Name: Jose Mari Tan Dela Peña	Date Performed: 09/16/2024
Course/Section: CPE31S2	Date Submitted: 09/16/2024
Instructor: Engr. Robin Valenzuela	Semester and SY: 1st Sem, 2024 - 2025

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?

```
liglig@workstation:~/t1$ ansible all -m apt -a update_cache=true
192.168.56.100 | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock directory
 /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)"
}
192.168.56.102 | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "msg": "Failed to lock apt for exclusive operation: Failed to lock directory
 /var/lib/apt/lists/: E:Could not open lock file /var/lib/apt/lists/lock - open
(13: Permission denied)"
}
```

It was 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.

```
liglig@workstation:~/t1$ ansible all -m apt -a update_cache=true --become --ask-
become-pass
BECOME password:
192.168.56.100 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447469,
    "cache_updated": true,
    "changed": true
}
192.168.56.102 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447470,
    "cache_updated": true,
    "changed": true
}
liglig@workstation:~/t1$
```

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?

```
liglig@workstation:~/t1$ which vim
liglig@workstation:~/t1$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/noble-updates,noble-security 2:9.1.0016-1ubuntu7.2 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/noble-updates,noble-security,now 2:9.1.0016-1ubuntu7.2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

liglig@workstation:~/t1$
```

Yes, it was 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.

```
liglig@workstation:/var/log/apt$ cat history.log

Start-Date: 2024-04-24  10:47:34
Commandline: apt-get -y --fix-policy install
Install: libgpg-error-l10n:amd64 (1.47-3build2, automatic), e2fsprogs-l10n:amd64 (1.47.0-2.4~exp1ubuntu4, automatic), libgpm2:amd64 (1.20.7-11, automatic), psmisc:amd64 (23.7-1
build1, automatic), uuid-runtime:amd64 (2.39.3-9ubuntu6, automatic), bash-completion:amd64 (1:2.11-8, automatic), bsdextrautils:amd64 (2.39.3-9ubuntu6, automatic)
End-Date: 2024-04-24  10:47:35

Start-Date: 2024-04-24  10:47:49
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes install accountsservice acl adduser adwaita-icon-theme alsa-base alsa-topology-conf alsa-ucm-conf alsa-utils an
acron apg apparmor apport apport-core-dump-handler apport-gtk apport-symptoms appstream apt apt-config-icons apt-config-icons-hidpi apt-utils aptdaemon aptdaemon-data aspell as
pell-en at-spi2-common at-spi2-core avahi-daemon baobab base-files base-passwd bash bash-completion bc bind9-dnsutils bind9-host bind9-libs bluez bluez-cups bluez-obexd bolt br
ltty bsdextrautils bsdutils bubblewrap busybox-initramfs busybox-static ca-certificates cloud-guest-utils cloud-init colord colord-data command-not-found console-setup console-
setup-linux coreutils cpio cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu cracklib-runtime cron cron-daemon-common cups cups-browsed cups-bsd cups-client cups-common c
ups-core-drivers cups-daemon cups-filters cups-filters-core-drivers cups-ipp-utils cups-pk-helper cups-ppdc cups-server-common dash dbus dbus-bin dbus-daemon dbus-session-bus-c
ommon dbus-system-bus-common dbus-user-session dc dconf-cli dconf-gsettings-backend dconf-service debconf debconf-i18n debianutils desktop-file-utils dhcpcd-base dictionaries-c
ommon diffutils dirmngr distro-info distro-info-data dmidecode dmsetup dmz-cursor-theme dns-root-data dnsmasq-base docbook-xml dosfstools dpkg dracut-install e2fsprogs eatmydat
a ed eject emacsen-common enchant-2 eog espeak-ng-data ethtool evince evince-common evolution-data-server evolution-data-server-common fdisk file findutils firefox fontconfig f
ontconfig-config fonts-arphic-ukai fonts-arphic-uming fonts-dejavu-core fonts-dejavu-mono fonts-droid-fallback fonts-liberation fonts-liberation-sans-narrow fonts-noto-cjk font
s-noto-color-emoji fonts-noto-core fonts-noto-mono fonts-ubuntu fonts-urw-base35 foomatic-db-compressed-ppds fprintd friendly-recovery ftp fuse3 fwupd fwupd-signed gamemode gam
emode-daemon gcc-13-base gcc-14-base gcr gcr4 gdb gdisk gdm3 geoclue-2.0 geocode-glib-common gettext-base ghostscript gir1.2-accountsservice-1.0 gir1.2-adw-1 gir1.2-atk-1.0 gir
1.2-atspi-2.0 gir1.2-freedesktop gir1.2-gck-2 gir1.2-gcr-4 gir1.2-gdesktopenums-3.0 gir1.2-gdkpixbuf-2.0 gir1.2-gdm-1.0 gir1.2-geoclue-2.0 gir1.2-girepository-2.0 gir1.2-glib-2
.0 gir1.2-gmenu-3.0 gir1.2-gnomeautoar-0.1 gir1.2-gnomebg-4.0 gir1.2-gnomebluetooth-3.0 gir1.2-gnomedesktop-3.0 gir1.2-gnomedesktop-4.0 gir1.2-graphene-1.0 gir1.2-gstreamer-1.0
 gir1.2-gtk-3.0 gir1.2-gtk-4.0 gir1.2-gweather-4.0 gir1.2-handy-1 gir1.2-harfbuzz-0.0 gir1.2-ibus-1.0 gir1.2-javascriptcoregtk-4.1 gir1.2-javascriptcoregtk-6.0 gir1.2-mutter-14
 gir1.2-nm-1.0 gir1.2-nma4-1.0 gir1.2-notify-0.7 gir1.2-packagekitglib-1.0 gir1.2-pango-1.0 gir1.2-peas-1.0 gir1.2-polkit-1.0 gir1.2-rsvg-2.0 gir1.2-secret-1 gir1.2-snapd-2 gir
```

Basically, it is all the installation and other processes that I have done in my Ubuntu workstation.

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?

```
liglig@workstation:~/t1$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447470,
    "cache_updated": false,
    "changed": false
}
192.168.56.100 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447469,
    "cache_updated": false,
    "changed": false
}
liglig@workstation:~/t1$
```

It successfully installed SNAP inside my 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.

```
liglig@workstation:~/t1$ ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447470,
    "cache_updated": false,
    "changed": false
}
192.168.56.100 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1726447469,
    "cache_updated": false,
    "changed": false
}
liglig@workstation:~/t1$
```

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
```

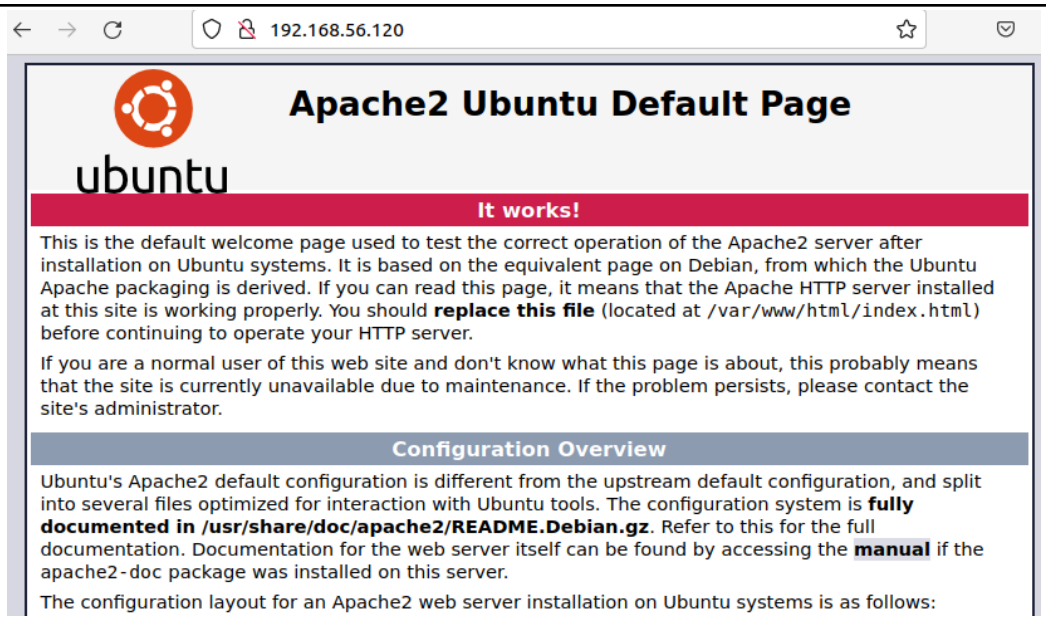
Make sure to save the file. Take note also of the alignments of the texts.

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

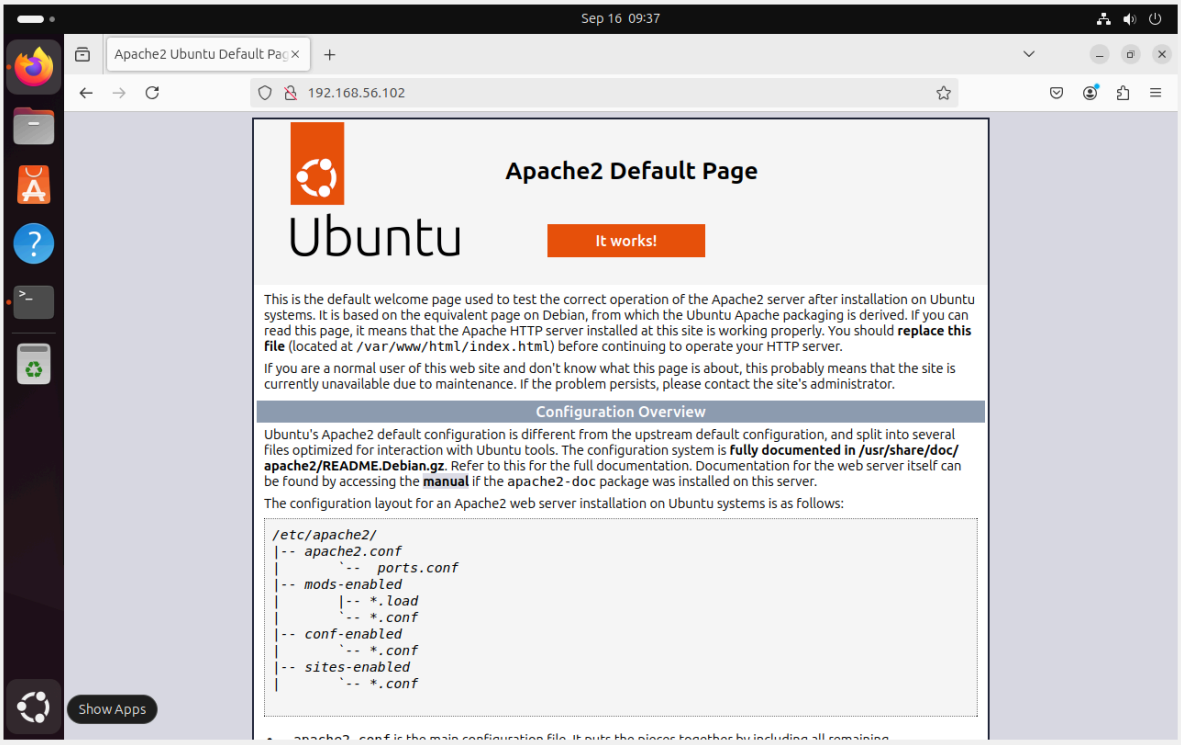
```
liglig@workstation:~/CPE212Act4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.100]

TASK [install apache2 package] *************************************************************************************
changed: [192.168.56.100]
changed: [192.168.56.102]

PLAY RECAP *********************************************************************************************************
192.168.56.100             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.102             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

It installed apache yml to my 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.



Mine:



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?

```
liglig@workstation:~/CPE212Act4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.100]

TASK [install apache2 package] *************************************************************************************
fatal: [192.168.56.100]: FAILED! => {"changed": false, "msg": "No package matching 'hotdog' is available"}
fatal: [192.168.56.102]: FAILED! => {"changed": false, "msg": "No package matching 'hotdog' is available"}

PLAY RECAP *********************************************************************************************************
192.168.56.100             : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
192.168.56.102             : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

It failed.

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.

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

```
liglig@workstation:~/CPE212Act4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.100]

TASK [update repository index] *************************************************************************************
changed: [192.168.56.100]
changed: [192.168.56.102]

TASK [install apache2 package] *************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.100]

PLAY RECAP *********************************************************************************************************
192.168.56.100             : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.102             : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

liglig@workstation:~/CPE212Act4$
```

The update repository index was integrated to the yml playbook.

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
```

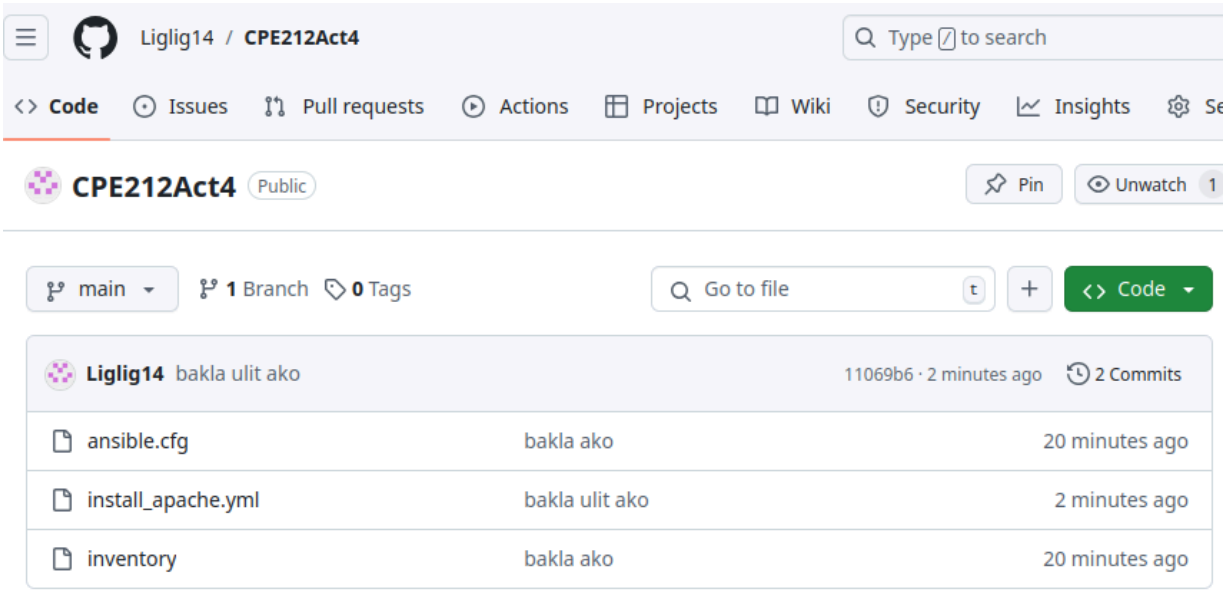
Save the changes to this file and exit.

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

```
liglig@workstation:~/CPE212Act4$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] ***********************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.100]

TASK [update repository index] ***************************************************************************************
changed: [192.168.56.102]
changed: [192.168.56.100]

TASK [install apache2 package] ***************************************************************************************
ok: [192.168.56.100]
ok: [192.168.56.102]

TASK [add PHP support for apache] ************************************************************************************
changed: [192.168.56.100]
changed: [192.168.56.102]

PLAY RECAP ***********************************************************************************************************
192.168.56.100             : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.56.102             : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

liglig@workstation:~/CPE212Act4$
```

The playbook is now completed and established with all the necessary functions.

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



Reflections:

Answer the following:

1. What is the importance of using a playbook?
   - Using a playbook in Ubuntu promotes efficiency, consistency across systems and makes the repetitive tasks automated. It also serves as documentation, and it allows version control, that helps system management to be more efficient and reliable that lessens human error.

2. Summarize what we have done on this activity.

    - In this exercise, we learned how to effectively manage remote servers with Ansible. After making sure we had the required permissions, we first proceeded by running commands to install software and update package information. After that, we did the specific processes to install Apache, which we checked with our web browser. We also added support for more software components and created a playbook with our necessary features. We used GitHub version control to keep our modifications up to date, along with the process, and all these show how Ansible may simplify server administration.