

Name: Buenvenida, Ken Benedict D.	Date Performed: 09/11/2023
Course/Section: CpE232 - CpE31S4	Date Submitted: 09/11/2023
Instructor: Engr. Jonathan Taylar	Semester and SY: 3rd Year 1st Semester
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: <i>Provide screenshots for each task.</i> 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 <i>sudo apt update</i> when we want to download package information from all configured resources. The sources often defined in <i>/etc/apt/sources.list</i> file and other files located in <i>/etc/apt/sources.list.d/</i> 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?

- It failed due to the lack of command that bypasses the sudo command. Compared to the second instructions that equates to a sudo.

```
ken@manageNode:~/ansible$ ansible all -m apt -a update_cache=true
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
r/lib/apt/lists/lock - open (13: Permission denied)"
}
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: Co
  "unreachable": true
}
```

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.

```
ken@manageNode:~/ansible$ 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": 1694440672,
  "cache_updated": true,
  "changed": true
}
192.168.56.103 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694440673,
  "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.

```
ken@manageNode:~/ansible$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
192.168.56.102 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694438940,
  "cache_updated": false,
  "changed": false
}
192.168.56.103 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694438473,
  "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 font-s-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n ruby-n
```

- 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?

```
ken@manageNode:~/ansible$ which vim
/usr/bin/vim
```

```
ken@manageNode:~/ansible$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy-updates,jammy-security 2:8.2.3995-1ubuntu2.11 amd64
Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy-updates,jammy-security,now 2:8.2.3995-1ubuntu2.11 amd64 [install
d,automatic]
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`.

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

```
ken@manageNode:~/ansible$ ansible all -m apt -a "name=snapd state=latest" --beco
me --ask-become-pass
BECOME password:
192.168.56.103 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694438473,
  "cache_updated": false,
  "changed": false
}
192.168.56.102 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694438940,
  "cache_updated": false,
  "changed": false
}
ken@manageNode:~/ansible$
```

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

```
ken@manageNode:~/ansible$ git add *
ken@manageNode:~/ansible$ git commit -m "created first playbook"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
ken@manageNode:~/ansible$ git push origin
Everything up-to-date
ken@manageNode:~/ansible$
```

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.

```

GNU nano 6.2                                install_apache.yml
--
- hosts: all
  become: true
  tasks:

    - name: install apache2 package
      apt:
        name: apache2

```

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

```

ken@manageNode:~/ansible$ 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 [install apache2 package] *****
ok: [192.168.56.103]
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
192.168.56.103      : ok=2    changed=0    unreachable=0    failed=0    s
kipped=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.

← → ↻

🔒 192.168.56.102

☆

🔔



Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
|
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining

← → ↻

🔒 192.168.56.103

☆

🔔



Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
|
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining

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?

```

ken@manageNode:~/ansible$ sudo nano install_apache.yml
[sudo] password for ken:
ken@manageNode:~/ansible$ 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 [install apache2 package] *****
fatal: [192.168.56.102]: FAILED! => {"changed": false, "msg": "No package matching 'test' is available"}
fatal: [192.168.56.103]: FAILED! => {"changed": false, "msg": "No package matching 'test' 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
ken@manageNode:~/ansible$

```

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.

```

--
- hosts: all
  become: true
  tasks:

    - name: update repository index
      apt:
        update_cache: yes
    - name: install apache2 package
      apt:
        name: apache2

```


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

```
ken@manageNode:~/ansible$ 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]

PLAY RECAP *****
192.168.56.102      : ok=3    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
192.168.56.103      : ok=3    changed=1    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

ken@manageNode:~/ansible$
```

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.

```
GNU nano 6.2                                install_apache.yml
---
- 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
```

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

```
ken@manageNode:~/ansible$ sudo nano install_apache.yml
ken@manageNode:~/ansible$ 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 [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]

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

ken@manageNode:~/ansible$ sudo nano install_apache.yml
ken@manageNode:~/ansible$ 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 [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: [192.168.56.102]
changed: [192.168.56.103]

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
```

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

<https://github.com/KBDBuenvenida/ansible.git>

Reflections:

Answer the following:

1. What is the importance of using a playbook?

- **A playbook serves as a guide that outlines the steps, methods and recommended practices. Utilizing a playbook offers advantages.**

1. Improved. Productivity: By providing teams with a roadmap a playbook enables them to work efficiently. This saves time and resources while minimizing errors.

2. Consistency: A playbook ensures that teams consistently deliver high quality outcomes. It achieves this by establishing procedures that everyone can adhere to.

3. Enhanced communication and collaboration: With a playbook in place, communication and collaboration within teams improve significantly. It fosters shared understanding of objectives, goals, roles and responsibilities among team members.

2. Summarize what we have done on this activity.

- **What we did this activity is to practice using ansible commands that allows teams to work easier and collaborate easily. We did the update cache command using ansible, and used apache2 to test the connection between machines. We also tried other ansible commands that truly failed to see if the command would still work or not, it outputs some errors but it still works in the end. Lastly, we used the Git command that allows us to update the changes we did to our repository.**