| | |
|---|---|
| **Name:** Bernice M. Peña | **Date Performed:** 10/17/2023 |
| **Course/Section:** Managing Enterprise Servers / CPE31S5 | **Date Submitted:** 10/17/2023 |
| **Instructor:** Engr. Roman Richard | **Semester and SY:** 1st semester, SY 2023-2024 |

**Activity 7: Managing Files and Creating Roles in Ansible**

1. **Objectives:**

1.1 Manage files in remote servers

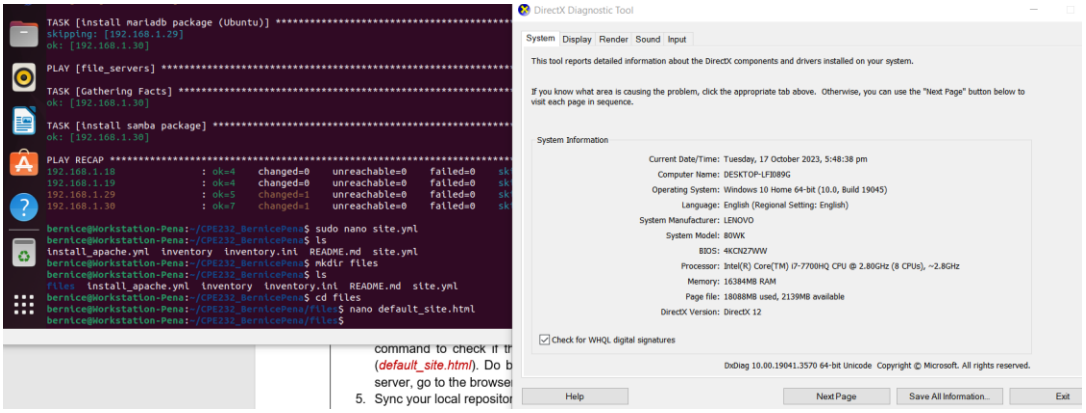
1.2 Implement roles in ansible

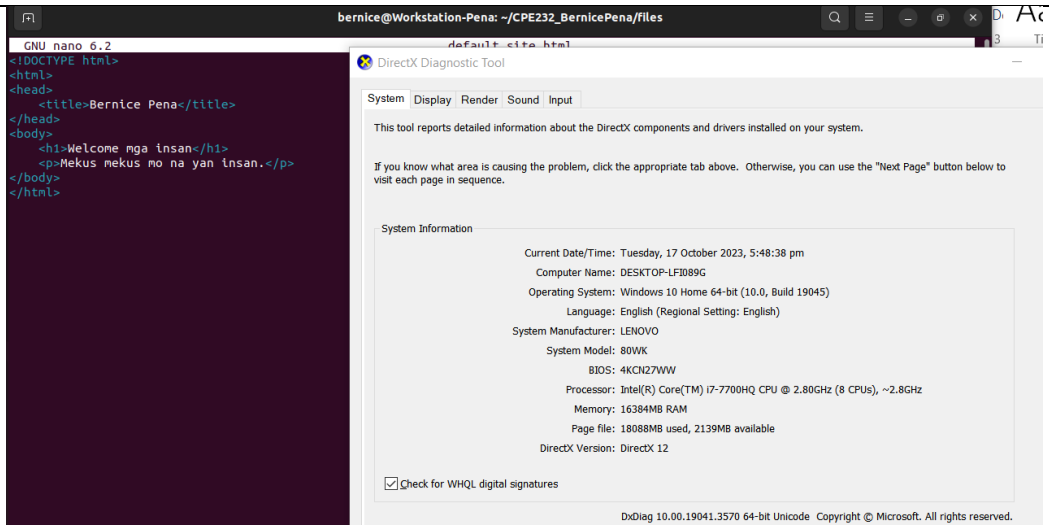
2. **Discussion**:

In this activity, we look at the concept of copying a file to a server. We are going to create a file into our git repository and use Ansible to grab that file and put it into a particular place so that we could do things like customize a default website, or maybe install a default configuration file. We will also implement roles to consolidate plays.

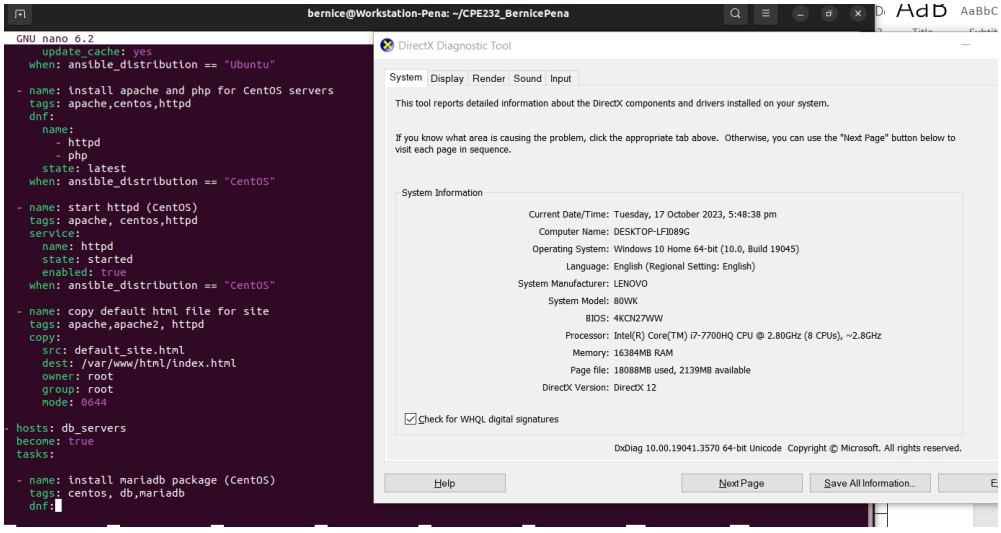
**Task 1: Create a file and copy it to remote servers**

1. Using the previous directory we created, create a directory, and named it "*files*." Create a file inside that directory and name it "*default_site.html*." Edit the file and put basic HTML syntax. Any content will do, as long as it will display text later. Save the file and exit.

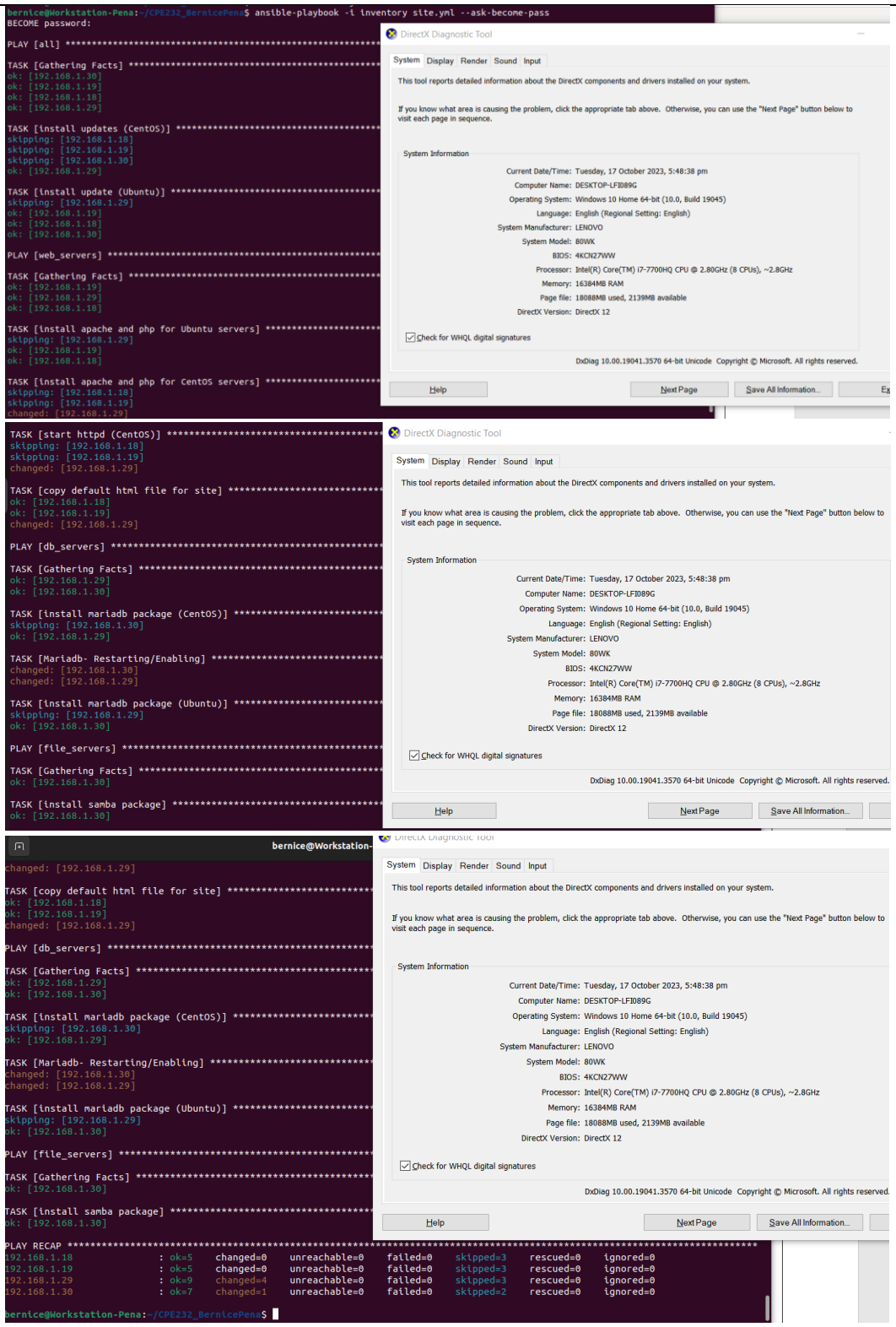





2. Edit the *site.yml* file and just below the *web_servers* play, create a new file to copy the default html file for site:
   - name: copy default html file for site
     tags: apache, apache2, httpd
     copy:
       src: default_site.html
       dest: /var/www/html/index.html
       owner: root
       group: root
       mode: 0644



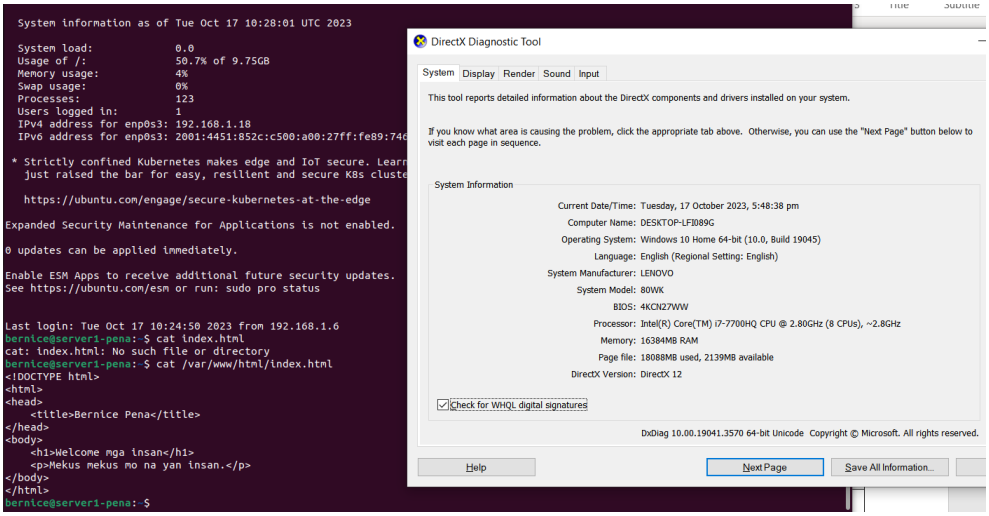

3. Run the playbook *site.yml*. Describe the changes.



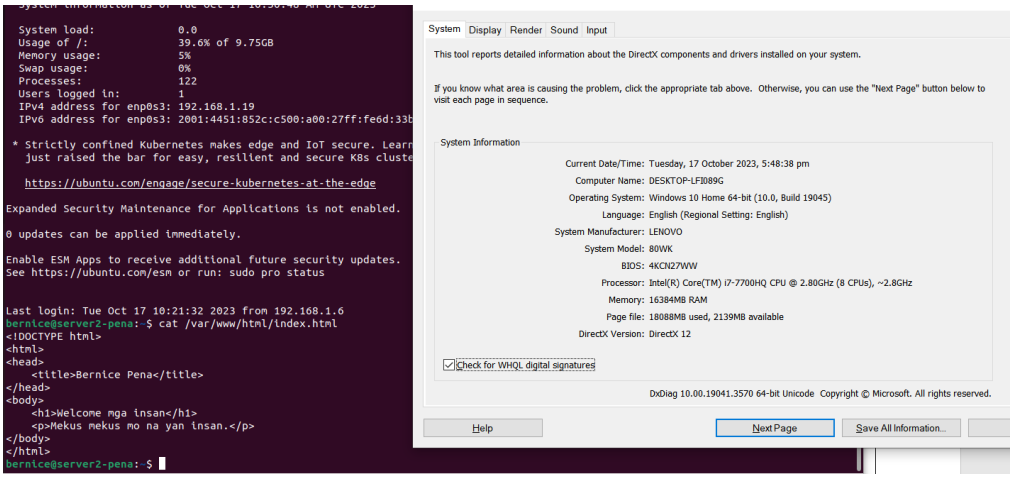

**Notice that there were changes in the recently added code for copying the default file under web_servers. The task to copy the default HTML file to the web_servers was executed leading to servers 1 and 2 to have the**

**ok status and CentOS server having the change status since I added its ip under my web_servers.**

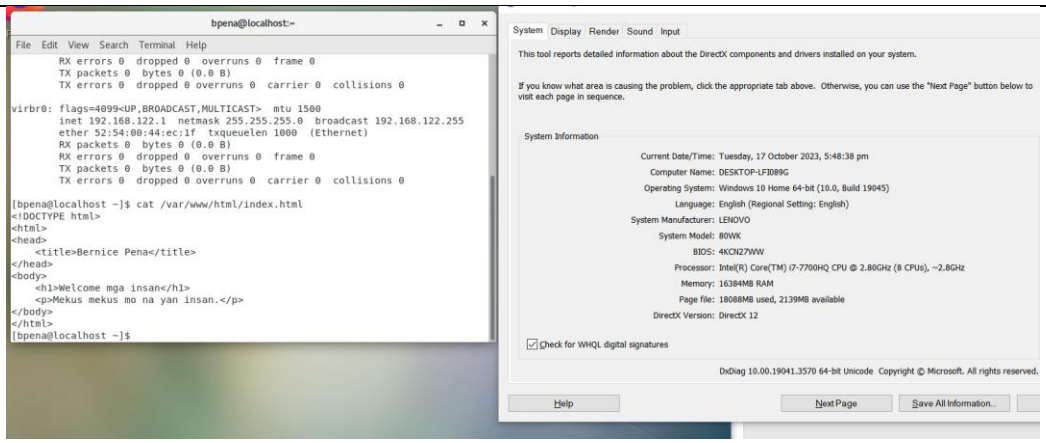
4. Go to the remote servers (*web_servers*) listed in your inventory. Use cat command to check if the index.html is the same as the local repository file (*default_site.html*). Do both for Ubuntu and CentOS servers. On the CentOS server, go to the browser and type its IP address. Describe the output.



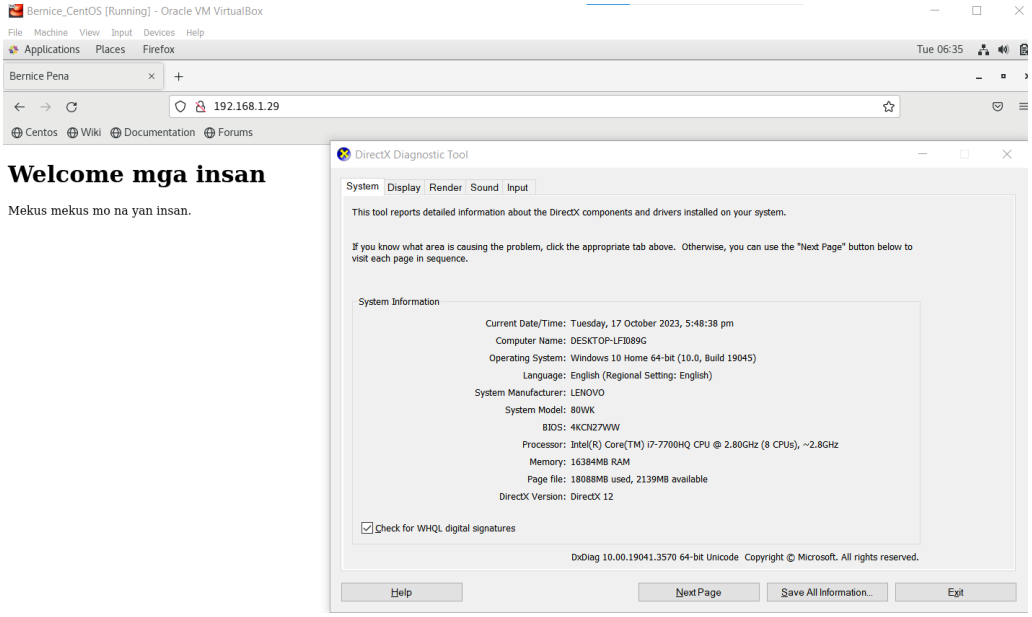

**I used ssh command to go to the remove server 1 and used the cat command to check if the index.html is the same as my default_site.html. It shows in the screenshot above that it is the same with my default_site.html.**




**I used the same process for server 2, and it shows that it is also same with my default_site.html.**

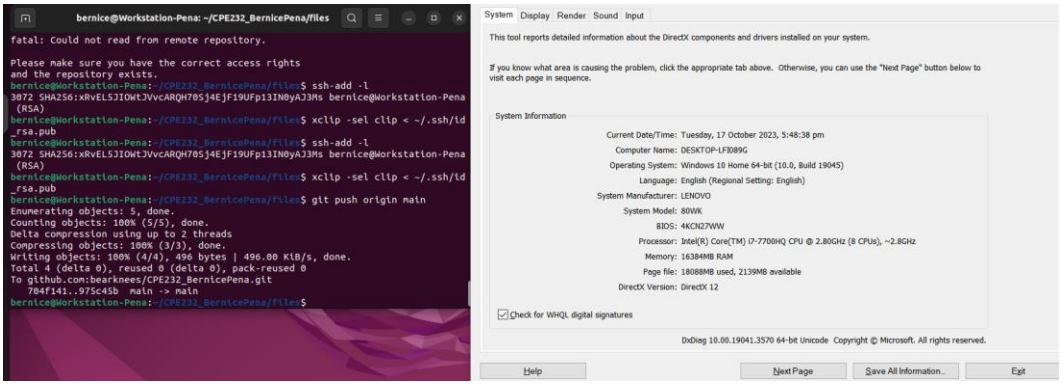



**I did the same process for my CentOS server and the result was same with my default_site.html.**

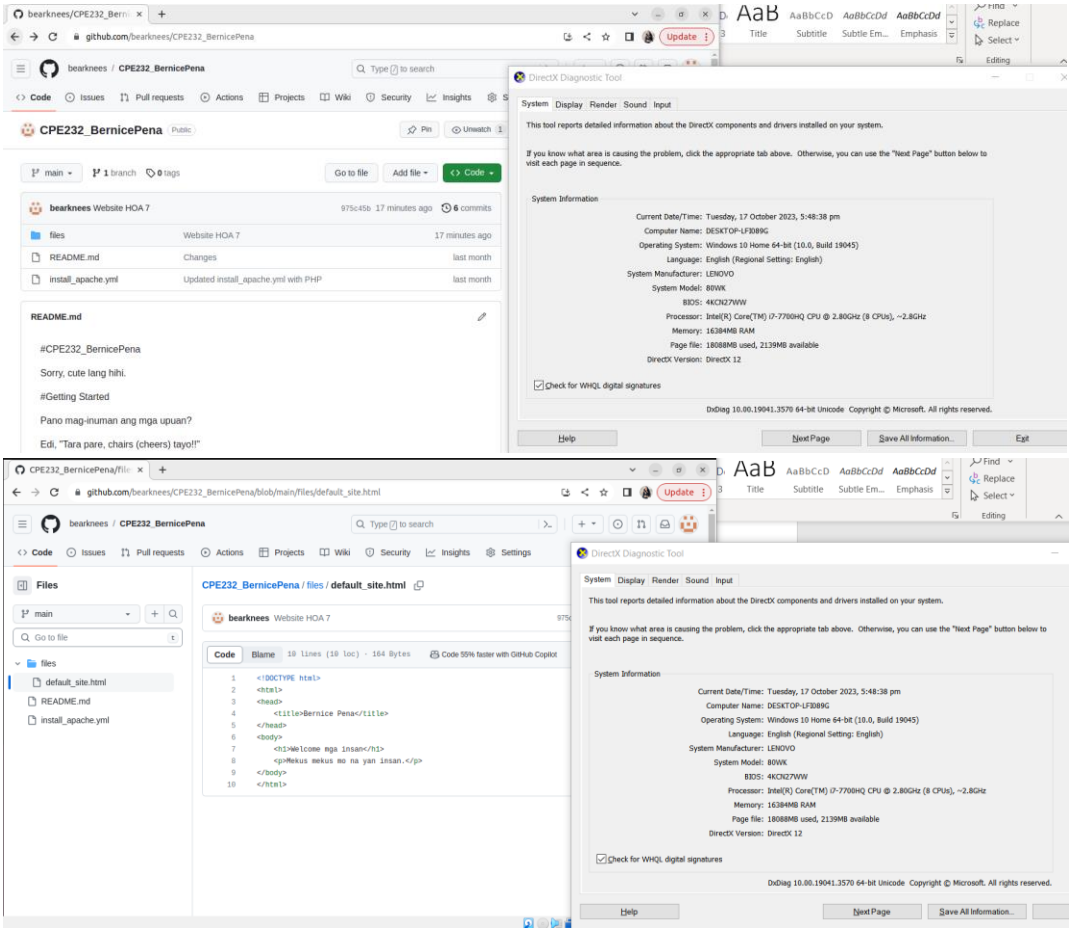



**To check it using my browser, I opened a browser on my CentOS and typed its ip address and this was the result, it's the same content as what I have in my default_site.html.**

5. Sync your local repository with GitHub and describe the changes.



**I created first a new SSH key in order to push what I committed in my GitHub.**



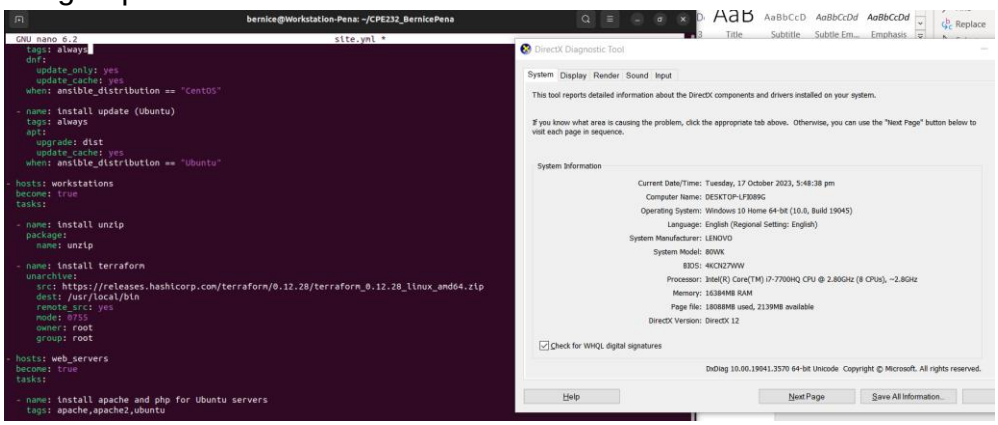

**The files directory including the contents of the directory was successfully synched in my GitHub.**

**Task 2: Download a file and extract it to a remote server**

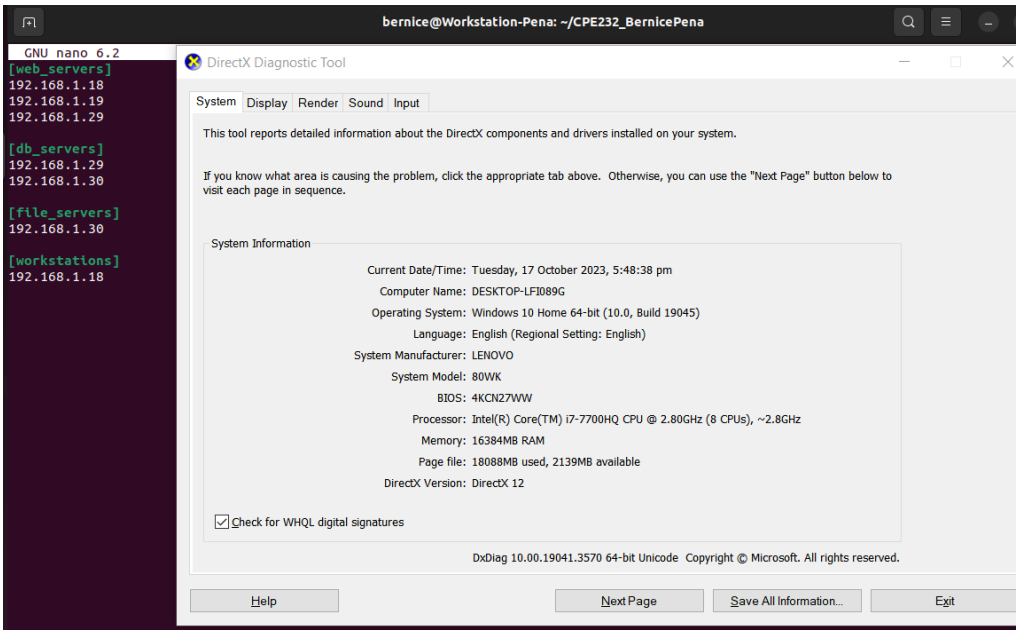
1. Edit the site.yml. Just before the web_servers play, create a new play:

```
- hosts: workstations
  become: true
  tasks:

  - name: install unzip
    package:
      name: unzip

  - name: install terraform
    unarchive:
```

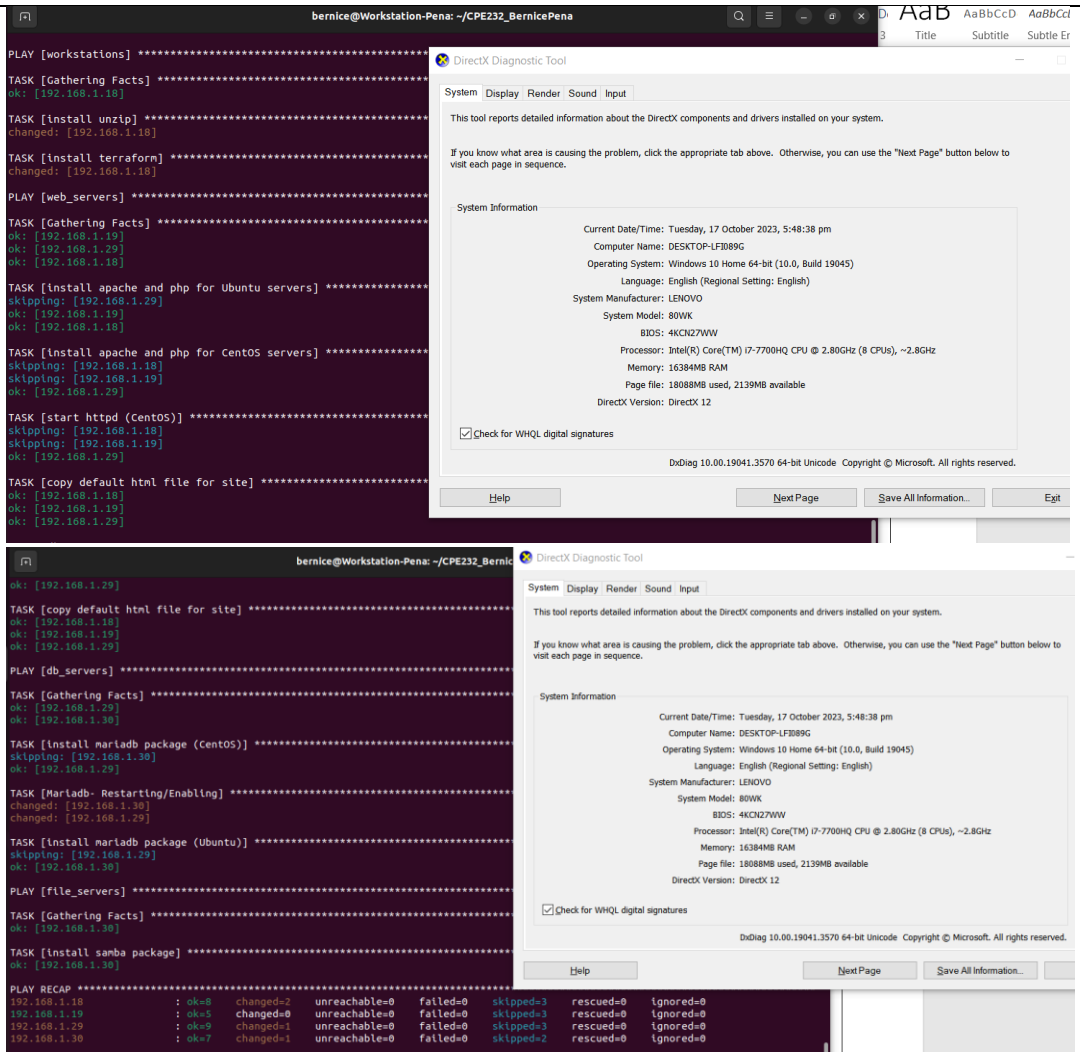
src:
https://releases.hashicorp.com/terraform/0.12.28/terraform_0.12.28_linux_amd64.zip
dest: /usr/local/bin
remote_src: yes
mode: 0755
owner: root
group: root




2. Edit the inventory file and add workstations group. Add any Ubuntu remote server. Make sure to remember the IP address.

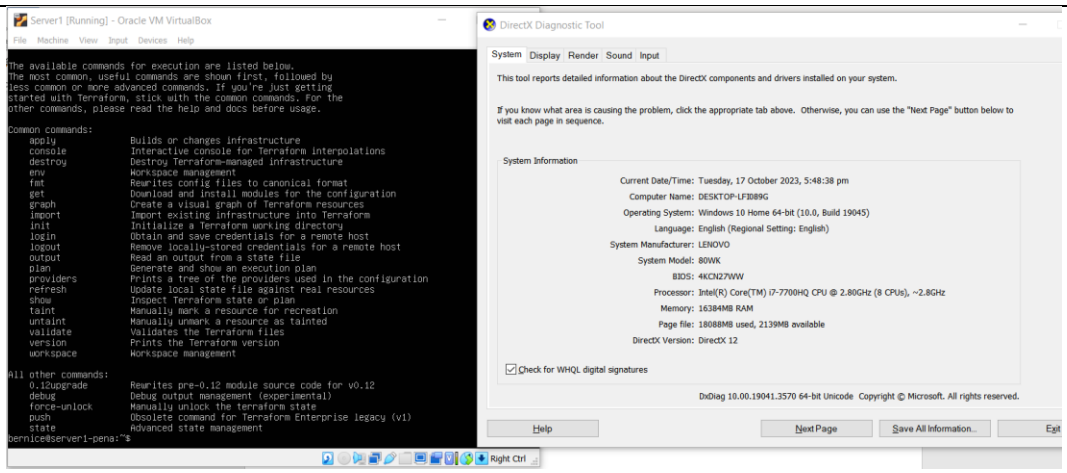



3. Run the playbook. Describe the output.




**As you can see in my recently added play named workstations, it read my server 1 ip address since that was the address that I added in my recently added group named workstations in my inventory. There was also a change happened in that server since the code is meant to make changes for that server.**

4. On the Ubuntu remote workstation, type terraform to verify installation of terraform. Describe the output.



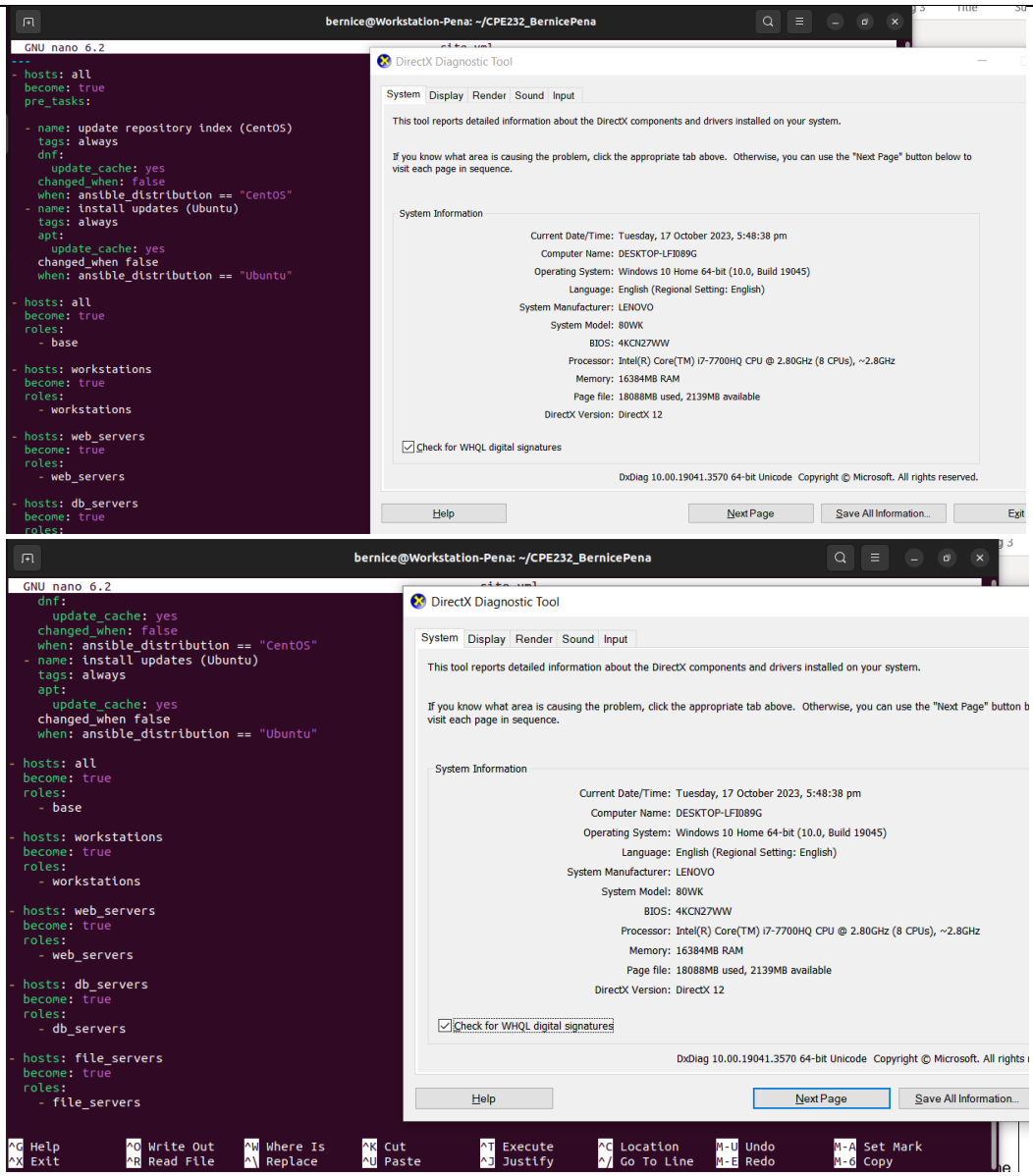

**After executing the command to verify the installation of terraform, the usage of terraform was shown including the commands for execution list.**

**Task 3: Create roles**

1. Edit the site.yml. Configure roles as follows: (make sure to create a copy of the old site.yml file because you will be copying the specific plays for all groups)

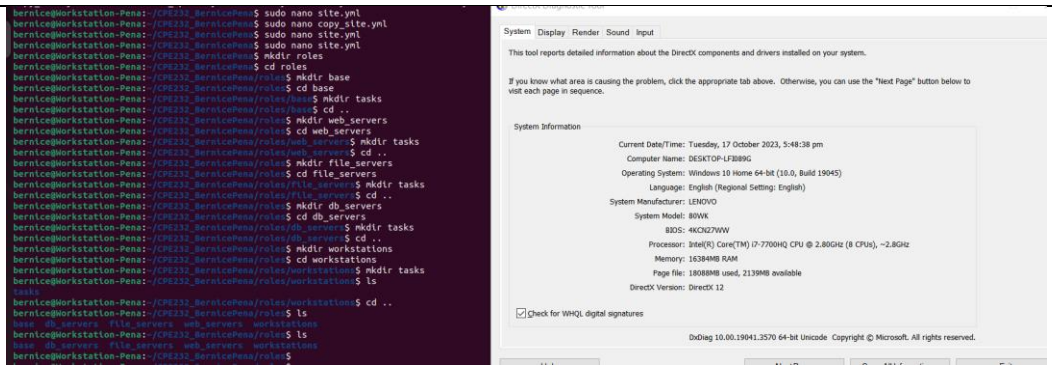
```
---
- hosts: all
  become: true
  pre_tasks:

  - name: update repository index (CentOS)
    tags: always
    dnf:
      update_cache: yes
    changed_when: false
    when: ansible_distribution == "CentOS"
  - name: install updates (Ubuntu)
    tags: always
    apt:
      update_cache: yes
    changed_when: false
    when: ansible_distribution == "Ubuntu"

- hosts: all
  become: true
  roles:
    -  base

- hosts: workstations
  become: true
  roles:
    - workstations

- hosts: web_servers
  become: true
  roles:
    - web_servers

- hosts: db_servers
  become: true
  roles:
    - db_servers

- hosts: file_servers
  become: true
  roles:
    - file_servers
```

Save the file and exit.



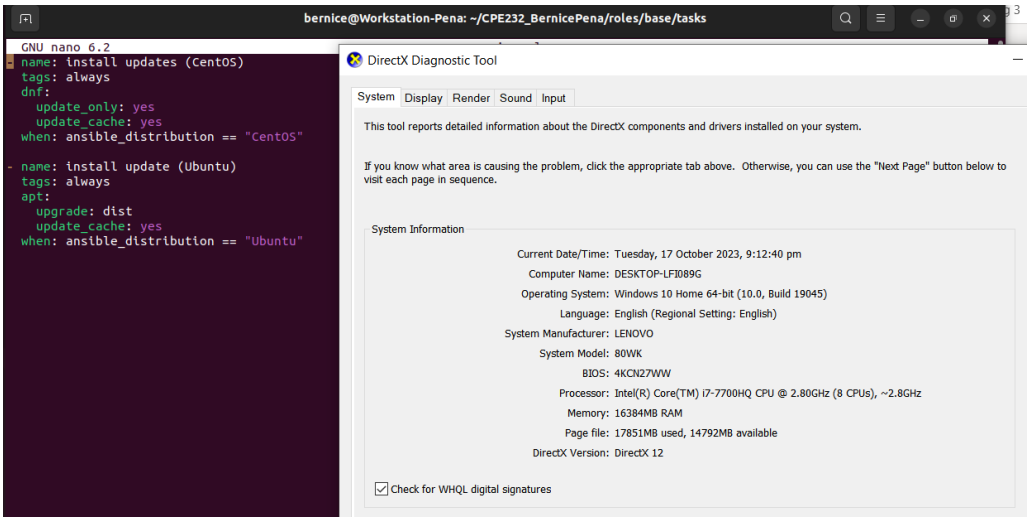

```
---
- hosts: all
  become: true
  pre_tasks:

  - name: update repository index (CentOS)
    tags: always
    dnf:
      update_cache: yes
    changed_when: false
    when: ansible_distribution == "CentOS"
  - name: install updates (Ubuntu)
    tags: always
    apt:
      update_cache: yes
    changed_when false
    when: ansible_distribution == "Ubuntu"

- hosts: all
  become: true
  roles:
    - base

- hosts: workstations
  become: true
  roles:
    - workstations

- hosts: web_servers
  become: true
  roles:
    - web_servers

- hosts: db_servers
  become: true
  roles:
```

2. Under the same directory, create a new directory and name it roles. Enter the roles directory and create new directories: base, web_servers, file_servers, db_servers and workstations. For each directory, create a directory and name it tasks.

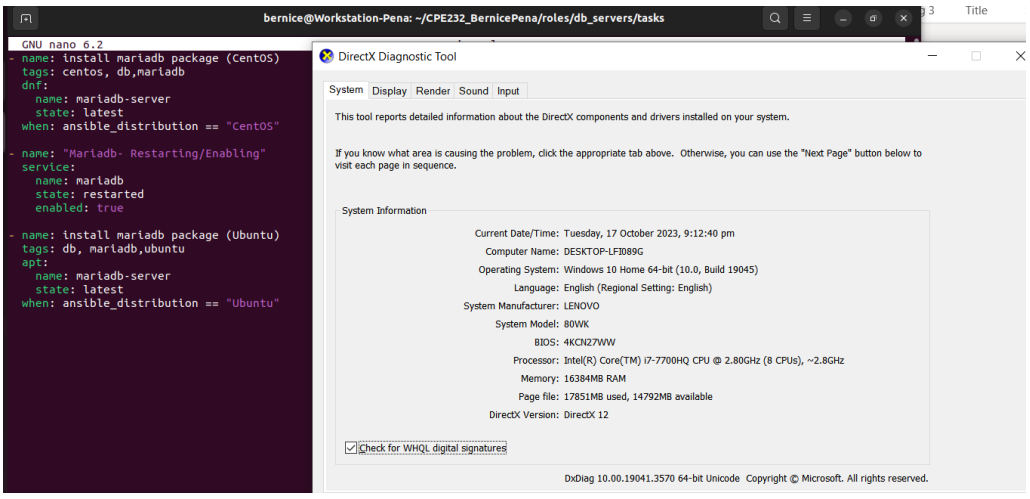



3. Go to tasks for all directory and create a file. Name it main.yml. In each of the tasks for all directories, copy and paste the code from the old site.yml file. Show all contents of main.yml files for all tasks.

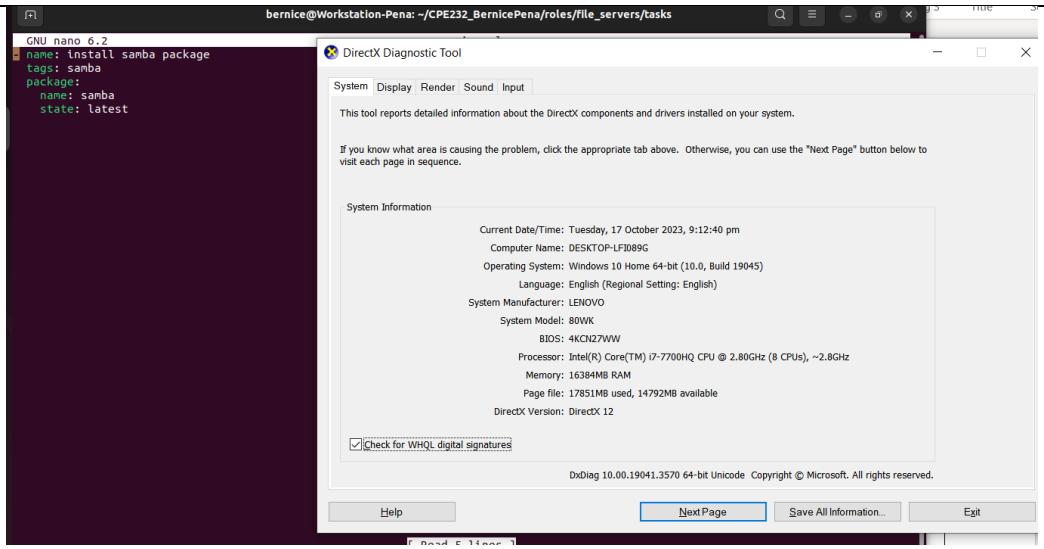
**base:**



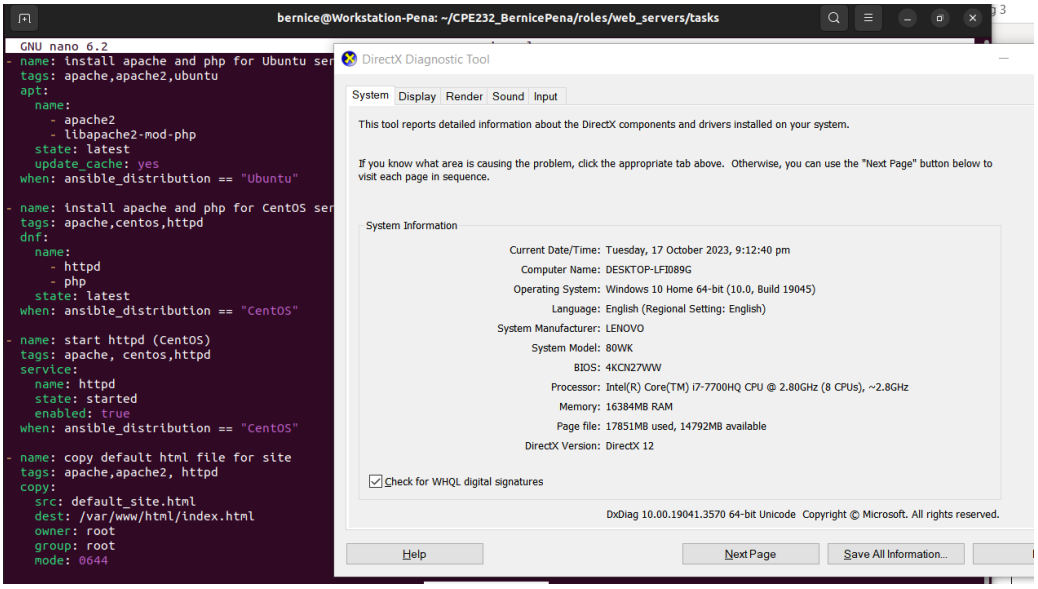

**db_servers:**



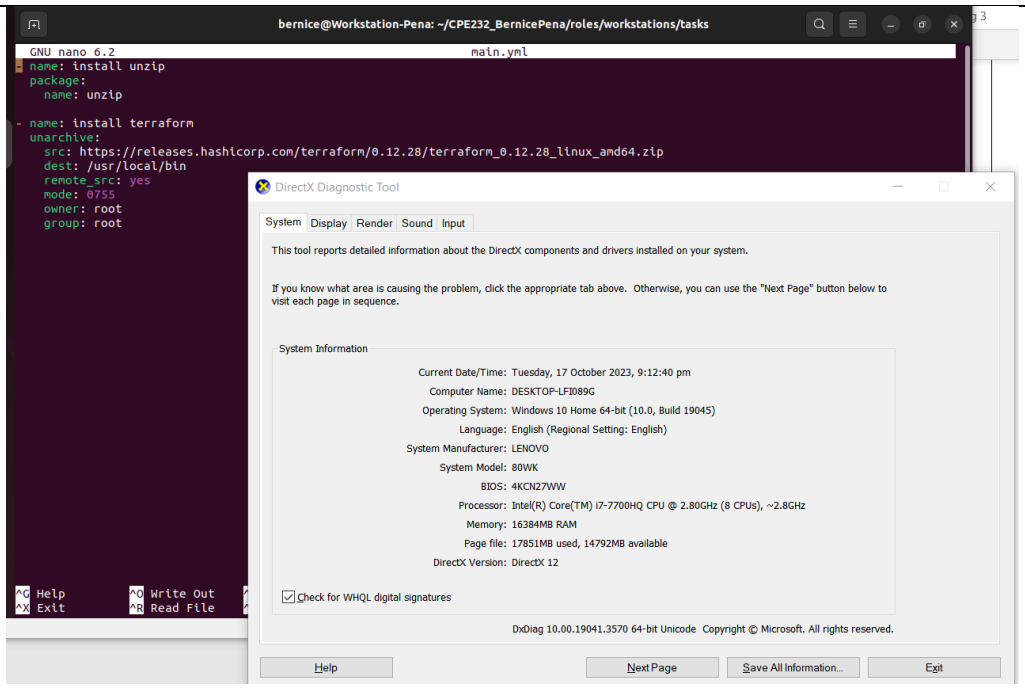

**file_servers:**




## web_servers:

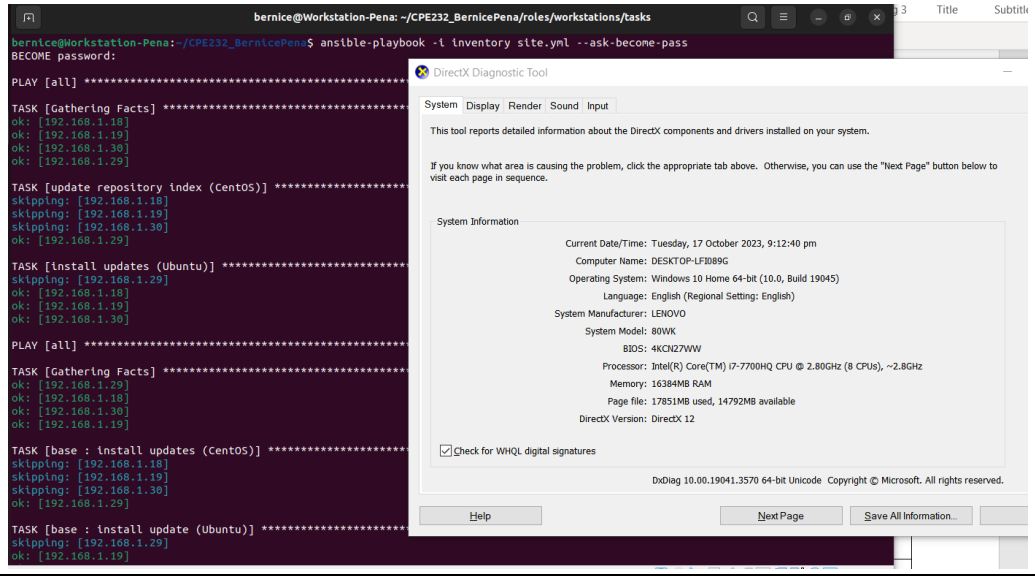


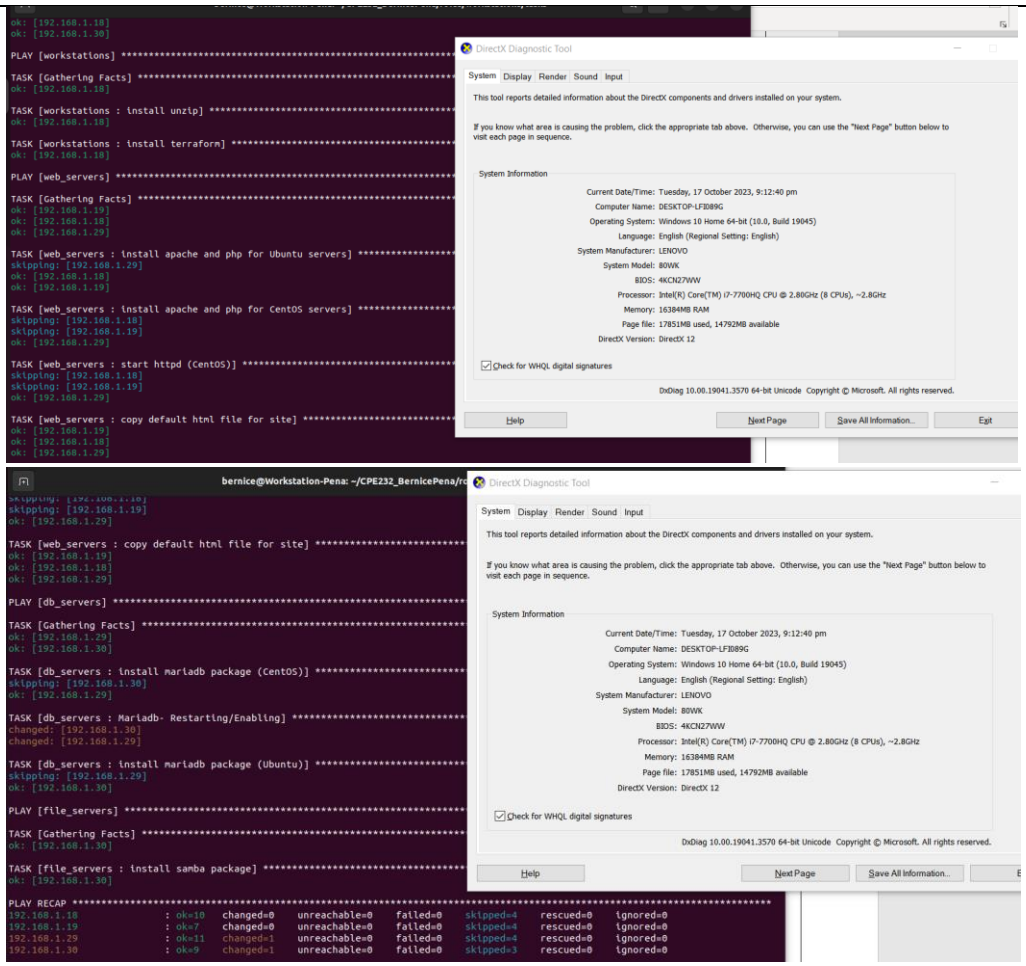

## workstations:




**What I did for each main.yml is that I copied the tasks meant for each group, I have groups such as base, db_servers, file_servers, web_servers, workstations. Each of this have their own tasks in the old site.yml, I copied the tasks for each group and paste it on the appropriate yml.**

4. Run the site.yml playbook and describe the output.






**The plays were executed successfully without any fails. Even though I removed all the tasks from the site.yml and just replace it with the use of roles command, the tasks were still executed successfully.**

**Reflections:**

Answer the following:

1. What is the importance of creating roles?

   **Creating roles makes it easier and less complex for users when managing and organizing multiple tasks with multiple groups. This helps in breaking down complex automations into smaller parts making it organized just like what I did in the previous tasks, I removed all the tasks written in my old site.yml and then created tasks files that only contains tasks for a specific group. With the use of roles, it makes it easier to identify and resolve issues, instead of analyzing the whole tasks in a single yml, you can just navigate to a specific task inside of a specific directory to fix errors, the separation of tasks allows simplifying the debugging process.**

2. What is the importance of managing files?

**Managing files is important when maintaining organized and efficient system, it ensures integrity as well as implementing backup files, this helps when navigating specific files, it is basically a time-saver for the users to locate specific files when needed. A well-organized file provides conveniency and less prone to data loss.**