Name: Agpaoa, Ma.Diane	**Date Performed: 09/06/2022**
Course/Section: CPE232–CPE31S22	**Date Submitted: 09/06/2022**
Instructor: Dr. Jonathan Taylar	**Semester and SY: 1st Sem 2022-2023**

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

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

```
madiane@workstation:~/cpe_ansible$ ansible all -m apt -a update_cache=true
192.168.56.101 | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock director
y /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - ope
n (13: Permission denied)"
}
192.168.56.103 | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock director
y /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - ope
n (13: Permission denied)"
}
```

Figure 1.1 Executing the command "*ansible all -m apt -a update_cache=true -The result of the command is not 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.

```
madiane@workstation:~/cpe_ansible$ ansible all -m apt -a update_cache=true --be
come --ask-become-pass
BECOME password:
192.168.56.103 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429150,
    "cache_updated": true,
    "changed": true
}
192.168.56.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429177,
    "cache_updated": true,
    "changed": true
}
```

Figure 1.2. Executing the command "*ansible all -m apt -a update_cache=true --become --ask-become-pass*"

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.

```
madiane@workstation:~/cpe_ansible$ ansible all -m apt -a name=vim-nox --become
--ask-become-pass
BECOME password:
192.168.56.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429177,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading s
tate information...\nThe following additional packages will be installed:\n  fo
nts-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n  rub
y-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n  rubygems-integra
tion vim-runtime\nSuggested packages:\n  apache2 | lighttpd | httpd ri ruby-dev
 bundler cscope vim-doc\nThe following NEW packages will be installed:\n  fonts
-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n  ruby-n
et-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n  rubygems-integratio
n vim-nox vim-runtime\n0 upgraded, 15 newly installed, 0 to remove and 38 not u
pgraded.\nNeed to get 17.5 MB of archives.\nAfter this operation, 76.3 MB of ad
ditional disk space will be used.\nGet:1 http://ph.archive.ubuntu.com/ubuntu ja
mmy/main amd64 fonts-lato all 2.0-2.1 [2696 kB]\nGet:2 http://ph.archive.ubuntu
.com/ubuntu jammy/main amd64 javascript-common all 11+nmu1 [5936 B]\nGet:3 http
buntu.com/ubuntu jammy/main amd64 libjs-jquery all 3.6.0+dfsg+~3
\nIgn:3 http://ph.archive.ubuntu.com/ubuntu jammy/main amd64 li
bjs-jquery all 3.6.0+dfsg+~3.5.13-1\nGet:4 http://ph.archive.ubuntu.com/ubuntu
```

Show Applications

Figure 1.3. Installation of VIM

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?

```
madiane@workstation:~/cpe_ansible$ which vim
madiane@workstation:~/cpe_ansible$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy 2:8.2.3995-1ubuntu2 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version
```

Figure 1.4. Verification of the installed package

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.

```
madiane@workstation:~/cpe_ansible$ cd /var/log
madiane@workstation:/var/log$ ls
alternatives.log    dmesg.0          kern.log
alternatives.log.1  dmesg.1.gz       kern.log.1
apport.log          dmesg.2.gz       lastlog
apt                 dmesg.3.gz       openvpn
auth.log            dmesg.4.gz       private
auth.log.1          dpkg.log         speech-dispatcher
boot.log            dpkg.log.1       syslog
boot.log.1          faillog          syslog.1
bootstrap.log       fontconfig.log   ubuntu-advantage.log
btmp                gdm3             ubuntu-advantage-timer.log
btmp.1              gpu-manager.log  ubuntu-advantage-timer.log.1
cups                hp               ufw.log
dist-upgrade        installer        unattended-upgrades
dmesg               journal          wtmp
madiane@workstation:/var/log$ cd apt
madiane@workstation:/var/log/apt$ ls
eipp.log.xz  history.log  history.log.1.gz  term.log  term.log.1.gz
madiane@workstation:/var/log/apt$ cat history.log
```

Figure 1.5. Executing the commands "*cd /var/log*", "*ls*", "*cd apt*"

```
  GNU nano 6.2                        history.log

Start-Date: 2022-09-06  08:38:47
Commandline: apt install ansible --fix-missing
Requested-By: madiane (1000)
Install: python-babel-localedata:amd64 (2.8.0+dfsg.1-7, automatic), python3-dn>
End-Date: 2022-09-06  08:39:08

Start-Date: 2022-09-06  08:40:39
Commandline: apt install ansible --fix-missing
Requested-By: madiane (1000)
Install: ansible:amd64 (2.10.7+merged+base+2.10.8+dfsg-1)
End-Date: 2022-09-06  08:41:48

Start-Date: 2022-09-06  08:47:45
Commandline: apt install python3-pip
Requested-By: madiane (1000)
Install: libalgorithm-merge-perl:amd64 (0.08-3, automatic), manpages-dev:amd64>
End-Date: 2022-09-06  08:49:04

                    [ File 'history.log' is unwritable ]
^G Help      ^O Write Out   ^W Where Is    ^K Cut       ^T Execute
^X Exit      ^R Read File   ^\ Replace     ^U Paste     ^J Justify
```

Figure 1.6. Informations in history.log

-The history.log contains the information about the logs in the server, such as the start and end date, command line, who requested the command, and what packages were installed.

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?

```
madiane@workstation:~/cpe_ansible$ ansible all -m apt -a name=snapd --become --
ask-become-pass
BECOME password:
192.168.56.101 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429177,
    "cache_updated": false,
    "changed": false
}
192.168.56.103 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429150,
    "cache_updated": false,
    "changed": false
}
madiane@workstation:~/cpe_ansible$
```

Figure 1.7. Installation of snapd

-Since snapd is pre-installed in the Ubuntu system, this command will check for the latest installation package. Based on the result, the command is successful and it didn't change anything in the remote 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.

```
madiane@workstation:~/cpe_ansible$ ansible all -m apt -a "name=snapd state=late
st" --become --ask-become-pass
BECOME password:
192.168.56.101 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429177,
    "cache_updated": false,
    "changed": false
}
192.168.56.103 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662429150,
    "cache_updated": false,
    "changed": false
}
madiane@workstation:~/cpe_ansible$
```

Figure 1.8. Issuing the command "*ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*"

-The output of this command is similar to the previous command "*ansible all -m apt -a name=snapd --become --ask-become-pass*" that we issued.

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:

```
  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.

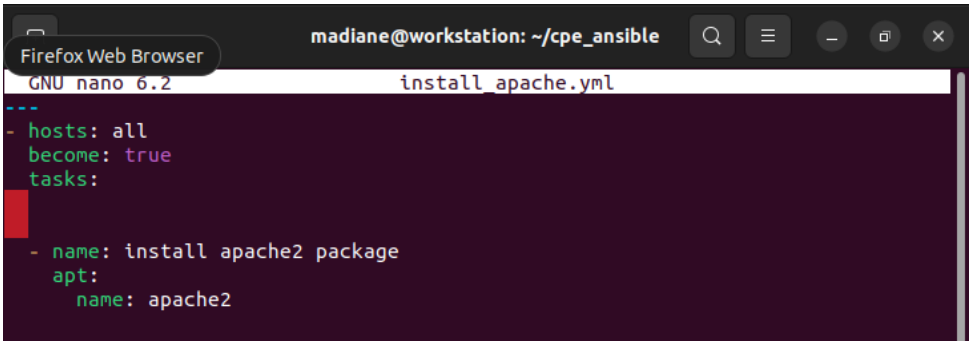


Figure 2.1. Creating a playbook

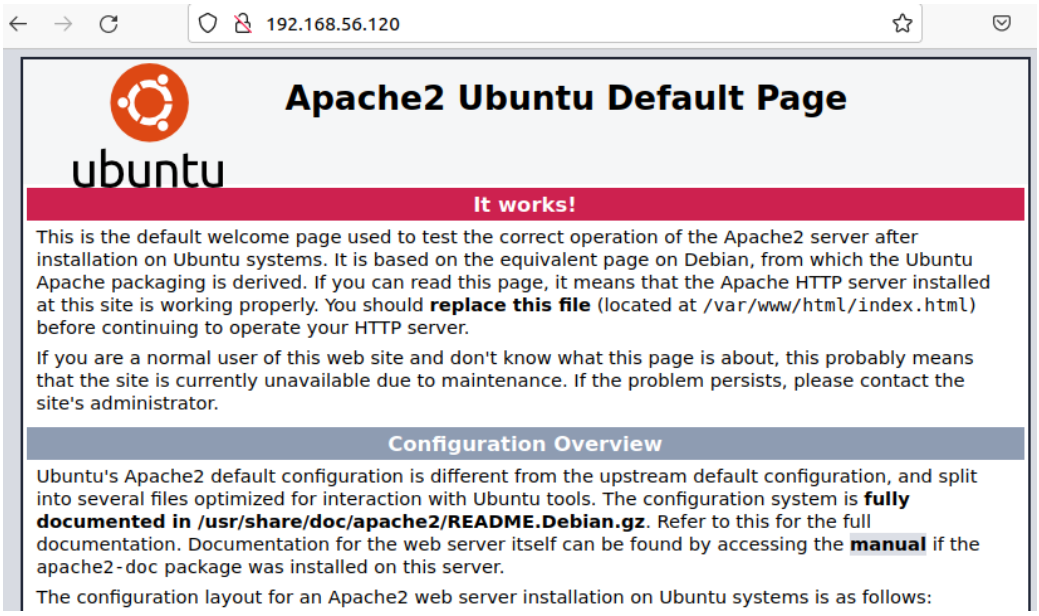
2. Run the yml file using the command: *ansible-playbook --ask-become-pass install_apache.yml.* Describe the result of this command.

```
madiane@workstation:~/cpe_ansible$ ansible-playbook --ask-become-pass install_a
pache.yml
BECOME password:

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

TASK [Gathering Facts] ***********************************************************
*
ok: [192.168.56.101]
ok: [192.168.56.103]

TASK [install apache2 package] ***************************************************
*
changed: [192.168.56.101]
changed: [192.168.56.103]

PLAY RECAP ***********************************************************************
*
192.168.56.101             : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.56.103             : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

madiane@workstation:~/cpe_ansible$
```

Figure 2.2. Running the yml file

-The result of this command will install the apache2 package to all the managed nodes or remote servers.

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.



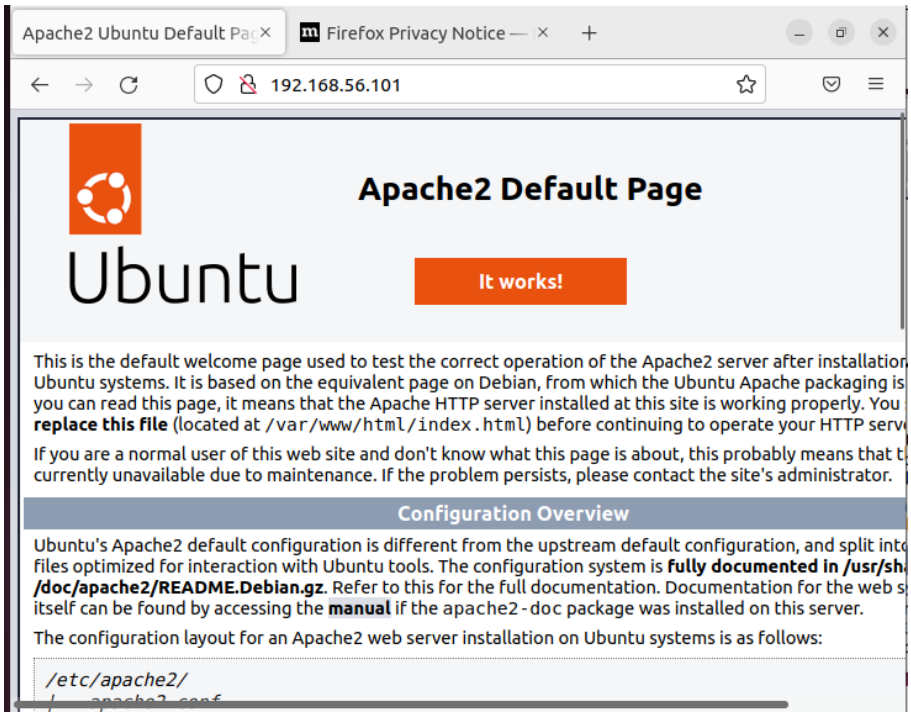


Figure 2.3. Verification that apache2 was installed in remote Server 1

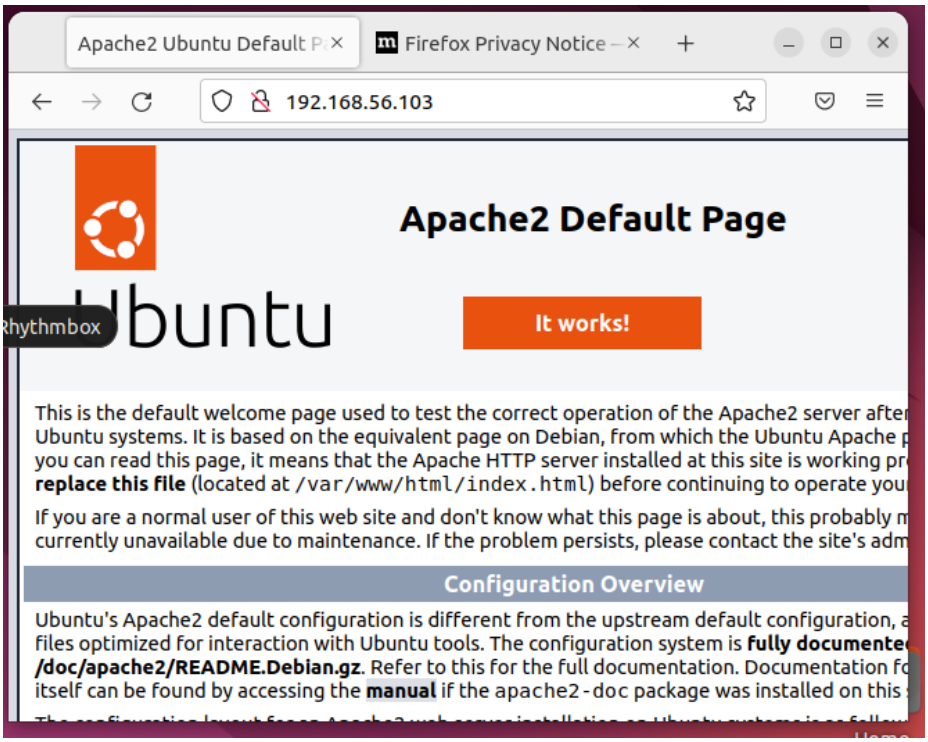


Figure 2.4. Verification that apache2 was installed in remote Server 1

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?

```
  GNU nano 6.2                    install_apache.yml
---
- hosts: all
  become: true
  tasks:


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

Figure 2.5. Changed the name of the package to an unrecognizable name

```
madiane@workstation:~/cpe_ansible$ ansible-playbook --ask-become-pass install_a
pache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.101]
ok: [192.168.56.103]

TASK [install apache2 package] *************************************************
*
fatal: [192.168.56.101]: FAILED! => {"changed": false, "msg": "No package match
ing 'bsiuodf' is available"}
fatal: [192.168.56.103]: FAILED! => {"changed": false, "msg": "No package match
ing 'bsiuodf' is available"}

PLAY RECAP *********************************************************************
*
192.168.56.101             : ok=1    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
192.168.56.103             : ok=1    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0

madiane@workstation:~/cpe_ansible$
```

Figure 2.6 Output of the install_apache.yml

-The task of gathering facts will have an ok state and the task of installing the apache2 with an unrecognizable package will fail because there is no package that matches the unrecognizable 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.

```
---
- 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.

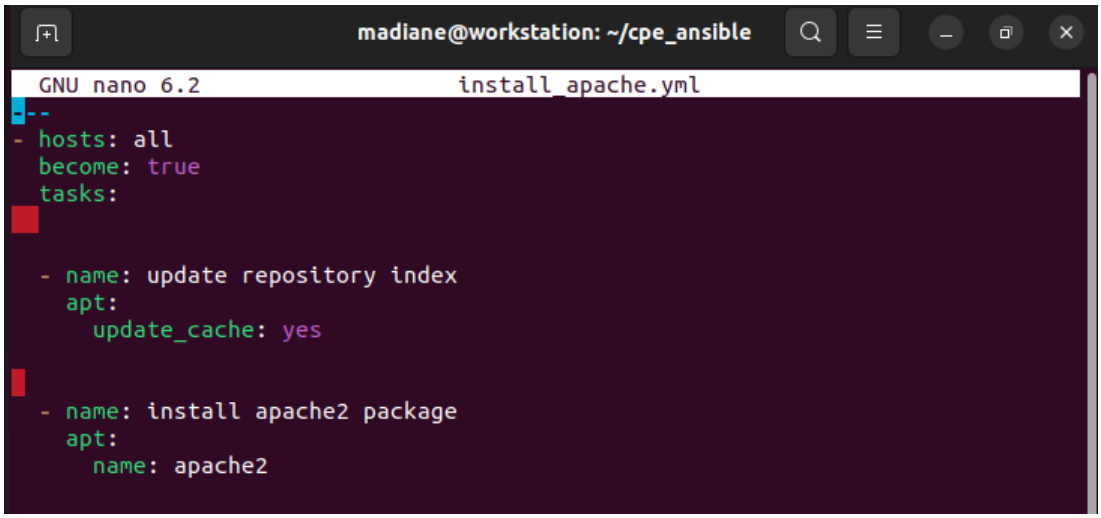


Figure 2.7. Adding the additional command "*update_cache*"

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

```
madiane@workstation:~/cpe_ansible$ ansible-playbook --ask-become-pass install_a
pache.yml
BECOME password:

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

TASK [Gathering Facts] ********************************************************
*
ok: [192.168.56.103]
ok: [192.168.56.101]

TASK [update repository index] ************************************************
*
changed: [192.168.56.101]
changed: [192.168.56.103]

TASK [install apache2 package] ************************************************
*
ok: [192.168.56.101]
ok: [192.168.56.103]

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

Figure 2.8. Running the playbook install_apache.yml

-Based on the output, the new command did changes on the remote servers, this change is from the task that updates the repository index.

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.

```
madiane@workstation: ~/cpe_ansible
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 2.9. Adding the PHP support for the apache package

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

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

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.103]
ok: [192.168.56.101]

TASK [update repository index] *************************************************
changed: [192.168.56.101]
changed: [192.168.56.103]

TASK [install apache2 package] *************************************************
ok: [192.168.56.101]
ok: [192.168.56.103]

TASK [add PHP support for apache] **********************************************
changed: [192.168.56.101]
changed: [192.168.56.103]

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

Figure 2.10. Running the playbook install_apache.yml

-Based on the output, the new command did changes on the remote servers, this change is specifically from the task “add PHP support for apache”.

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

   Link of the GitHub repository:

   https://github.com/qmja/CPE232_Agpaoa-Ma.Diane.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?

   The importance of the playbook is it can define the state we desire for all the remote servers we are managing. In addition, the conditions, variables and task in the playbook can be saved which means that we can share and use the playbook we created again.

2. Summarize what we have done on this activity.

   To summarize, what we have done on this activity is use the commands that make changes to remote machines. In this activity, we learn ad hoc commands that would install, update and upgrade packages in the remote machines. In addition, in this activity we also created playbooks that record and execute ansible's configuration, deployment and orchestration functions.