Name: Yuri P. Nollan	Date Performed: 09/14/2023
Course/Section: CPE31S6	Date Submitted: 09/14/2023
Instructor: Dr. Jonathan Taylar	Semester and SY: 1st Sem AY 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?

```
workstation@workstation:~/sysAds6$ ansible all -m apt -a update_cache=true
192.168.56.104 | FAILED! => {
    "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! => {
    "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)"
}
workstation@workstation:~/sysAds6$
```

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.

```
workstation@workstation:~/sysAds6$ ansible all -m apt -a update_cache=true --bec
ome --ask-become-pass
BECOME password:
192.168.56.104 | CHANGED => {
    "cache_update_time": 1694682764,
    "cache_updated": true,
    "changed": true
}
192.168.56.102 | CHANGED => {
    "cache_update_time": 1694682760,
    "cache_updated": true,
    "changed": true
}
```

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.

```
workstation@workstation:~/sysAds6$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
192.168.56.102 | CHANGED => {
    "cache_update_time": 1694682760,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\nThe following additional packages will be installed:\n  fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby2.5 libtcl8.6\n  rake ruby ruby-did-you-mean ruby-minitest ruby-net-telnet ruby-power-assert\n  ruby-test-unit ruby2.5 rubygems-integration vim-runtime\nSuggested packages:\n  apache2 | lighttpd | httpd tcl8.6 ri ruby-dev bundler cscope vim-doc\nThe following NEW packages will be installed:\n  fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby2.5 libtcl8.6\n  rake ruby ruby-did-you-mean ruby-minitest ruby-net-telnet ruby-power-assert\n  ruby-test-unit ruby2.5 rubygems-integration vim-nox vim-runtime\n0 upgraded, 17 newly installed, 0 to remove and 53 not upgraded.\nNeed to get 13.8 MB of archives.\nAfter this operation, 64.5 MB of additional disk space will be used.\nGet:1 http://ph.archive.ubuntu.com/ubuntu bionic/main am
```

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?

```
workstation@workstation:~/sysAds6$ which vim
workstation@workstation:~/sysAds6$ 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,automatic]
  Vi IMproved - enhanced vi editor - compact version

workstation@workstation:~/sysAds6$
```

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.

```
workstation@workstation:/var/log$ cd /var/log/apt
workstation@workstation:/var/log/apt$ ls
eipp.log.xz  history.log  term.log
workstation@workstation:/var/log/apt$ cat history.log

Start-Date: 2021-09-15  20:17:59
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes install ubuntu-
keyring
Upgrade: ubuntu-keyring:amd64 (2018.02.28, 2018.09.18.1~18.04.2)
End-Date: 2021-09-15  20:17:59

Start-Date: 2021-09-15  20:18:05
Commandline: apt-get --yes -oDebug::pkgDepCache::AutoInstall=yes --force-yes upg
rade
Upgrade: libpam0g:amd64 (1.1.8-3.6ubuntu2, 1.1.8-3.6ubuntu2.18.04.3), debconf:am
d64 (1.5.66, 1.5.66ubuntu1), fdisk:amd64 (2.31.1-0.4ubuntu3, 2.31.1-0.4ubuntu3.7
), perl-base:amd64 (5.26.1-6, 5.26.1-6ubuntu0.5), libdevmapper1.02.1:amd64 (2:1.
02.145-4.1ubuntu3, 2:1.02.145-4.1ubuntu3.18.04.3), dmsetup:amd64 (2:1.02.145-4.1
ubuntu3, 2:1.02.145-4.1ubuntu3.18.04.3), libpython3.6-minimal:amd64 (3.6.5-3, 3.
6.9-1~18.04ubuntu1.4), libseccomp2:amd64 (2.3.1-2.1ubuntu4, 2.5.1-1ubuntu1~18.04
.1), libaudit-common:amd64 (1:2.8.2-1ubuntu1, 1:2.8.2-1ubuntu1.1), libcom-err2:a
md64 (1.44.1-1, 1.44.1-1ubuntu1.3), libdbus-1-3:amd64 (1.12.2-1ubuntu1, 1.12.2-1
```

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?

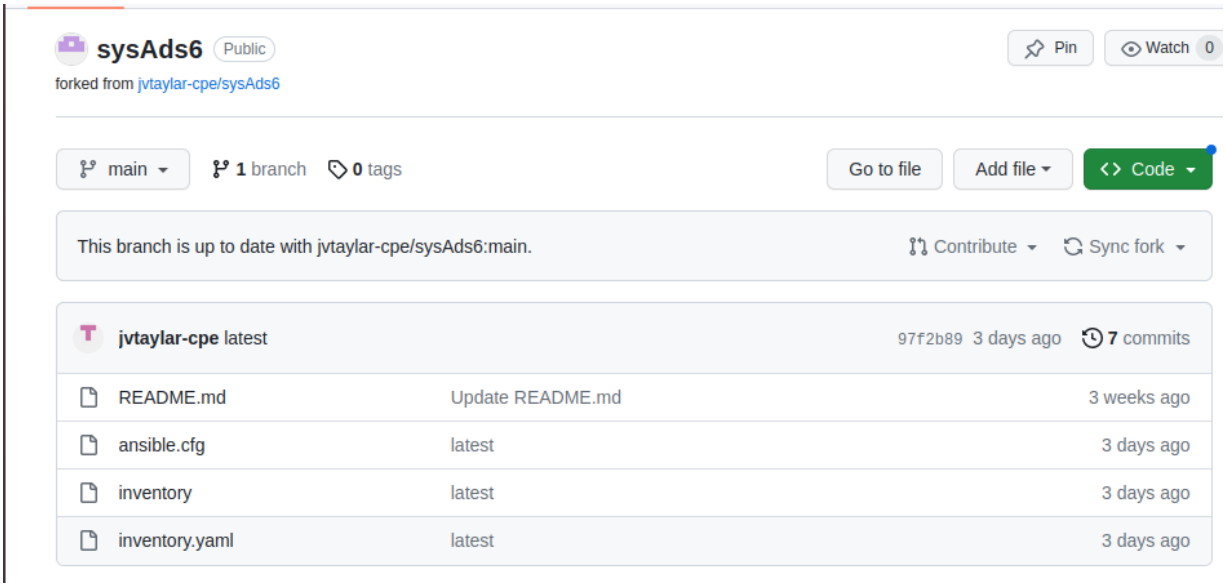
```
workstation@workstation:~/sysAds6$ ansible all -m apt -a name=snapd --become --a
sk-become-pass
BECOME password:
192.168.56.104 | SUCCESS => {
    "cache_update_time": 1694682764,
    "cache_updated": false,
    "changed": false
}
192.168.56.102 | SUCCESS => {
    "cache_update_time": 1694682760,
    "cache_updated": false,
    "changed": false
}
workstation@workstation:~/sysAds6$
```

   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.

```
workstation@workstation:~/sysAds6$ ansible all -m apt -a "name=snapd state=lates
t" --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
    "cache_update_time": 1694682760,
    "cache_updated": false,
    "changed": false
}
192.168.56.104 | SUCCESS => {
    "cache_update_time": 1694682764,
    "cache_updated": false,
    "changed": false
}
workstation@workstation:~/sysAds6$
```

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:

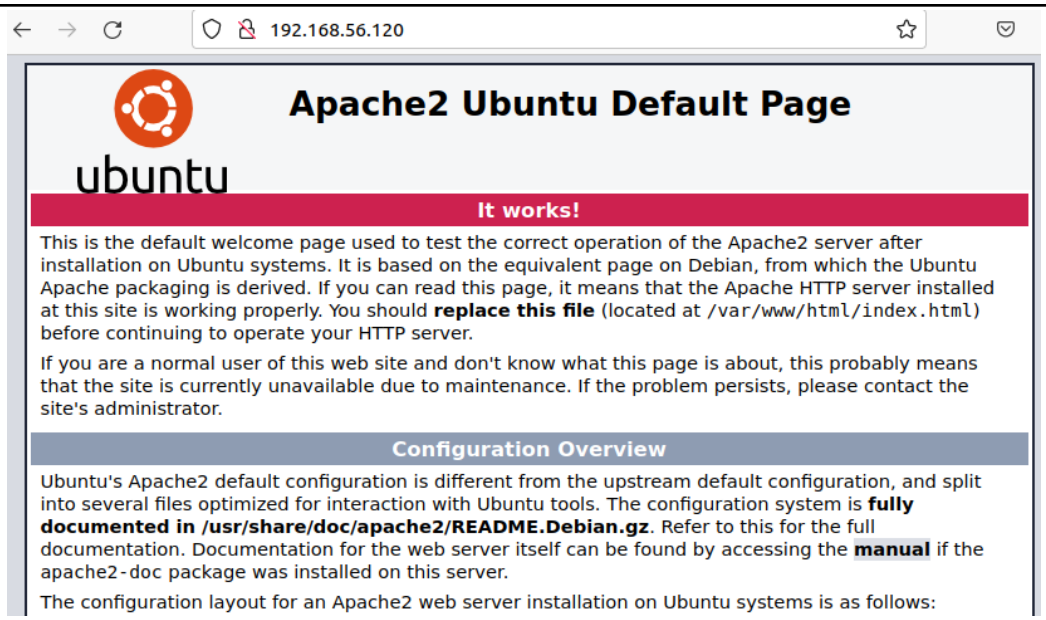
```
  GNU nano 4.8                  install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - 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.

```
workstation@workstation:~/sysAds6$ ansible-playbook --ask-become-pass install_ap
ache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [install apache2 package] *************************************************
changed: [192.168.56.104]
changed: [192.168.56.102]

PLAY RECAP *********************************************************************
192.168.56.102             : ok=2    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.56.104             : ok=2    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

workstation@workstation:~/sysAds6$
```

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?
   - Error.
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?

```
workstation@workstation:~/sysAds6$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.104]
ok: [192.168.56.102]

TASK [update repository index] *************************************************
changed: [192.168.56.104]
changed: [192.168.56.102]

TASK [install apache2 package] *************************************************
ok: [192.168.56.102]
ok: [192.168.56.104]

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

workstation@workstation:~/sysAds6$
```

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?

```
workstation@workstation:~/sysAds6$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.56.102]
ok: [192.168.56.104]

TASK [update repository index] *************************************************
changed: [192.168.56.104]
changed: [192.168.56.102]

TASK [install apache2 package] *************************************************
ok: [192.168.56.102]
ok: [192.168.56.104]

TASK [add PHP support for apache] **********************************************
changed: [192.168.56.104]
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
192.168.56.104             : ok=4    changed=2    unreachable=0    failed=0    skipped=0
  rescued=0    ignored=0

workstation@workstation:~/sysAds6$
```

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

   https://github.com/yorehh/sysAds6.git

Reflections:

Answer the following:

1. What is the importance of using a playbook?
   - Playbooks are crucial instruments for achieving consistency, efficiency, risk mitigation, and continual improvement in a variety of areas of business operations. Whether in business, athletics, cybersecurity, or other sectors, using a playbook can improve decision-making, streamline processes, and lead to success in the long run.
2. Summarize what we have done on this activity.
   - First of all, we have installed ansible and python3 in our remote computer or our workstation to properly execute the activity, we then used it to create and connect it to a github repository to make sure that the system is working properly. Lastly, we tried pinging the connection to other networks and servers to verify that it is working properly.

SUMMARY AND CONCLUSION

Use commands that makes changes to remote machines

- Use playbook in automating ansible commands
- In this activity, I have learned different types of commands that make changes to the remote computers. I was also able to use the playbook properly when automating ansible commands. Overall, this activity helped me understand more about linux commands and enhance my skills in troubleshooting and navigating through the ubuntu linux.