Name: Tendencia, Jasmin Raiza S.	**Date Performed:** 09/11/2023
Course/Section: CPE31S4	**Date Submitted:** 09/12/2023
Instructor: Dr. Jonathan Taylar	**Semester and SY:** 1st - 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:

ansible all -m apt -a update_cache=true

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

```
tendencia@workstation:~/CPE232_JasminTendencia$ ansible all -m apt -a update_cac
he=true
192.168.56.103 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168.
56.103 port 22: No route to host",
    "unreachable": true
}
192.168.56.102 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: connect to host 192.168.
56.102 port 22: No route to host",
    "unreachable": true
}
```

No, since the command is incomplete, it only displays either "“failed" or "unreachable" which means that it has not been run 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.

```
tendencia@workstation:~/CPE232_JasmInTendencia$ ansible all -m apt -a update_cache=true --become --ask-become-pass
BECOME password:
192.168.56.102 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1694505875,
    "cache_updated": true,
    "changed": true
}
192.168.56.103 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1694505879,
    "cache_updated": true,
    "changed": true
}
```

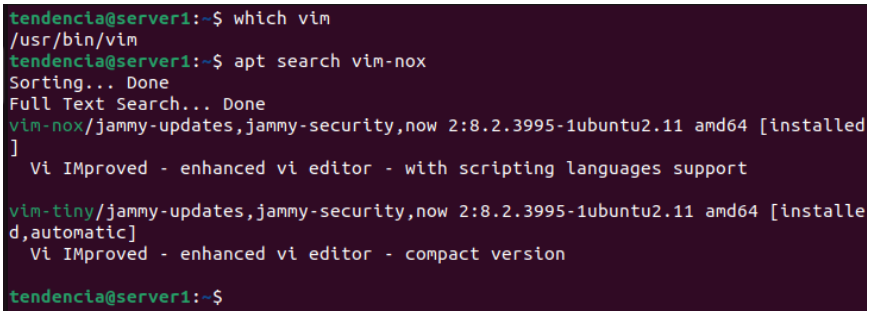
The cache of the servers has been successfully updated (indicated by the CHANGED) with the use of the modified command.

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

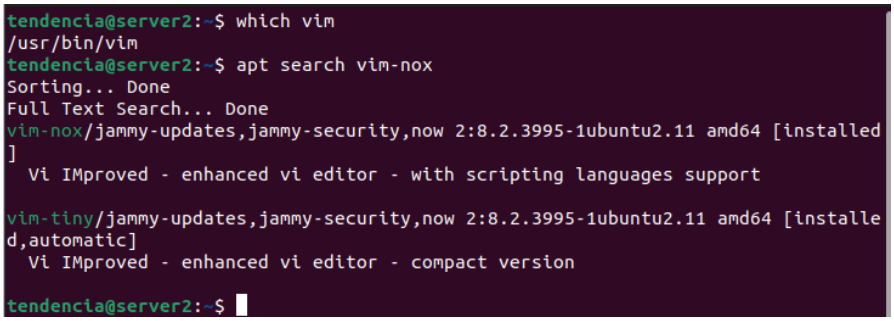
password because the local machine instructed the remote servers to actually install the package.

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?

Server 1:



Server 2:



The command for both servers were 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.

```
tendencia@server1:~$ cd /var/log
tendencia@server1:/var/log$ ls
alternatives.log    cups          faillog          openvpn
alternatives.log.1  dist-upgrade  fontconfig.log   private
apt                 dmesg         gdm3             speech-dispatcher
auth.log            dmesg.0       gpu-manager.log  syslog
auth.log.1          dmesg.1.gz    hp               syslog.1
boot.log            dmesg.2.gz    installer        ubuntu-advantage.log
boot.log.1          dmesg.3.gz    journal          ubuntu-advantage.log.1
bootstrap.log       dmesg.4.gz    kern.log         unattended-upgrades
btmp                dpkg.log      kern.log.1       wtmp
btmp.1              dpkg.log.1    lastlog
tendencia@server1:/var/log$ cd apt
tendencia@server1:/var/log/apt$ sudo nano history.log
[sudo] password for tendencia:
```

```
  GNU nano 6.2                          history.log

Start-Date: 2023-09-12  15:37:20
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Op>
Requested-By: tendencia (1000)
Install: fonts-lato:amd64 (2.0-2.1, automatic), liblua5.2-0:amd64 (5.2.4-2, aut>
Upgrade: vim-common:amd64 (2:8.2.3995-1ubuntu2.10, 2:8.2.3995-1ubuntu2.11), vim>
End-Date: 2023-09-12  15:38:57
```

Server 2:

```
tendencia@server2:~$ cd /var/log
tendencia@server2:/var/log$ ls
alternatives.log    cups          faillog          openvpn
alternatives.log.1  dist-upgrade  fontconfig.log   private
apt                 dmesg         gdm3             speech-dispatcher
auth.log            dmesg.0       gpu-manager.log  syslog
auth.log.1          dmesg.1.gz    hp               syslog.1
boot.log            dmesg.2.gz    installer        ubuntu-advantage.log
boot.log.1          dmesg.3.gz    journal          ubuntu-advantage.log.1
bootstrap.log       dmesg.4.gz    kern.log         unattended-upgrades
btmp                dpkg.log      kern.log.1       wtmp
btmp.1              dpkg.log.1    lastlog
tendencia@server2:/var/log$ cd apt
tendencia@server2:/var/log/apt$ sudo nano history.log
[sudo] password for tendencia:
```

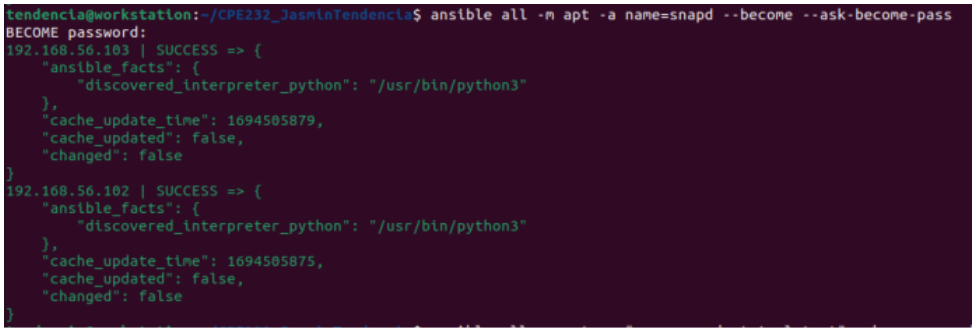
```
  GNU nano 6.2                          history.log

Start-Date: 2023-09-12  15:36:35
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Op>
Requested-By: tendencia (1000)
Install: fonts-lato:amd64 (2.0-2.1, automatic), liblua5.2-0:amd64 (5.2.4-2, aut>
Upgrade: vim-common:amd64 (2:8.2.3995-1ubuntu2.10, 2:8.2.3995-1ubuntu2.11), vim>
End-Date: 2023-09-12  15:37:25
```

*In the **history.log** file, it is a log of package management activities, consists of installations, upgrades, and removals, along with timestamps and details about each action. Tracking changes and diagnosing issues related to package management on the Ubuntu system.*

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?



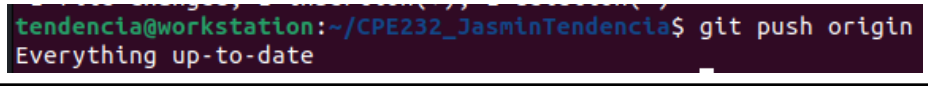
It has been successful. The command is using Ansible to attempt to install the snapd package on all hosts specified by the "all" group. Since the Ansible package is already installed and no available updates, Ansible will report success.

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.

This command tells Ansible to attempt to ensure that the snapd package is at its latest version on all hosts specified in the "all" group. The output will indicate that any changes were not made, the package was updated, and the task was successful.

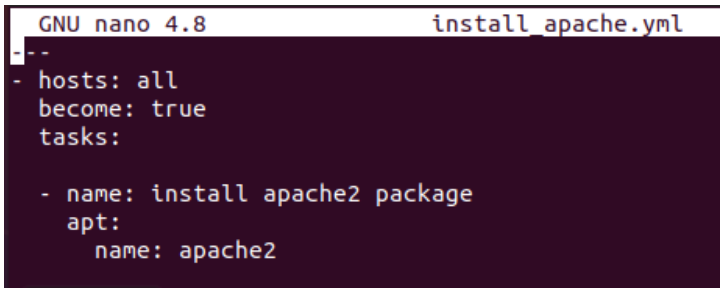
4. At this point, make sure to commit all changes to GitHub.



Task 2: Writing our First Playbook

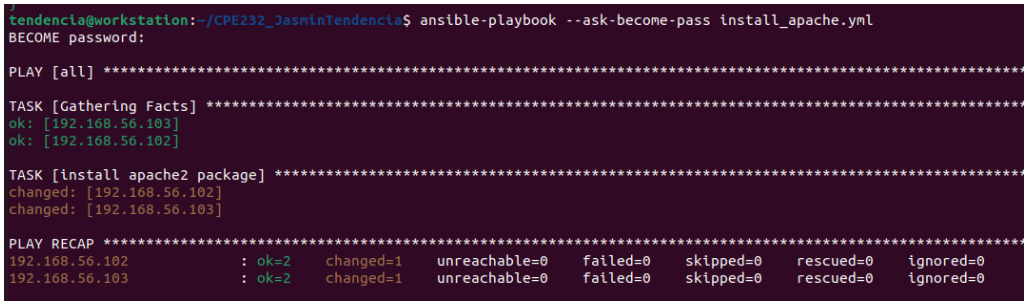
1. With ad hoc commands, we can simplify the administration of remote servers. For example, we can install updates, packages, and applications, etc. However, the real strength of ansible comes from its playbooks. When we write a playbook, we can define the state that we want our servers to be in and 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:



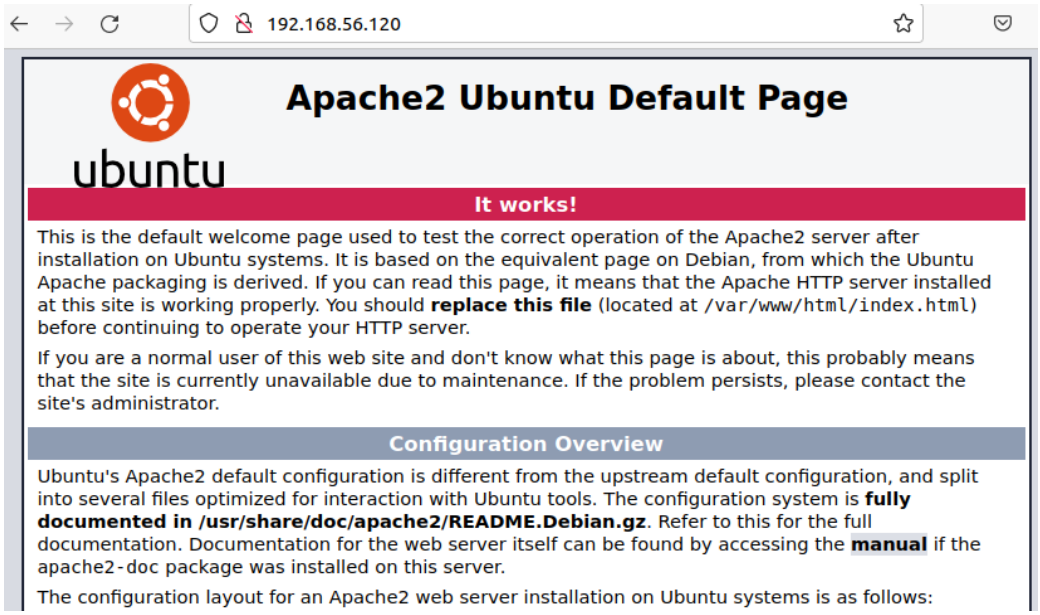
   Make sure to save the file. Take note also of the alignments of the texts.

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

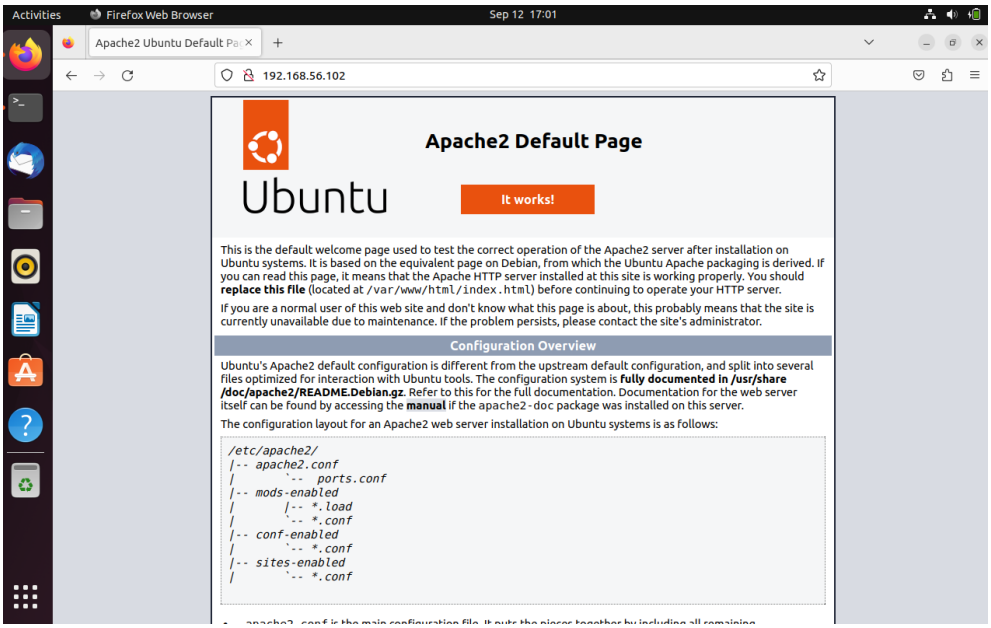


   It indicates that 2 is ok and there is 1 change for each of the servers.

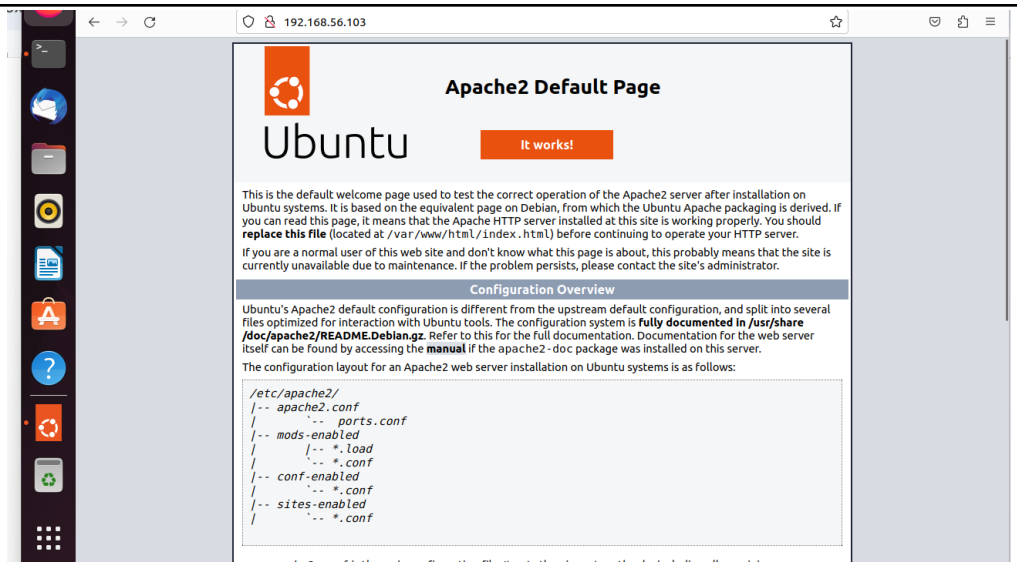
3. To verify that apache2 was installed automatically in the remote servers, go to the web browsers on each server and type its IP address. You should see something like this.



Server 1:



Server 2:



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?

```
tendencia@workstation:~/CPE232_JasminTendencia$ nano install_apache.yml
```

```
  GNU nano 6.2
---
- hosts: all
  become: true
  tasks:

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

```
tendencia@workstation:~/CPE232_JasminTendencia$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] ***********************************************************************************************
ok: [192.168.56.103]
ok: [192.168.56.102]

TASK [install apache2 package] ***************************************************************************************
fatal: [192.168.56.103]: FAILED! => {"changed": false, "msg": "No package matching 'ror' is available"}
fatal: [192.168.56.102]: FAILED! => {"changed": false, "msg": "No package matching 'ror' is available"}

PLAY RECAP ***********************************************************************************************************
192.168.56.102             : 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
```

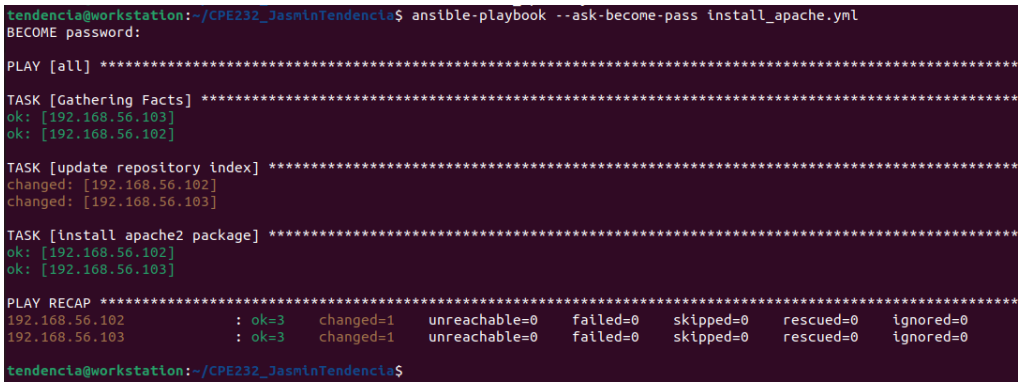
Does not recognize the package to be installed as a task since there is no such name of the package.

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?



As we take a good look on it, we can notice that there are 3 changes that happened on the 'play recap'.

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?

```
tendencia@workstation:~/CPE232_JasminTendencia$ nano install_apache.yml
tendencia@workstation:~/CPE232_JasminTendencia$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.103]

TASK [update repository index] *************************************************************************************
changed: [192.168.56.102]
changed: [192.168.56.103]

TASK [install apache2 package] *************************************************************************************
ok: [192.168.56.102]
ok: [192.168.56.103]

TASK [add PHP support for apache] **********************************************************************************
changed: [192.168.56.103]
changed: [192.168.56.102]

PLAY RECAP *********************************************************************************************************
192.168.56.102             : 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
```

This case is almost the same with the previous command but PHP support for the apache package was installed or added here which made 2 changes and 4 'ok' for the two servers.

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

   - *https://github.com/jrstendencia/CPE232_JasminTendencia.git*

Reflections:

Answer the following:

1. What is the importance of using a playbook?

- Ansible playbook is an essential tool in configuration management and automation for several reasons: it enables the automation of complex tasks and configurations, ensures that configurations are applied consistently across multiple systems, can be reused across different environments or for similar tasks, serves as documentation of the steps and configurations applied, can be shared among team members, and can be adapted and extended to accommodate new requirements, making them a scalable solution.

2. Summarize what we have done on this activity.

- We undertook a series of tasks related to Ansible and configuration management. We began by ensuring that our target machines were up to date by installing updates and upgrades, a common initial step. Then we installed Ansible itself, a powerful automation tool that allows us to manage remote machines efficiently. Here, we created an Ansible playbook, a set of instructions written in YAML format, to automate specific tasks. We executed this playbook to apply configurations and changes to remote machines. We modified the Ansible playbook to accommodate changing requirements or to enhance automation. By iterating on the playbook, we improved its effectiveness and efficiency. We adopted version control practices by using Git to track changes in our playbook. This allowed us to commit, save, and modify playbook files and then push these changes to a GitHub repository.

Conclusion:

This activity revolves around leveraging Ansible or an automation tool to effectively manage remote machines. By utilizing Ansible, we could achieve several key benefits. It allows us to execute commands across multiple remote machines in a simultaneous way, which reduces the time and effort required for system administration tasks. It also ensures that the same set of commands and configurations are applied consistently across all target machines. Then, automating tasks through Ansible playbooks enhances the reliability of system changes. Human errors are reduced, and rollback procedures can be easily implemented in case of unexpected issues, enhancing the overall stability of the environment.

In conclusion, the strategic use of Ansible commands and playbooks empowers us to efficiently manage remote machines while maintaining consistency, reliability, and scalability. This approach not only simplifies day-to-day operations but also sets the foundation for more advanced automation and orchestration of IT processes.