

Name: Rustom C. Cariño	Date Performed: 9/12/2023
Course/Section: CPE232/S5	Date Submitted: 9/12/2023
Instructor: Engr. Roman Richard	Semester and SY: 1st sem / 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: <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:	

```

rustom@LocalMachine:~$ sudo apt update
[sudo] password for rustom:
Hit:1 http://ph.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://ph.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://ph.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://ph.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [972 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:6 http://ph.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [484 kB]
Get:7 http://ph.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [223 kB]
Get:8 http://ph.archive.ubuntu.com/ubuntu jammy-updates/main amd64 DEP-11 Metadata [101 kB]
Get:9 http://ph.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [15.6 kB]
Get:10 http://ph.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [484 kB]

```

ansible all -m apt -a update_cache=true

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

-The command is unsuccessful since ansible is not yet installed.

```

rustom@LocalMachine:~$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ieee-data python-babel-localedata python3-argcomplete python3-babel

```

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.

```

rustom@LocalMachine:~$ ansible all -m apt -a update_cache=true
127.0.0.1 | 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/: E:Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)"
}

```

```

rustom@LocalMachine:/etc/ansible$ ansible all -m apt -a update_cache=true --become
me --ask-become-pass
BECOME password:
127.0.0.1 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694513816,
  "cache_updated": true,
  "changed": true
}

```

You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

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.

```

rustom@LocalMachine:~$ ansible all -m apt -a name=vim-nox --become --ask-become-pass
BECOME password:
127.0.0.1 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694513816,
  "cache_updated": false,
  "changed": true,
  "stderr": "",
  "stderr_lines": [],
  "stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\n\nThe following additional packages will be installed:\n fonts-lato javascript-common libjs-jquery liblua5.2-0 libruby3.0 rake ruby\n ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0\n rubygems-integration 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

```

- 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?

```

rustom@LocalMachine:~$ which vim
/usr/bin/vim
rustom@LocalMachine:~$ apt search vim-box
Sorting... Done
Full Text Search... Done

```

- 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.

```

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

```

```

GNU nano 6.2 history.log
Start-Date: 2023-09-12 17:43:25
Commandline: apt install ansible
Requested-By: rustom (1000)
Install: python-babel-localedata:amd64 (2.8.0+dfsg.1-7, automatic), python3-dns>
End-Date: 2023-09-12 17:44:14

Start-Date: 2023-09-12 18:19:16
Commandline: /usr/bin/unattended-upgrade
Upgrade: thunderbird:amd64 (1:102.13.0+build1-0ubuntu0.22.04.1, 1:102.15.0+buil>
End-Date: 2023-09-12 18:19:19

Start-Date: 2023-09-12 18:19:22
Commandline: /usr/bin/unattended-upgrade
Upgrade: libjson-c5:amd64 (0.15-3~ubuntu1.22.04.1, 0.15-3~ubuntu1.22.04.2)
End-Date: 2023-09-12 18:19:23

Start-Date: 2023-09-12 18:19:26
Commandline: /usr/bin/unattended-upgrade
Upgrade: mokutil:amd64 (0.6.0-2~22.04.1, 0.6.0-2~22.04.2)
[ Read 43 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line

```

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.

```

rustom@LocalMachine:~$ sudo apt install snapd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
snapd is already the newest version (2.58+22.04.1).
snapd set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.

```

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?

```

rustom@LocalMachine:~$ ansible all -m apt -a name=snapt --become --ask-become-pass
BECOME password:
127.0.0.1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694513816,
  "cache_updated": false,
  "changed": false
}

```

3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapt 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.

```

rustom@LocalMachine:~$ ansible all -m apt -a "name=snapt state=latest" --become --ask-become-pass
BECOME password:
127.0.0.1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1694513816,
  "cache_updated": false,
  "changed": false
}

```

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

```

rustom@LocalMachine:~/CPE232_Rustom$ git add .
rustom@LocalMachine:~/CPE232_Rustom$ git commit -m "ATOM"
[main b326f6a] ATOM
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 sample1.txt
rustom@LocalMachine:~/CPE232_Rustom$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 265 bytes | 265.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:atomcarino91/CPE232_Rustom.git
 adba872..b326f6a  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:

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

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

```
rustom@LocalMachine: ~/CPE232_Rustom
GNU nano 6.2                                install_apache.yml *
--
- hosts: all
  become: tru
  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.

```
rustom@LocalMachine:~/CPE232_Rustom$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [127.0.0.1]

TASK [install apache2 package] *****
changed: [127.0.0.1]

PLAY RECAP *****
127.0.0.1          : 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.

← → ↺ 192.168.56.120 ☆



Apache2 Ubuntu 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:

× Apache2 Ubuntu Default Page × +

127.0.0.1 ☆



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
```

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?

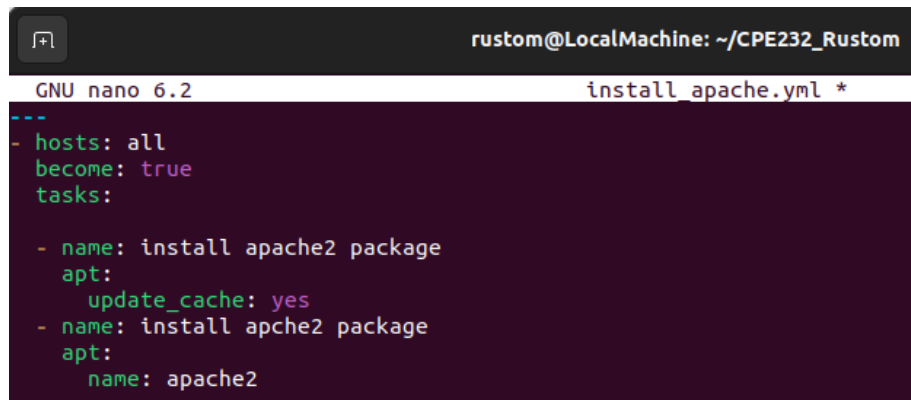
5. This time, we are going to put additional task to our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional command, which is the *update_cache*. This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities) 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.



```
rustom@LocalMachine: ~/CPE232_Rustom
GNU nano 6.2                                install_apache.yml *
---
- hosts: all
  become: true
  tasks:

    - name: install apache2 package
      apt:
        update_cache: yes
    - name: install apache2 package
      apt:
        name: apache2
```

6. 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.

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

```

rustom@LocalMachine:~/CPE232_Rustom$ ansible-playbook --ask-become-pass install_apache.yml
BECOME password:

PLAY [all] *****

TASK [Gathering Facts] *****
ok: [127.0.0.1]

TASK [install apache2 package] *****
changed: [127.0.0.1]

TASK [install apche2 package] *****
ok: [127.0.0.1]

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

```

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

```

rustom@LocalMachine:~/CPE232_Rustom$ git add install_apache.yml
rustom@LocalMachine:~/CPE232_Rustom$ git commit -m "install apache"
[main 42d2047] install apache
1 file changed, 12 insertions(+)
create mode 100644 install_apache.yml
rustom@LocalMachine:~/CPE232_Rustom$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 418 bytes | 418.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:atomcarino91/CPE232_Rustom.git
b326f6a..42d2047  main -> main

```

Reflections:

Answer the following:

1. What is the importance of using a playbook?

- **Ansible's setup, deployment, and orchestration functions are recorded and executed using playbooks. They can represent a policy that your distant systems should follow or a series of stages in a general IT procedure. Ansible is a tool for automating elements of a specific task, such as initial server setup on ubuntu. By executing a script on one or more systems, ansible playbooks are used to set up tricky system environments.**

2. Summarize what we have done on this activity.

- **In this activity we will be able to use commands that make changes to remote machines and use playbooks in automating ansible commands. We also used Ansible for managing remote systems via automation, we also installed different packages to progress in the next part of the activity. After the activity, we discover that playbooks can be used to manage configurations of and deployments to remote machines.**