Name: Julius Mark A. De Omampo	Date Performed: 09/11/24
Course/Section: CPE212 – CPE31S2	Date Submitted: 09/13/24
Instructor: Engr. Robin Valenzuela	Semester and SY:
Activity 4: Running Elevated Ad hoc Commands	

1. Objectives:

- 1.1 Use 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 multitier 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? No, it was not.

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.

```
julius-de-omampo@workstation:-$ ansible all -m apt -a update_cache=true --become
    --ask-become-pass
BECOME password:
server2 | CHANGED => {
        "ansible_facts": {
            "discovered_interpreter_python": "/usr/bin/python3"
      },
        "cache_update_time": 1726068045,
        "cache_updated": true,
        "changed": true
}
server1 | CHANGED => {
        "ansible_facts": {
            "discovered_interpreter_python": "/usr/bin/python3"
      },
        "cache_update_time": 1726068045,
        "cache_updated": true,
        "changed": true
}
julius-de-omampo@workstation:-$
```

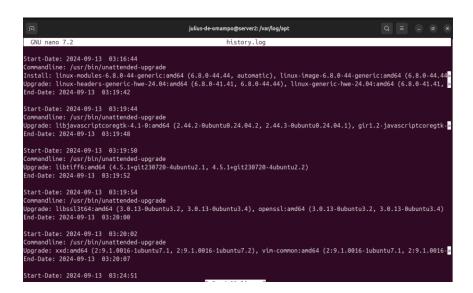
You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

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? Yes

```
julius-de-omampo@server1:~$ which vim
/usr/bin/vim
julius-de-omampo@server1:~$ apt search vim-vox
Sorting... Done
Full Text Search... Done
julius-de-omampo@server2:~$ apt search vim-vox
Sorting... Done
Full Text Search... Done
```

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.



- 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? It was a success and it added snapd packages through the respective servers

```
julius-de-onampo@workstation:-$ 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": 1726068045,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
},
    "cache_update_time": 1726068045,
    "cache_updated": false,
    "changed": false,
    "changed": false
}
julius-de-onampo@workstation:-$ |
```

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.

```
julius-de-omampo@workstation:-$ 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": 1726068045,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726068045,
    "cache_update_time": 1726068045,
    "cache_updated": false,
    "changed": false
}
julius-de-omampo@workstation:-$
```

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.

```
julius-de-omampo@workstation:~/Hands-on-Activity-4.1-Ansible-Basics$ nano install_apache.yml
julius-de-omampo@workstation:~/Hands-on-Activity-4.1-Ansible-Basics$ ls
install_apache.yml inventory.yaml README.md
```

When the editor appears, type the following:

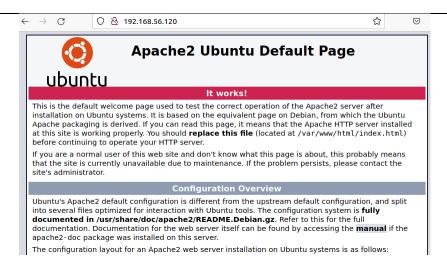
```
GNU nano 4.8 install_apache.yml
---
- hosts: all
become: true
tasks:
- name: install apache2 package
apt:
    name: apache2
```

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.

3.

4. 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.







5. 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? Failed

```
- hosts: all
become: true
tasks:
- name: install apache2 package
apt:
name: fake-apache2
```

6. 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.

```
- hosts: all
become: true
tasks:
- name: update repository index
apt:
    update_cache: yes
- name: install apache2 package
apt:
    name: apache2
```

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

```
julius-de-omampo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$ nano install_apache.yml
julius-de-omampo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all]

TASK [athering Facts]

ok: [server2]

ok: [server2]

ok: [server2]

changed: [server2]

changed: [server1]

TASK [install apache2 package]

ok: [server1]

TASK [install apache2 package]

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
julius-de-omampo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$
```

8. 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.

```
- 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
```

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

```
julius-de-onanpo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$ nano install_apache.yml
julius-de-onanpo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$ nano install_apache.yml
julius-de-onanpo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$ ansible-playbook --ask-become-pass install_apache.y
ml
BECOME password:

PLAY [all]

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

TASK [update repository index]
changed: [server2]

TASK [install apache2 package]
ok: [server2]

TASK [add PHP support for apache]
***Changed: [server2]
changed: [server1]
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
julius-de-onanpo@workstation:-/Hands-on-Activity-4.1-Ansible-Basics$
```

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

https://github.com/jmado-biscoff/Hands-on-Activity-4.1-Ansible-Basics.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?

A playbook in Ansible, or in fact in any tool, assists in ramping up automation and simplification by keeping the configuration consistent. Playbooks make reproduction easier and versioning possible. They can even provide detailed documentation of any deployments. They allow scalability with parallel server management and reduce errors since they specify an ideal state for the system, hence are vital to productivity and reliability in IT operations.

2. Summarize what we have done on this activity.

In this activity, we explored about the basics of using Ansible and utilizing them to some sample servers. First and foremost, we tried basic remote server updates via Ansible commands and it demonstrated states of success, failures, and changed and explained the status of which. Moreover, we also used playbook wherein we could utilize to process the command to Ansible for lesser error. Playbooks are important script files that is passed through remote servers to do certain tasks, elimination redundancy, human errors, server automation and promote efficient server maintenance. Overall, I have learned a lot about some basics in maintaining remote servers and I hope that I could apply these skills in real life server management.