Name: Christian Jaraula	Date Performed: 9/15/2022
Course/Section: CPE232	Date Submitted: 9/15/2022
Instructor: Prof. Taylar	Semester and SY:

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

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible all -m apt -a update_ca
che=true
192.168.56.106 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Load key \"/home/christianja
raula/.ssh\": Is a directory\r\nchristianjaraula@192.168.56.106: Permission den
ied (publickey,password).",
    "unreachable": true
}
192.168.56.105 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Load key \"/home/christianja
raula/.ssh\": Is a directory\r\nchristianjaraula@192.168.56.105: Permission den
ied (publickey,password).",
    "unreachable": true
}
```

What is the result of the command? Is it successful?
The command failed.

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.

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible all -m apt -a update_ca
che=true --become --ask-become-pass
BECOME password:

192.168.56.106 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219228,
    "cache_updated": true,
    "changed": true
}
192.168.56.105 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219231,
    "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.

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible all -m apt -a name=vim-
nox --become --ask-become-pass
BECOME password:
192.168.56.106 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219228,
    "cache_updated": false,
    "changed": true,
    "stderr": "",
    "stderr_lines": [],
    "stdout": "Reading package lists...\nBuilding dependency tree...\nReading s
tate information...\nThe following additional packages will be installed:\n  fo
nts-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n  rub
y-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n  rubygems-integra
tion vim-runtime\nSuggested packages:\n  apache2 | lighttpd | httpd ri ruby-dev
 bundler cscope vim-doc\nThe following NEW packages will be installed:\n  fonts
-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n  ruby-n
et-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n  rubygems-integratio
n vim-nox vim-runtime\n0 upgraded, 15 newly installed, 0 to remove and 46 not u
pgraded.\nNeed to get 17.5 MB of archives.\nAfter this operation, 76.3 MB of ad
ditional disk space will be used.\nGet:1 http://ph.archive.ubuntu.com/ubuntu ja
mmy/main amd64 fonts-lato all 2.0-2.1 [2696 kB]\nGet:2 http://ph.archive.ubuntu
.com/ubuntu jammy/main amd64 javascript-common all 11+nmu1 [5936 B]\nGet:3 http
://ph.archive.ubuntu.com/ubuntu jammy/main amd64 libjs-jquery all 3.6.0+dfsg+~3
.5.13-1 [321 kB]\nGet:4 http://ph.archive.ubuntu.com/ubuntu jammy/universe amd6
```

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

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ which vim
christianjaraula@LocalMachine:~/CPE232_Jaraula$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy 2:8.2.3995-1ubuntu2 amd64
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

christianjaraula@LocalMachine:~/CPE232_Jaraula$
```

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.

```
  GNU nano 6.2                          history.log

Start-Date: 2022-09-15  09:57:48
Commandline: apt install ansible
Requested-By: christianjaraula (1000)
Install: python-babel-localedata:amd64 (2.8.0+dfsg.1-7, automatic), python3-dn>
End-Date: 2022-09-15  09:58:10

Start-Date: 2022-09-15  12:54:27
Commandline: apt install git
Requested-By: christianjaraula (1000)
Install: git:amd64 (1:2.34.1-1ubuntu1.4), liberror-perl:amd64 (0.17029-1, auto>
End-Date: 2022-09-15  12:54:28
```

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?
   The command was effective. It has no effect on the remote's functionality. servers.

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible all -m apt -a name=snap
d --become --ask-become-pass
BECOME password:
192.168.56.106 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219228,
    "cache_updated": false,
    "changed": false
}
192.168.56.105 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219231,
    "cache_updated": false,
    "changed": false
}
christianjaraula@LocalMachine:~/CPE232_Jaraula$
```

3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible all -m apt -a "name=s
pd state=latest" --become --ask-become-pass
BECOME password:
192.168.56.106 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219228,
    "cache_updated": false,
    "changed": false
}
192.168.56.105 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1663219231,
    "cache_updated": false,
    "changed": false
}
```

Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

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

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git add ansible.cfg
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git add inventory
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git commit -m "git"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'christianjaraula@LocalMachine.
(none)')
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git config --global user.email
"jaraulachristian@gmail.com
> "
```

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git config --global user.name
christianjaraula"
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git commit -m "git"
[main a8586c2] git
 2 files changed, 13 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 inventory
christianjaraula@LocalMachine:~/CPE232_Jaraula$
```

```
christianjaraula@LocalMachine:~$ cd CPE232_Jaraula
christianjaraula@LocalMachine:~/CPE232_Jaraula$ git push origin main
Username for 'https://github.com': christianjaraula
Password for 'https://christianjaraula@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 468 bytes | 468.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/christianjaraula/CPE232_Jaraula
   33b70cb..a8586c2  main -> main
christianjaraula@LocalMachine:~/CPE232_Jaraula$
```

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
```

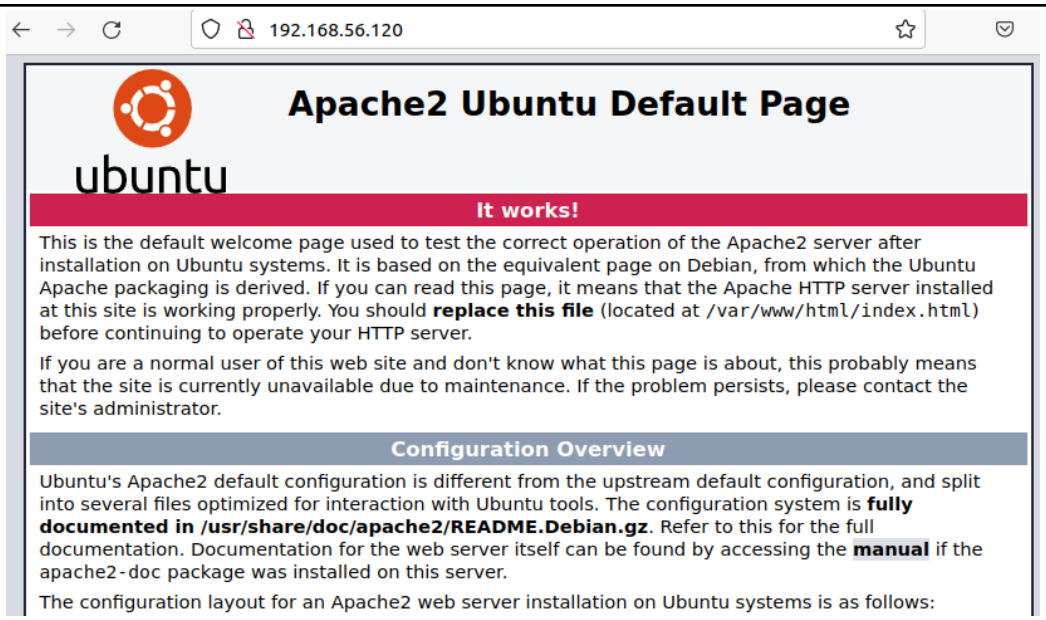
Make sure to save the file. Take note also of the alignments of the texts.

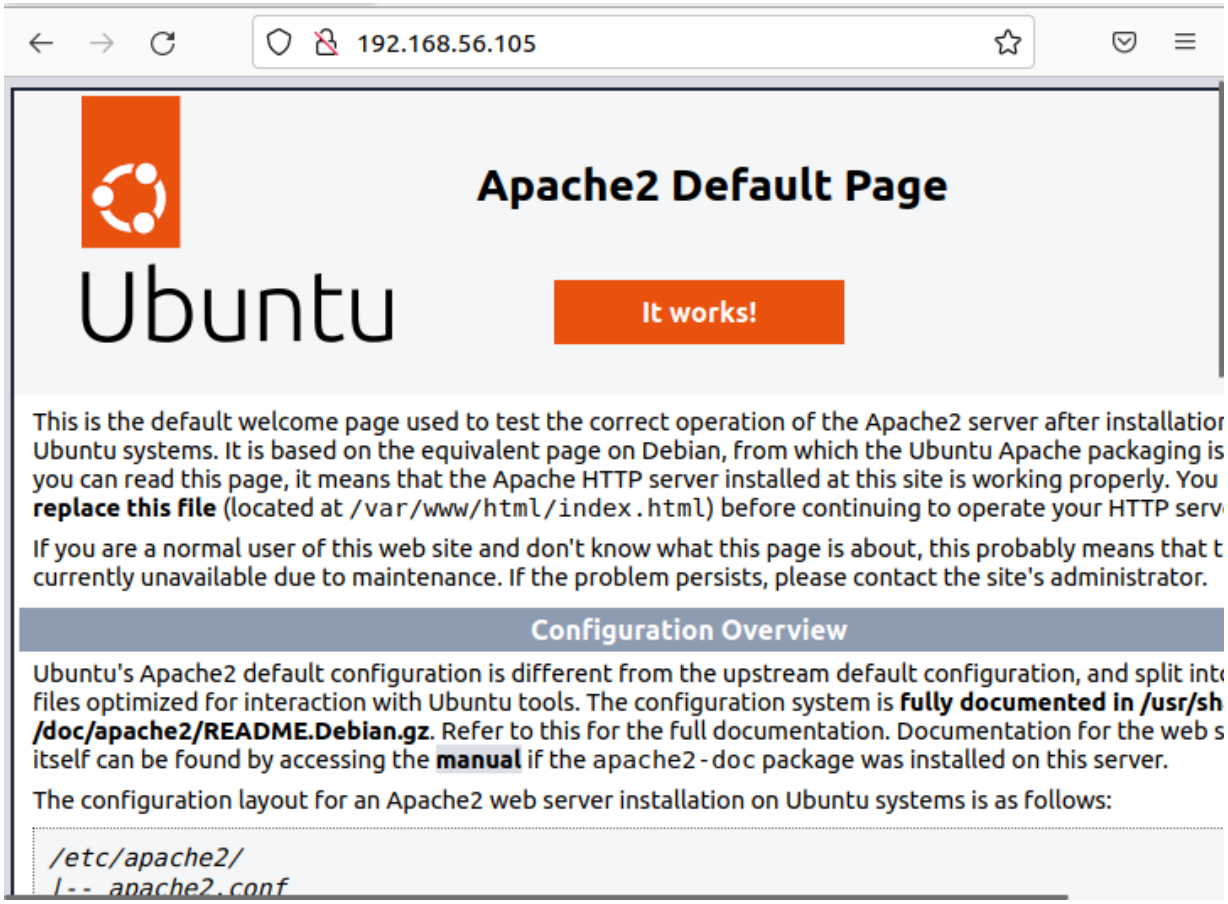
```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ls
ansible.cfg  install_apache.yml  inventory  README.md
christianjaraula@LocalMachine:~/CPE232_Jaraula$
```

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

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ sudo nano install_apache.yml
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible-playbook --ask-become-p
ass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.105]
ok: [192.168.56.106]

TASK [install apache2 package] *************************************************
*
changed: [192.168.56.105]
changed: [192.168.56.106]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.56.106             : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=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?
   The remote servers are updated after the command is successful.

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?
The remote servers are updated after the command is successful.

```
christianjaraula@LocalMachine:~/CPE232_Jaraula$ ansible-playbook --ask-become-p
ass install_apache.yml
BECOME password:

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

TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.106]
ok: [192.168.56.105]

TASK [update repository index] *************************************************
*
changed: [192.168.56.105]
changed: [192.168.56.106]

TASK [install apache2 package] *************************************************
*
ok: [192.168.56.105]
ok: [192.168.56.106]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.56.106             : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=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.

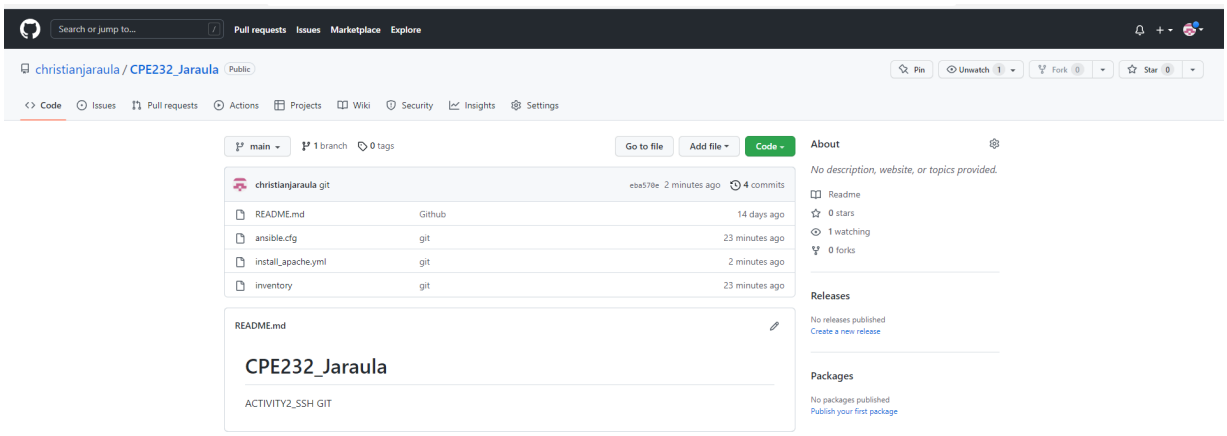
```
---
- hosts: all
  become: true
  tasks:

  - 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?
The command is successful and it adds PHP support for apache.

```
TASK [Gathering Facts] *********************************************************
*
ok: [192.168.56.106]
ok: [192.168.56.105]

TASK [update repository index] *************************************************
*
changed: [192.168.56.106]
changed: [192.168.56.105]

TASK [install apache2 package] *************************************************
*
ok: [192.168.56.106]
ok: [192.168.56.105]

TASK [add PHP support for apache] **********************************************
*
changed: [192.168.56.106]
changed: [192.168.56.105]

PLAY RECAP *********************************************************************
*
192.168.56.105             : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.56.106             : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

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



https://github.com/christianjaraula/CPE232_Jaraula

Reflections:

Answer the following:

1. What is the importance of using a playbook?

   Ansible Playbooks are essentially frameworks of prewritten code that developers can use ad hoc or as a starting point. Ansible Playbooks are commonly used to automate IT infrastructure, networks, security systems, and developer personas.

2. Summarize what we have done on this activity.

   I must first use ssh keys to connect to the other virtual machine, and then I'll connect using a public key. After that, I set up Ansible so I could modify servers located far away. Additionally, I was able to create a plan for installing PHP, Apache 2, and a repository index as well as changing the remote servers. In order to automate ansible commands, I utilized a playbook. I synced my playbook, made a commit, then uploaded it to my github account before finishing.