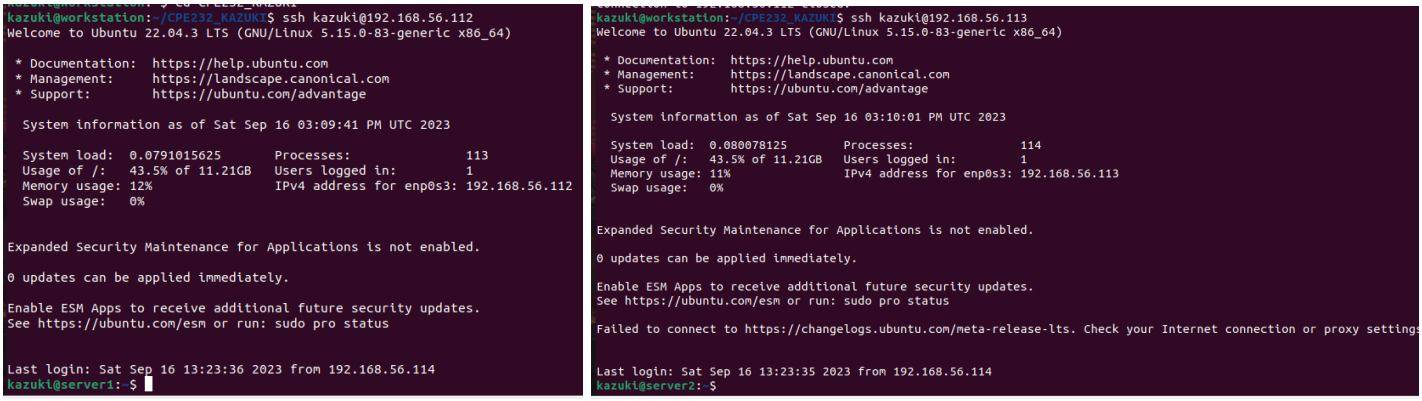

Figure 1.1.4-5 - SSH Connections Verification

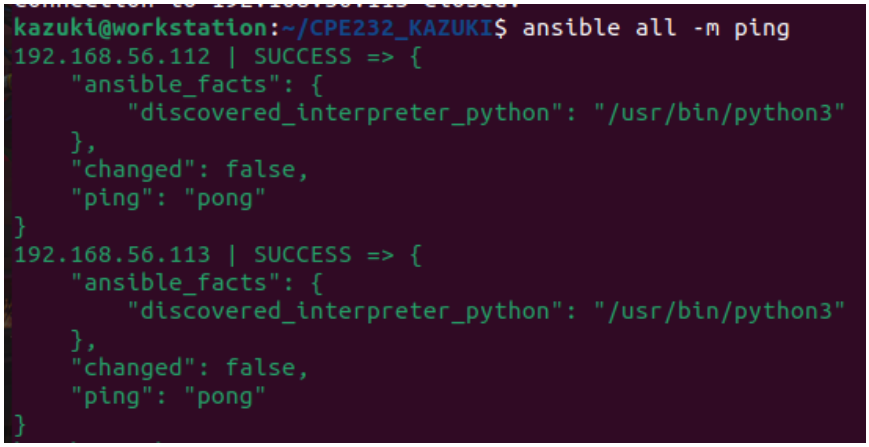

Figure 1.1.6 - Pinging Manage Nodes

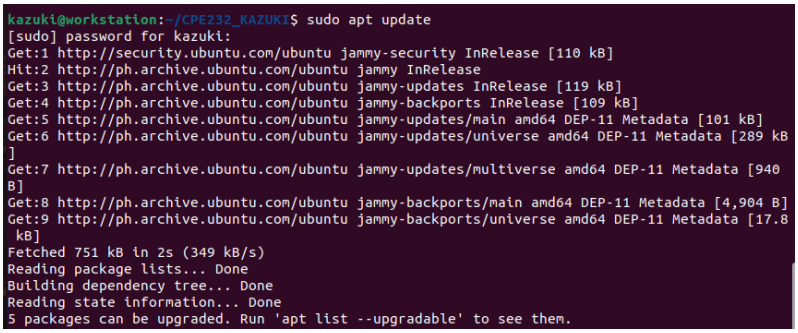

Figure 1.1.7 - Updating Package Cache

This command downloads package information from all configured resources to make sure that we have the latest information about those packages in our system.

ansible all -m apt -a update_cache=true
What is the result of the command? Is it successful?

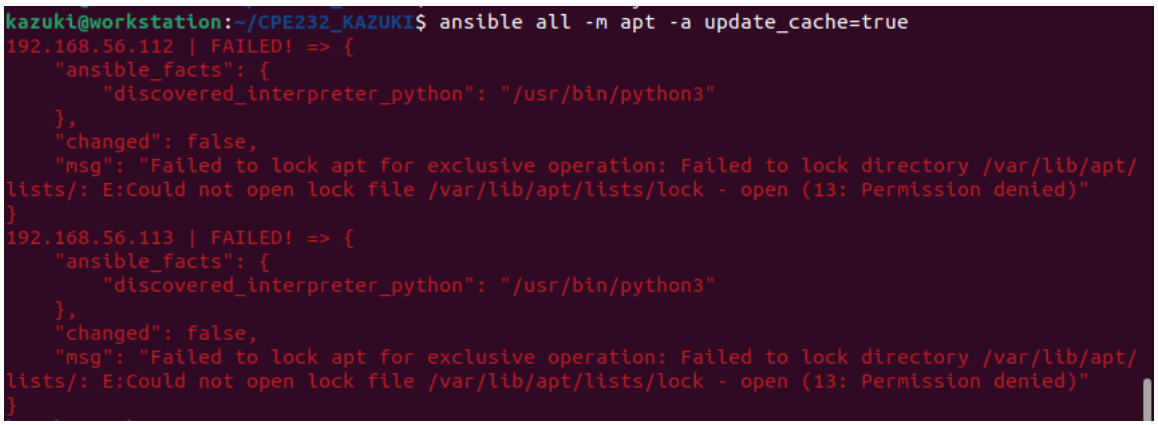

Figure 1.1.8 - Updating Package Cache using Ad Hoc Command (FAILED)

The result of the command is unsuccessful because Ansible lacks necessary permissions in order to run the command successfully.

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.

Figure 1.1.9 - Updating Package Cache using Ad Hoc Command (CHANGED)

The result of the command is now successful, this is because of the “--become” command that gives privilege to the command, allowing Ansible to update the package cache successfully.

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.

Figure 1.2.1 - Installing VIM

The ansible command "ansible all -m apt -a name=vim-nox --become --ask-become-pass" installs the package vim-nox on all configured servers. The –become ensures that it has the right permissions for installation so there will be no problem.

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?

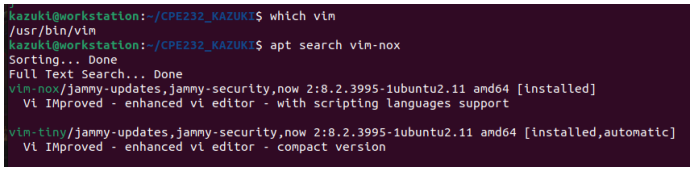

Figure 1.2.2 - Verifying Installed VIM

The "which vim" command shows the path of VIM and the "apt search vim-nox" command searches and displays the vim-nox package. These two commands showed and verified that we successfully installed VIM.

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.

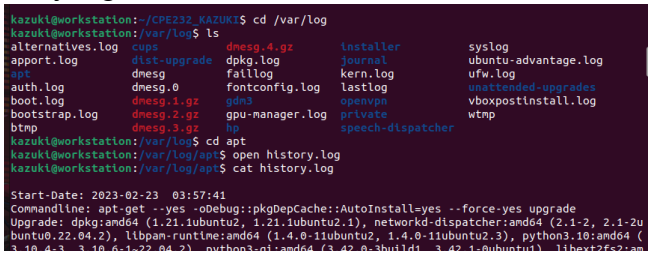

Figure 1.2.3 - Checking the Logs in Servers

The "open" command prompts a new tab containing the history of package management actions in the apt directory, so I just use the "cat" command since they are similar.

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?

Figure 1.3.1 - Installing Snapd

I successfully installed the snapd package 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.

Figure 1.3.2 - Installing Snapd (state=latest)

The output is the same above without the “state=latest” both installed the latest version, but with “state=latest” make sure that the Ansible command will install the latest package of snapd.

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

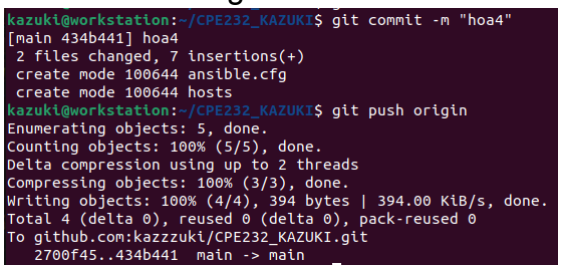

Figure 1.4.1 - Committing Changes to GitHub

I used the command “commit” to commit all changes to my GitHub. With “-m” I was able to put a commit message. The “git push origin” command pushed the committed changes from my repository (CPE232_KAZUKI) to a remote repository.

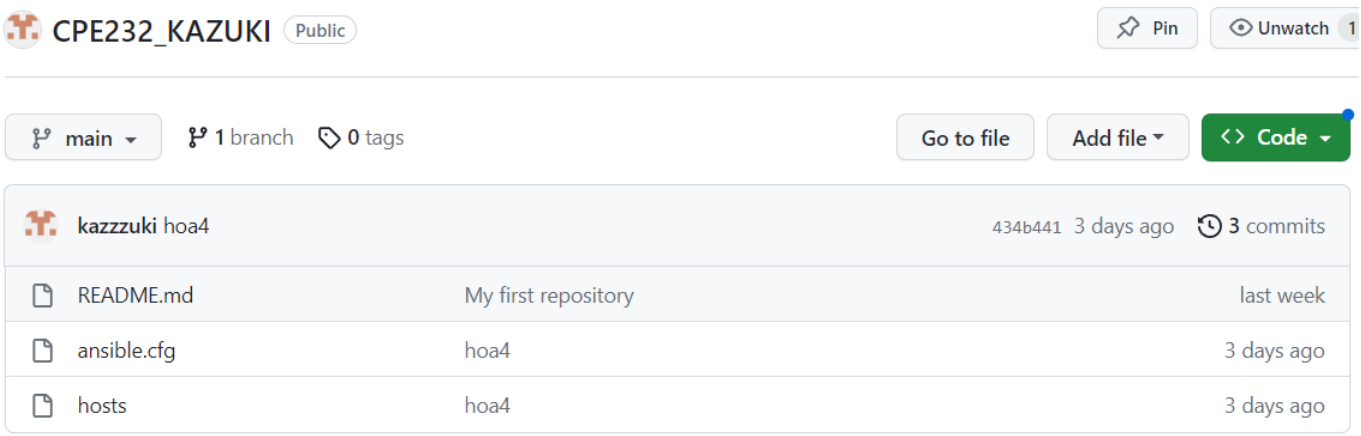

Figure 1.4.2 - GitHub

As we can see, it successfully pushes the inventories that I created with “hoa” as its repository name.

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:

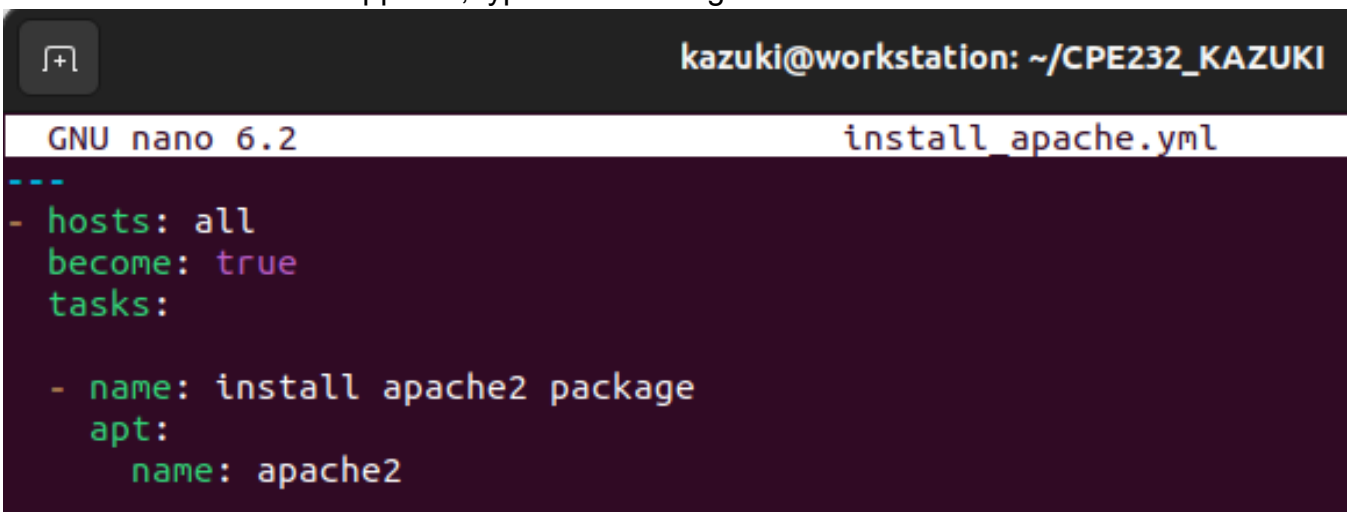

Figure 2.1 - Writing Playbook: Installing Apache2 Package

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.

```
kazuki@workstation:~/CPE232_KAZUKI$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

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

PLAY RECAP *********************************************************************************
192.168.56.112             : ok=2    changed=0    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
192.168.56.113             : ok=2    changed=0    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
```

Figure 2.2 - Running Playbook

It successfully executed the tasks inside my Playbook.

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.

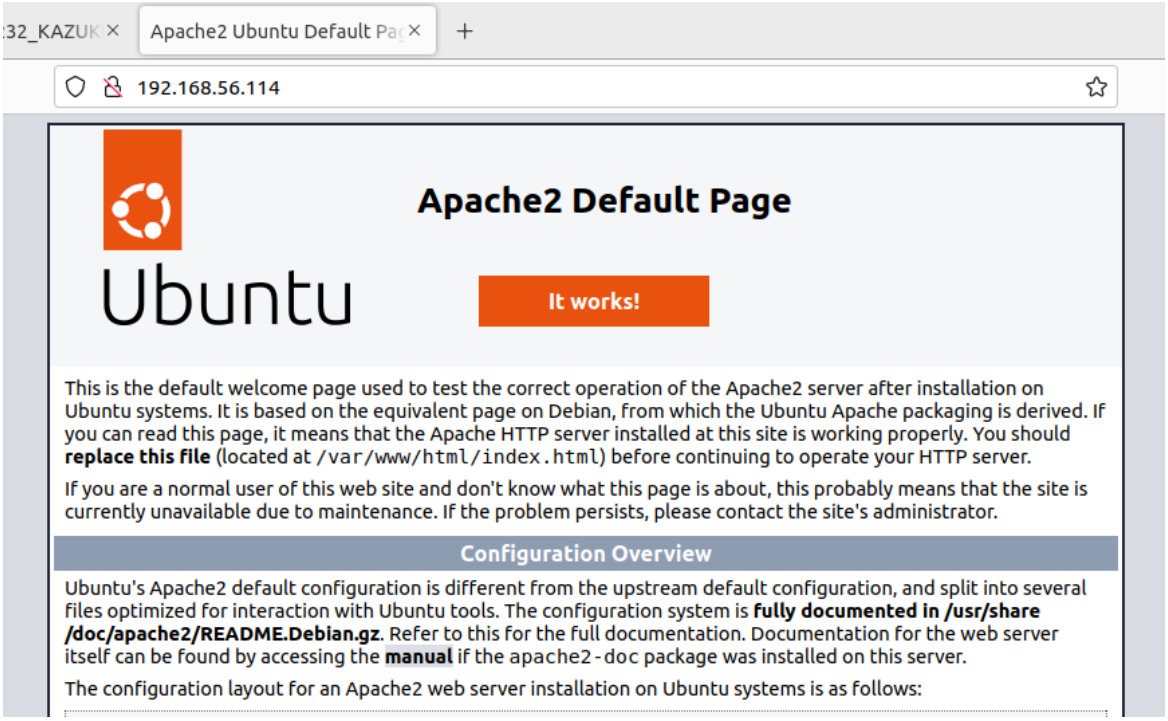

Figure 2.3 - Verifying Installed Apache2

My servers does not have a GUI so I don't know how to access any browsers there. I already tried the solutions I saw on the internet. So, I just use the browsers from my local machine using its own ip address.

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?

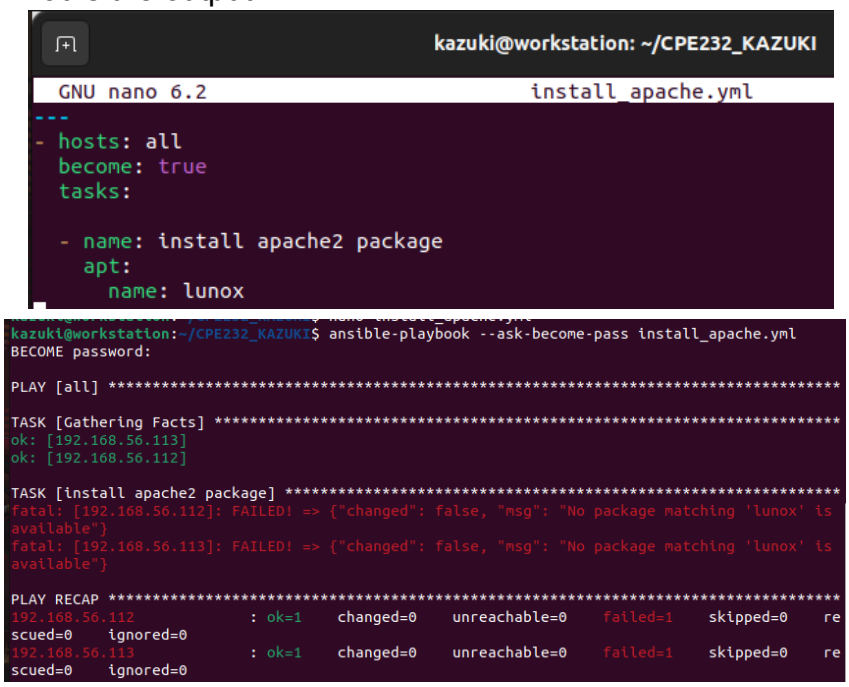

Figure 2.4 - Experimenting Playbook

The output failed because the name that I input does not exist in the repository of the package manager. It shows an error message saying that the package manager can't find any package with "lunox" name.

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.

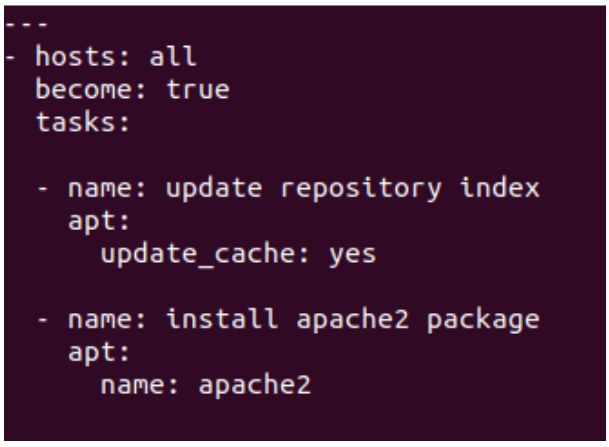

Figure 2.5 - Adding Additional Task in Playbook (Update Repository Index)

I just added an "update repository index" task in my playbook.

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

```
kazuki@workstation:~/CPE232_KAZUKI$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

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    skipped=0    re
scued=0    ignored=0
192.168.56.113             : ok=3    changed=1    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
```

Figure 2.6 - Running the Updated Playbook

It successfully did the tasks inside my playbook, it updated the repository index and installed apache2 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.

   Save the changes to this file and exit.

```
kazuki@workstation: ~/CPE232_KAZUKI
 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 - Adding Additional Task in Playbook (Adding PHP Support for Apache)

For adding PHP support for apache, we need the “libapache2-mod-php” package.

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

```
kazuki@workstation:~/CPE232_KAZUKI$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

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

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

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

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

PLAY RECAP *********************************************************************************
192.168.56.112             : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.113             : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 2.8 - Running the Updated Playbook

The output of the old task are still the same and the output of the new task which is adding PHP support for apache is also successful. The “ok” is the proof that the execution is successful and there is no error encountered.

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

https://github.com/kazzzuki/CPE232_KAZUKI.git

Reflections:
Answer the following:

1. What is the importance of using a playbook?
   - Based on my understanding in the discussion, the importance of Playbook is that it tells Ansible what to do/execute instead of manually doing a task. They are like a "to-do list" for Ansible since it contains a list of tasks and the Playbook contains the steps on how to/what to execute on remote machines.

2. Summarize what we have done on this activity.
   - In this activity, I run elevated ad hoc commands on remote servers, like updating cache packages, and installing VIM and snapd packages. And the most important part of this activity is writing my first Playbook. I created a Playbook file named installed_apache.yml, with the objective or purpose of automating the installation of Apache2 packages on remote servers.