

Name: Emmanuel Luis B. Cuyugan	Date Performed: 02/12/2024
Course/Section: CPE 018-CPE31S1	Date Submitted: 02/20/2024
Instructor: DR. JONATHAN V. TAYLAR	Semester and SY: 2nd 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: <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. Issue the following command:

ansible all -m apt -a update_cache=true

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

```
emncuygn@server2:~$ ansible all -m apt update_cache=true
Usage: ansible <host-pattern> [options]

Define and run a single task 'playbook' against a set of hosts

Options:
  -a MODULE_ARGS, --args=MODULE_ARGS
                        module arguments
  --ask-vault-pass      ask for vault password
  -B SECONDS, --background=SECONDS
                        run asynchronously, failing after X seconds
                        (default=N/A)
  -C, --check           don't make any changes; instead, try to predict some
                        of the changes that may occur
  -D, --diff           when changing (small) files and templates, show the
                        differences in those files; works great with --check
  -e EXTRA_VARS, --extra-vars=EXTRA_VARS
                        set additional variables as key=value or YAML/JSON, if
                        filename prepend with @
  -f FORKS, --forks=FORKS
                        specify number of parallel processes to use
                        (default=5)
  -h, --help           show this help message and exit
  -i INVENTORY, --inventory=INVENTORY, --inventory-file=INVENTORY
                        specify inventory host path or comma separated host
                        list. --inventory-file is deprecated
  -l SUBSET, --limit=SUBSET
                        further limit selected hosts to an additional pattern
  --list-hosts         outputs a list of matching hosts: does not execute
```

```
Privilege Escalation Options:
  control how and which user you become as on target hosts

  -s, --sudo           run operations with sudo (nopasswd) (deprecated, use
                        become)
  -U SUDO_USER, --sudo-user=SUDO_USER
                        desired sudo user (default=root) (deprecated, use
                        become)
  -S, --su            run operations with su (deprecated, use become)
  -R SU_USER, --su-user=SU_USER
                        run operations with su as this user (default=None)
                        (deprecated, use become)
  -b, --become        run operations with become (does not imply password
                        prompting)
  --become-method=BECOME_METHOD
                        privilege escalation method to use (default=sudo),
                        valid choices: [ sudo | su | pbrun | pfexec | doas |
                        dzdo | ksu | runas | pmsu | pmrun | enable ]
  --become-user=BECOME_USER
                        run operations as this user (default=root)
  --ask-sudo-pass      ask for sudo password (deprecated, use become)
  --ask-su-pass       ask for su password (deprecated, use become)
  -K, --ask-become-pass
                        ask for privilege escalation password
```

Some modules do not make sense in Ad-Hoc (include, meta, etc)

ERROR! Extraneous options or arguments

```
emncuygn@server2:~$
```

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.

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?
 - 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`.

```

emncuygn@workstation:~$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
SUDO password:
192.168.56.110 | FAILED! => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false,
  "msg": "'/usr/bin/apt-get -y -o \"Dpkg::Options::=--force-confdef\" -o \"Dpkg::Options::=--force-confold\" --install 'vim-nox' failed: E: Could not get lock /var/lib/dpkg/lock-frontent - open (11: Resource temporarily unavailable)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontent), is another process using it?\n",
  "rc": 100,
  "stderr": "E: Could not get lock /var/lib/dpkg/lock-frontent - open (11: Resource temporarily unavailable)\nE: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontent), is another process using it?\n",
  "stderr_lines": [
    "E: Could not get lock /var/lib/dpkg/lock-frontent - open (11: Resource temporarily unavailable)",
    "E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontent), is another process using it?"
  ],
  "stdout": "",
  "stdout_lines": []
}
192.168.56.111 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}

emncuygn@workstation:~$ ansible all -m apt --become --ask-become-pass
SUDO password:
192.168.56.111 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}
192.168.56.110 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}

```

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?

```
emncuygn@workstation:~$ ansible all -m apt -a name=snapd --become --ask-become-pass
SUDO password:
192.168.56.111 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}
192.168.56.110 | SUCCESS => {
  "cache_update_time": 1708422093,
  "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 placed them in double quotations.

```
emncuygn@workstation:~$ ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
SUDO password:
192.168.56.111 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}
192.168.56.110 | SUCCESS => {
  "cache_update_time": 1708422093,
  "cache_updated": false,
  "changed": false
}
```

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

```
emncuygn@workstation:~/CPE232_CuyuganEmmanuel$ git push origin main
Everything up-to-date
```

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.

```
emncuygn@workstation: ~/CPE232_CuyuganEmmanuel
File Edit View Search Terminal Help
GNU nano 2.9.3                                install_apache.yml
--
- hosts: 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.

```
emncuygn@workstation: ~/CPE232_CuyuganEmmanuel$ ansible-playbook --ask-become-pass install_apache.yml
SUDO password:

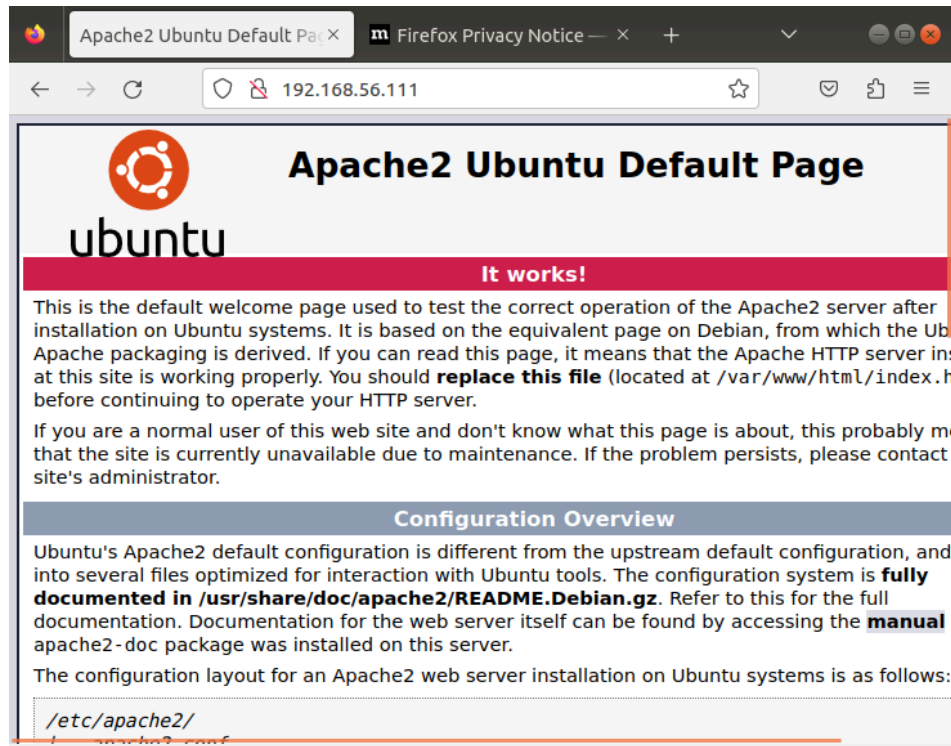
PLAY [all] *****
*

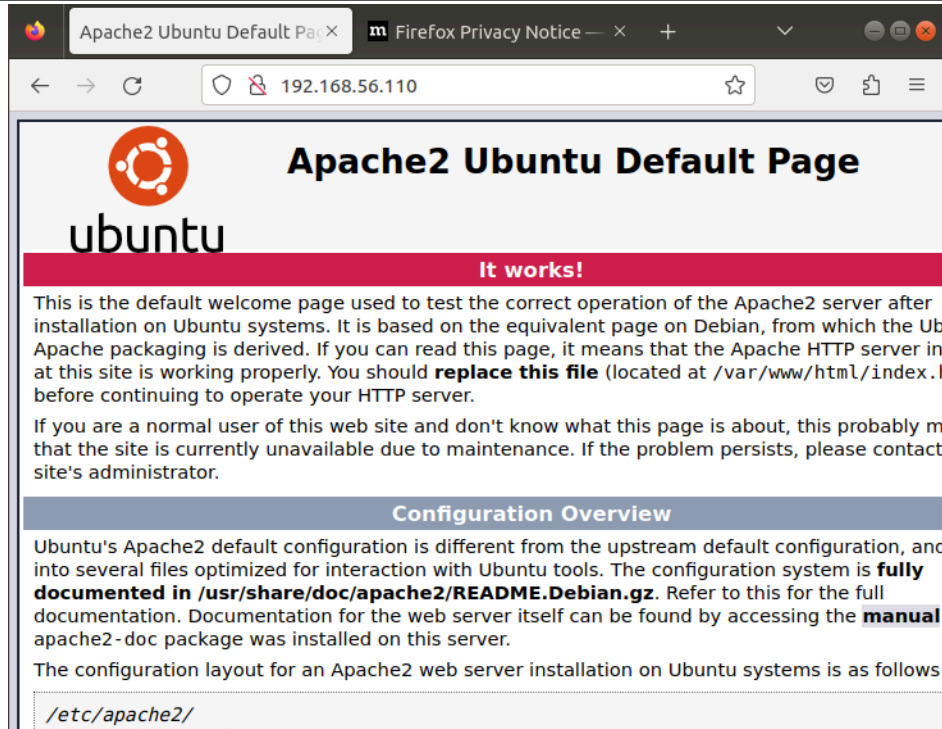
TASK [Gathering Facts] *****
*
ok: [192.168.56.111]
ok: [192.168.56.110]

TASK [install apache2 package] *****
*
changed: [192.168.56.110]
changed: [192.168.56.111]

PLAY RECAP *****
*
192.168.56.110           : ok=2    changed=1    unreachable=0    failed=0
192.168.56.111           : ok=2    changed=1    unreachable=0    failed=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?
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.


```
emncuygn@workstation: ~/CPE232_CuyuganEmmanuel
File Edit View Search Terminal Help
GNU nano 2.9.3      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?

```
emncuygn@workstation:~/CPE232_CuyuganEmmanuel$ ansible-playbook --ask-become-pass install_apache.yml
SUDO password:

PLAY [all] *****
*

TASK [Gathering Facts] *****
*
ok: [192.168.56.111]
ok: [192.168.56.110]

TASK [update repository index] *****
*
changed: [192.168.56.110]
changed: [192.168.56.111]

TASK [install apache2 package] *****
*
ok: [192.168.56.110]
ok: [192.168.56.111]

PLAY RECAP *****
*
192.168.56.110      : ok=3    changed=1    unreachable=0    failed=0
192.168.56.111      : ok=3    changed=1    unreachable=0    failed=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.

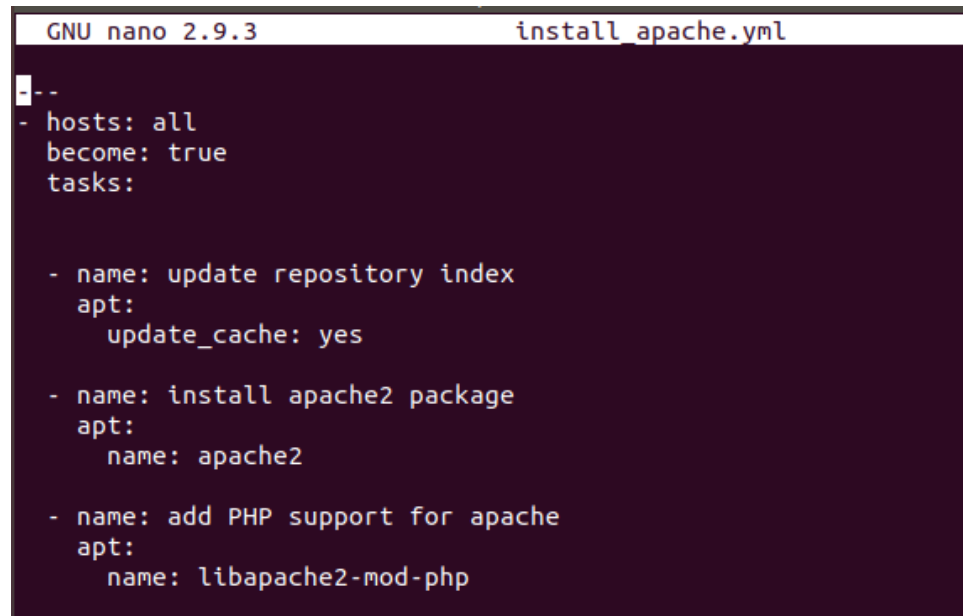
```
---
- hosts: all
  become: true
  tasks:

    - 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.



```
GNU nano 2.9.3      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?

```

SUDO password:
PLAY [all] *****
*

TASK [Gathering Facts] *****
*
ok: [192.168.56.111]
ok: [192.168.56.110]

TASK [update repository index] *****
*
changed: [192.168.56.110]
changed: [192.168.56.111]

TASK [install apache2 package] *****
*
ok: [192.168.56.110]
ok: [192.168.56.111]

TASK [add PHP support for apache] *****
*
changed: [192.168.56.111]
changed: [192.168.56.110]

PLAY RECAP *****
192.168.56.110      : ok=4    changed=2    unreachable=0    failed=0
192.168.56.111      : ok=4    changed=2    unreachable=0    failed=0

```

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

```

emmancuygn@workstation:~/CPE232_CuyuganEmmanuel$ git add install_apache.yml
emmancuygn@workstation:~/CPE232_CuyuganEmmanuel$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   install_apache.yml

emmancuygn@workstation:~/CPE232_CuyuganEmmanuel$ git push origin main
Everything up-to-date
emmancuygn@workstation:~/CPE232_CuyuganEmmanuel$ git commit -m "First Playbook!"
[main a999a80] First Playbook!
 1 file changed, 17 insertions(+)
 create mode 100644 install_apache.yml
emmancuygn@workstation:~/CPE232_CuyuganEmmanuel$ git push origin main
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:emmancuyugan/CPE232_CuyuganEmmanuel.git
 99a9718..a999a80  main -> main

```

[GitHub - emmancuyugan/CPE232_CuyuganEmmanuel](https://github.com/emmancuyugan/CPE232_CuyuganEmmanuel)



Reflections:

Answer the following:

1. What is the importance of using a playbook?

Playbooks tell Ansible what to do on which devices.

2. Summarize what we have done on this activity.

We learned how to install Apache and use ansible. With Ansible, we made Playbooks for easier access of other hosts.