| **Name:** Torrecampo, Juan Piolo S. | **Date Performed:** Sep 6, 2022 |
| --- | --- |
| **Course/Section:** CPE 232 / CPE31S22 | **Date Submitted:** Sep 6, 2022 |
| **Instructor:** Dr. Jonathan Taylar | **Semester and SY:** 1st Sem, 2022 - 2023 |
| **Activity 4: Running Elevated Ad hoc Commands** | |
| 1. **Objectives:**    1. Use commands that makes changes to remote machines    2. Use playbook in automating ansible commands | |
| 1. **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](https://docs.ansible.com/ansible/latest/user_guide/playbooks.html) | |
| **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?   * **No, it display a bunch of errors.**     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.   1. 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.      * 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, as we can see in the screenshot below when we issue the command “which vim” in different servers it displays as where the vim is installed.**   **Server 1**    **Server 2**     * 1. 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.   **Server 1**    **Server 2**     1. 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 result command is the same when you are pinging the remote servers. Yes it is a success but it does not change anything to the remote servers. Because the “snapd” is already downloaded in the remote servers. To prove my analyzation, I provided a screenshot below of history logs of server 1 and server 2.ansi**     **Server 1 - history.log**    **Server 2 - history.log**    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.   * The “state=latest” is used when we used to install the latest or update      1. At this point, make sure to commit all changes to GitHub.   **Copying the ansible directory to the CPE232\_piolo git folder.**    **Listing the contents of CPE232\_piolo**    **Committing the directory to Github**    **Github Repository** | |
| **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:    Make sure to save the file. Take note also of the alignments of the texts.     1. Run the yml file using the command: *ansible-playbook --ask-become-pass install\_apache.yml.* Describe the result of this command.      1. 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.     **Server 1**    **Server 2**     1. 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?   **Edited “install\_apache.yml” Package Name**    **Ansible Output**     * **Since the nano is installed to the remote server, it shows ok only because the software is in the latest and installed to the remote servers.**  1. 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.     Save the changes to this file and exit.     1. Run the playbook and describe the output. Did the new command change anything on the remote servers?      * **Yes, according to the “PLAY RECAP” there are 1 change in both remote servers. This is because we issue the update\_cache which updates its repository.**  1. Edit again the *install\_apache.yml*. This time, we are going to add a PHP support for the apache package we installed earlier.     Save the changes to this file and exit.     1. Run the playbook and describe the output. Did the new command change anything on the remote servers?      * **Yes, since the “libapache2-mod-php” is not installed in the system then the ansible will install this application in the 2 remote servers. Also, it will update the cache again because it is stated in the yml file.**  1. Finally, make sure that we are in sync with GitHub. Provide the link of your GitHub repository.   **Adding the install\_apache.yml file in the Github Repository**    **Github CPE232\_piolo Repository**    **Github Repository Link:** *https://github.com/piolotorrecampo/CPE232\_piolo.git* | |
| **Reflections:**  Answer the following:   * 1. What is the importance of using a playbook? * The importance of using playbook is that when we have different actions that we want to perform in the different machines simontaunely, we can issue a yml and run in ansible. This ansible feature is useful in performing different configurations in multiple servers. Also, it reduces the time for system administrators in configuring different machines.   1. Summarize what we have done on this activity. * To sum up this activity, the first part is focused on performing elevated and ad hoc commands using ansible. It uses the “–become –ask-become-pass” to elevate the privilege in performing commands in remote servers like initiating an installation of vim as I performed in task one. The task one provides the verification in the two servers, if there is a change in the system the actions are provided in the file of /etc/apt/library.log. On the other hand, task two introduces me to the yml configuration of ansible playbook. This file is a series of actions that executes when it is called by the ansible software. Also, it includes the use of reading the “PLAY RECAP” to determine if there are any changes to the remote servers. | |
| **Honor Pledge:**  *“I affirm that I will not give or receive unauthorized help on this activity and that all will be my own.*” | |

\