

Name: Pacinos, Angela Monique A.	Date Performed: 09-11-23
Course/Section: CPE232 - CPE31S4	Date Submitted: 09-12-23
Instructor: Dr. Jonathan V. Taylor	Semester and SY: 1st Semester '23 - '24
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: 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	

```

angela@workstation:~/Ansible_S4$ ansible all -m apt -a update_cache=true
192.168.56.115 | FAILED! => {
  "changed": false,
  "msg": "Failed to lock apt for exclusive operation"
}
192.168.56.114 | FAILED! => {
  "changed": false,
  "msg": "Failed to lock apt for exclusive operation"
}

```

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

The result shows a failed message for both server- "Failed to lock apt for exclusive operation"

Try editing the command and add 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.

```

angela@workstation:~/Ansible_S4$ ansible all -m apt -a update_cache=true
--become --ask-become-pass
SUDO password:
192.168.56.114 | SUCCESS => {
  "cache_update_time": 1694408255,
  "cache_updated": true,
  "changed": true
}
192.168.56.115 | SUCCESS => {
  "cache_update_time": 1694408256,
  "cache_updated": true,
  "changed": true
}

```

- Let's try to install VIM, which is an almost compatible version of the UNIX editor Vi. To do this, we will just change 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.

```

angela@workstation:~/Ansible_S4$ ansible all -m apt -a name=vim-nox
--become --ask-become-pass
SUDO password:
192.168.56.114 | SUCCESS => {
    "cache_update_time": 1694408255,
    "cache_updated": false,
    "changed": false
}
192.168.56.115 | SUCCESS => {
    "cache_update_time": 1694408256,
    "cache_updated": false,
    "changed": false
}

```

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, it shows the version installed.

```

angela@workstation:~/Ansible_S4$ which vim
angela@workstation:~/Ansible_S4$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/bionic-updates,bionic-security 2:8.0.1453-1ubuntu1.13 amd64
Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/bionic-updates,bionic-security,now 2:8.0.1453-1ubuntu1.13
amd64 [installed]
Vi IMproved - enhanced vi editor - compact version

```

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*.

```

angela@server1:~$ cd /var/log
angela@server1:/var/log$ ls
alternatives.log      cups                  journal              tallylog
alternatives.log.1    dist-upgrade         kern.log             ubuntu-advantage.log
apt                   dpkg.log             kern.log.1           ubuntu-advantage.log.1
auth.log              dpkg.log.1           kern.log.2.gz        ufw.log
auth.log.1            faillog              lastlog              ufw.log.1
auth.log.2.gz         fontconfig.log        speech-dispatcher     ufw.log.2.gz
boot.log              gdm3                 syslog               unattended-upgrades
bootstrap.log         gpu-manager.log       syslog.1             wtmp
btmtp                 hp                    syslog.2.gz          wtmp.1
btmtp.1               installer            syslog.3.gz
angela@server1:/var/log$ cd apt
angela@server1:/var/log/apt$ ls
eipp.log.xz  history.log  history.log.1.gz  term.log  term.log.1.gz
angela@server1:/var/log/apt$ cat history.log

Start-Date: 2023-09-11 12:52:00
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Options::=--force-confold install vim-nox
Requested-By: angela (1000)
Install: javascript-common:amd64 (11, automatic), ruby2.5:amd64 (2.5.1-1ubuntu1.16, automatic), rake:amd64 (12.3.1-1ubuntu0.1, automatic), ruby-net-telnet:amd64 (0.1.1-2, automatic), libtcl8.6:amd64 (8.6.8+dfsg-3, automatic), libjs-jquery:amd64 (3.2.1-1, automatic), vim-nox:amd64 (2:8.0.1453-1ubuntu1.13), ruby-minitest:amd64 (5.10.3-1, automatic), libruby2.5:amd64 (2.5.1-1ubuntu1.16, automatic), ruby:amd64 (1:2.5.1, automatic), vim-runtime:amd64 (2:8.0.1453-1ubuntu1.13, automatic), liblua5.2-0:amd64 (5.2.4-1.1build1, automatic), ruby-power-assert:amd64 (0.3.0-1, automatic), rubygems-integration:amd64 (1.11, automatic), fonts-lato:amd64 (2.0-2, automatic), ruby-test-unit:amd64 (3.2.5-1, automatic), ruby-did-you-mean:amd64 (1.2.0-2, automatic)
End-Date: 2023-09-11 12:52:07

```

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 is the same as the previous result where the package was installed, but this time it was for snapd.

```

angela@workstation:~/Ansible_S4$ ansible all -m apt -a name=snapd --become --ask-become-pass
SUDO password:
192.168.56.114 | SUCCESS => {
    "cache_update_time": 1694408255,
    "cache_updated": false,
    "changed": false
}
192.168.56.115 | SUCCESS => {
    "cache_update_time": 1694408256,
    "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.

```
angela@workstation:~/Ansible_S4$ ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass
SUDO password:
192.168.56.114 | SUCCESS => {
  "cache_update_time": 1694408255,
  "cache_updated": false,
  "changed": false
}
192.168.56.115 | SUCCESS => {
  "cache_update_time": 1694408256,
  "cache_updated": false,
  "changed": false
}
```

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

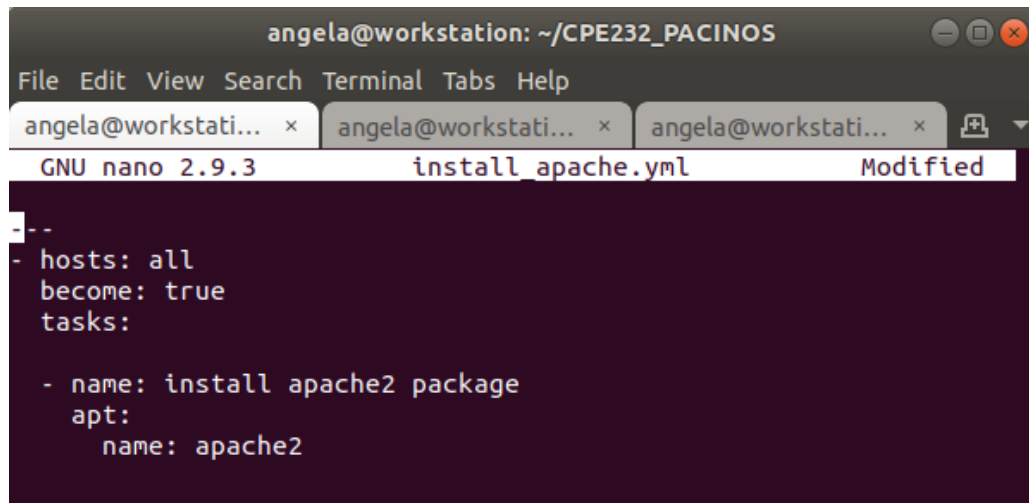
```
angela@workstation:~/Ansible_S4$ git add *
angela@workstation:~/Ansible_S4$ git commit -m "Act4"
[main 9e426cb] Act4
 2 files changed, 13 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 inventory
angela@workstation:~/Ansible_S4$ git push origin
Username for 'https://github.com': Angela-Pacinos
Password for 'https://Angela-Pacinos@github.com':
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 489 bytes | 489.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/Angela-Pacinos/Ansible_S4.git
 3b349c5..9e426cb  main -> main
```

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:



```
angela@workstation: ~/CPE232_PACINOS
File Edit View Search Terminal Tabs Help
angela@workstati... x angela@workstati... x angela@workstati... x
GNU nano 2.9.3 install_apache.yml Modified
---
- 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.

The apache2 package was successfully installed in both of the servers.



```
angela@workstation:~/CPE232_PACINOS$ ansible-playbook --ask-become-pass inst
all_apache.yml
SUDO password:

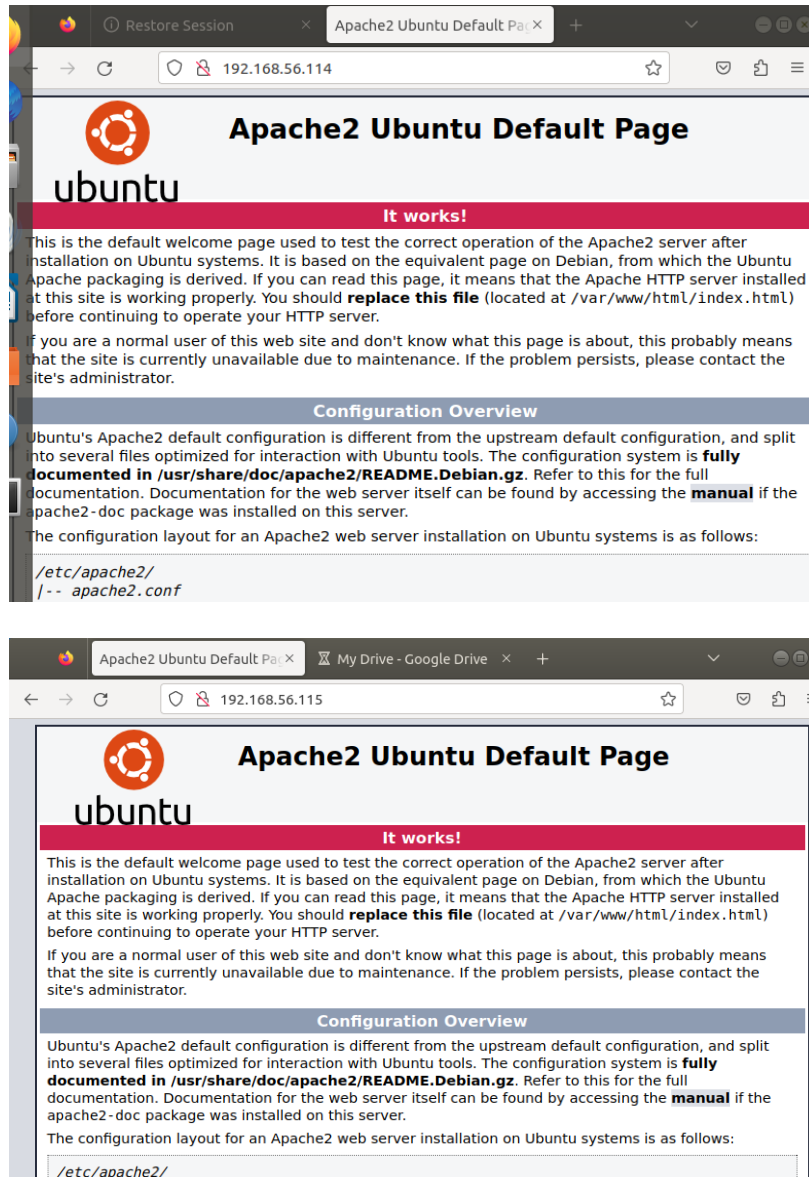
PLAY [all] *****
****

TASK [Gathering Facts] *****
****
ok: [192.168.56.114]
ok: [192.168.56.115]

TASK [install apache2 package] *****
****
changed: [192.168.56.114]
changed: [192.168.56.115]

PLAY RECAP *****
****
192.168.56.114      : ok=2    changed=1    unreachable=0    failed=0
192.168.56.115      : 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?

An error message was shown since it didn't recognize the package "apachE2" that I put inside the *install_apache.yml* file.


```

angela@workstation:~/CPE232_PACINOS$ sudo nano install_apache.yml
angela@workstation:~/CPE232_PACINOS$ ansible-playbook --ask-become-pass in
stall_apache.yml
SUDO password:

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

TASK [Gathering Facts] *****
*****
ok: [192.168.56.114]
ok: [192.168.56.115]

TASK [install apache2 package] *****
*****
fatal: [192.168.56.114]: FAILED! => {"changed": false, "msg": "No package
matching 'apache2' is available"}
fatal: [192.168.56.115]: FAILED! => {"changed": false, "msg": "No package
matching 'apache2' is available"}
to retry, use: --limit @/home/angela/CPE232_PACINOS/install_apache
.retry

PLAY RECAP *****
*****
192.168.56.114      : ok=1    changed=0    unreachable=0    failed=
1
192.168.56.115      : ok=1    changed=0    unreachable=0    failed=
1

```

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?

Yes, the package was installed and the repository was updated.


```

angela@workstation:~/CPE232_PACINOS$ sudo nano install_apache.yml
[sudo] password for angela:
angela@workstation:~/CPE232_PACINOS$ ansible-playbook --ask-become-pass in
stall_apache.yml
SUDO password:

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

TASK [Gathering Facts] *****
*****
ok: [192.168.56.114]
ok: [192.168.56.115]

TASK [update repository index] *****
*****
changed: [192.168.56.114]
changed: [192.168.56.115]

TASK [install apache2 package] *****
*****
ok: [192.168.56.114]
ok: [192.168.56.115]

PLAY RECAP *****
*****
192.168.56.114      : ok=3    changed=1    unreachable=0    failed=
0
192.168.56.115      : 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.

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

Yes, the package was installed and the repository was updated.

```

angela@workstation:~/CPE232_PACINOS$ sudo nano install_apache.yml
angela@workstation:~/CPE232_PACINOS$ ansible-playbook --ask-become-pass in
stall_apache.yml
SUDO password:

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

TASK [Gathering Facts] *****
*****
ok: [192.168.56.114]
ok: [192.168.56.115]

TASK [update repository index] *****
*****
changed: [192.168.56.114]
changed: [192.168.56.115]

TASK [install apache2 package] *****
*****
ok: [192.168.56.114]
ok: [192.168.56.115]

TASK [add PHP support for apache] *****
*****
changed: [192.168.56.114]
changed: [192.168.56.115]

PLAY RECAP *****
*****
192.168.56.114      : ok=4    changed=2    unreachable=0    failed=
0
192.168.56.115      : ok=4    changed=2    unreachable=0    failed=
0

```

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

```

angela@workstation:~/CPE232_PACINOS$ git init
Reinitialized existing Git repository in /home/angela/CPE232_PACINOS/.git/
angela@workstation:~/CPE232_PACINOS$ git add *
angela@workstation:~/CPE232_PACINOS$ git commit -m "ACT4"
[main dadb46f] ACT4
4 files changed, 29 insertions(+)
create mode 100644 ansible.cfg
create mode 100644 install_apache.retry
create mode 100644 install_apache.yml
create mode 100644 inventory
angela@workstation:~/CPE232_PACINOS$ git push origin
Warning: Permanently added the ECDSA host key for IP address '20.87.225.21
2' to the list of known hosts.
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 734 bytes | 734.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:Angela-Pacinos/CPE232_PACINOS.git
7236010..dadb46f  main -> main

```

Reflections:

Answer the following:

1. What is the importance of using a playbook?

A playbook is an automation file that we can use to store configurations for the system. This one is important to use because it can make the tasks a little bit easier. As we can store the configurations we want to make changes in the system, inside the playbook and run the playbook to execute it all at once.

2. Summarize what we have done on this activity.

The first thing that we have done is to install the ansible because this is the automation tool that we used to run all the proceeding process. There was a bit of struggle with connecting the ansible with the 2 servers that I have. It was resolved when I found out that the problem was with the ssh of the server 2. The proceeding process when smoothly as there were given guides on what to do. We have installed and updated the ansible to properly use it successfully. We created a playbook and tried different configurations on it to see if it was a success or fail.

Conclusions:

For this activity the main tool that we have used for the system is Ansible. This is a tool that we can and have used for different tasks and for continuous running of the tasks without sacrificing the downtime of the system. We have installed different packages like vim and snapd which was successful. We also were able to create an ansible playbook where we modified the install_apache.yml file with different configurations to see the different results that it will put out. We have tried adding two configurations for the system inside the file and it worked perfectly. I also have tried adding one more to the existing two and it also worked just fine. We just have to make sure the commands inside the file was properly spelled and spaced for the file to run smoothly. Overall, we have learned new was that we can manage the system and the servers connected to it as well as for easier automation.

"I affirm that I will not give or receive any unauthorized help on this activity and that all work will be my own."