Name: Shaniah Rose Hope M. Sumaoang	**Date Performed: October 10, 2023**
Course/Section: CPE 232-CPE31S5	**Date Submitted: October 14, 2023**
Instructor: Engr. Roman Richard	**Semester and SY: 1st Semester SY 2023-2024**

Activity 6: Targeting Specific Nodes and Managing Services

1. **Objectives:**

1.1 Individualize hosts
1.2 Apply tags in selecting plays to run
1.3 Managing Services from remote servers using playbooks

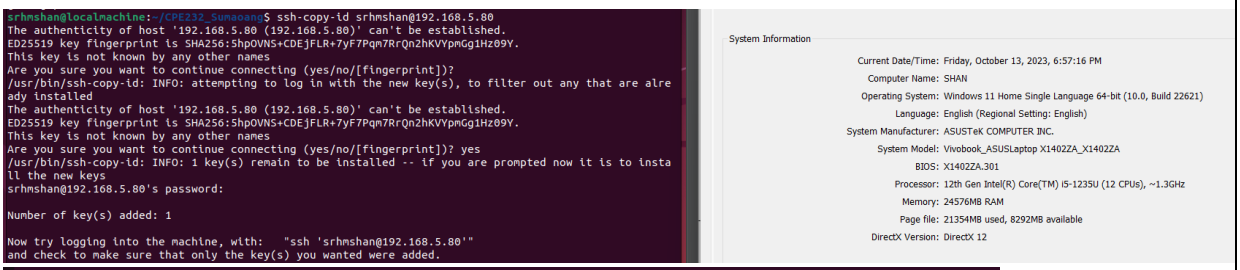
2. **Discussion**:

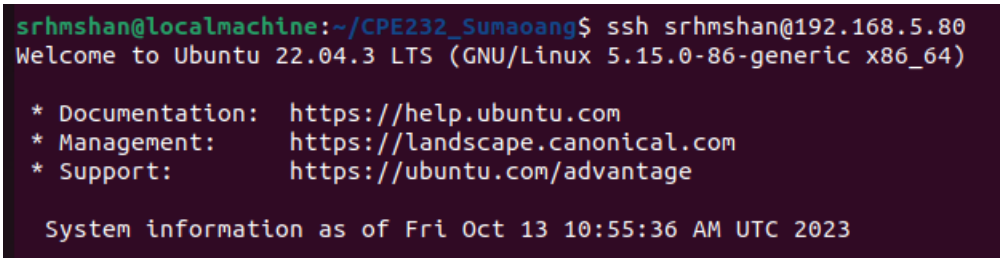
In this activity, we try to individualize hosts. For example, we don't want apache on all our servers, or maybe only one of our servers is a web server, or maybe we have different servers like database or file servers running different things on different categories of servers and that is what we are going to take a look at in this activity.

We also try to manage services that do not automatically run using the automations in playbook. For example, when we install web servers or httpd for CentOS, we notice that the service did not start automatically.

Requirement:

In this activity, you will need to create another Ubuntu VM and name it Server 3. Likewise, you need to activate the second adapter to a host-only adapter after the installations. Take note of the IP address of the Server 3. Make sure to use the command *ssh-copy-id* to copy the public key to Server 3. Verify if you can successfully SSH to Server 3.





I created my server 3 then used the command “ssh-copy-id” in my local machine to copy the public key to server 3.

Task 1: Targeting Specific Nodes

1. Create a new playbook and named it site.yml. Follow the commands as shown in the image below. Make sure to save the file and exit.

```
---

- hosts: all
  become: true
  tasks:

  - name: install apache and php for Ubuntu servers
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install apache and php for CentOS servers
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "CentOS"
```

srhmshan@localmachine: ~/CPE

```
GNU nano 6.2                                site.yml
---

- hosts: all
  become: true
  tasks:

  - name: install apache and php for Ubuntu servers
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install apache and php for CentOS servers
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "CentOS"
```

DirectX Diagnostic Tool

System | Display | Sound | Input

This tool reports detailed information about the DirectX components and drivers installed on your system.

If you know what area is causing the problem, click the appropriate tab above. Otherwise, you can use the "Next Page" visit each page in sequence.

System Information

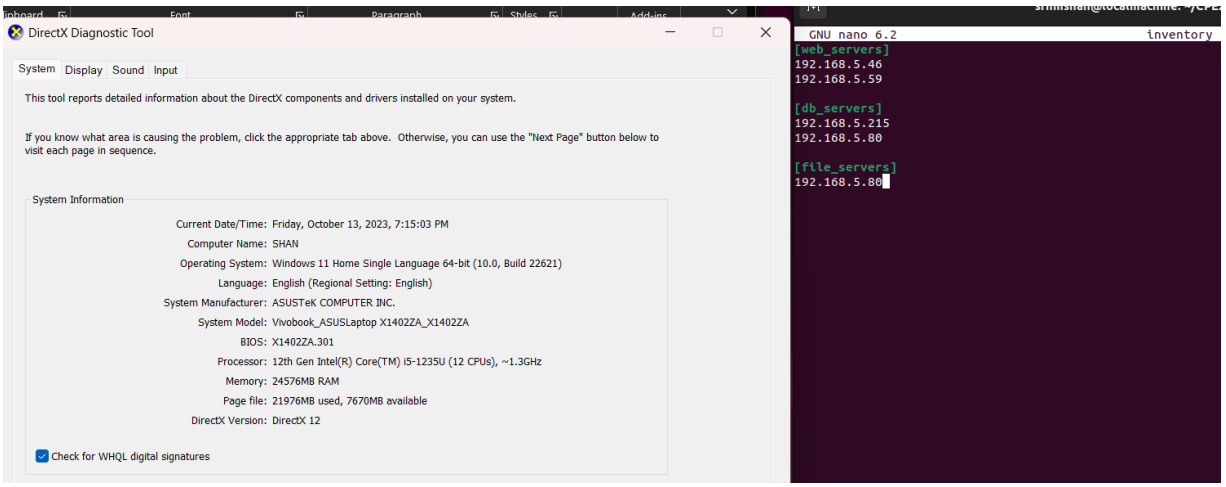
Current Date/Time:	Tuesday, October 10, 2023, 8:59:48 PM
Computer Name:	SHAN
Operating System:	Windows 11 Home Single Language 64-bit (10.0, Build 22621)
Language:	English (Regional Setting: English)
System Manufacturer:	ASUSTeK COMPUTER INC.
System Model:	Vivobook_ASUSLaptop X1402ZA_X1402ZA
BIOS:	X1402ZA.301
Processor:	12th Gen Intel(R) Core(TM) i5-1235U (12 CPUs), ~1.3GHz
Memory:	24576MB RAM
Page file:	12092MB used, 17555MB available
DirectX Version:	DirectX 12

☑ Check for WHQL digital signatures

2. Edit the inventory file. Remove the variables we put in our last activity and group according to the image shown below:

```
[web_servers]
192.168.56.120
192.168.56.121

[db_servers]
192.168.56.122


[file_servers]
192.168.56.123
```

Make sure to save the file and exit.



I assigned the following for the servers in the inventory:

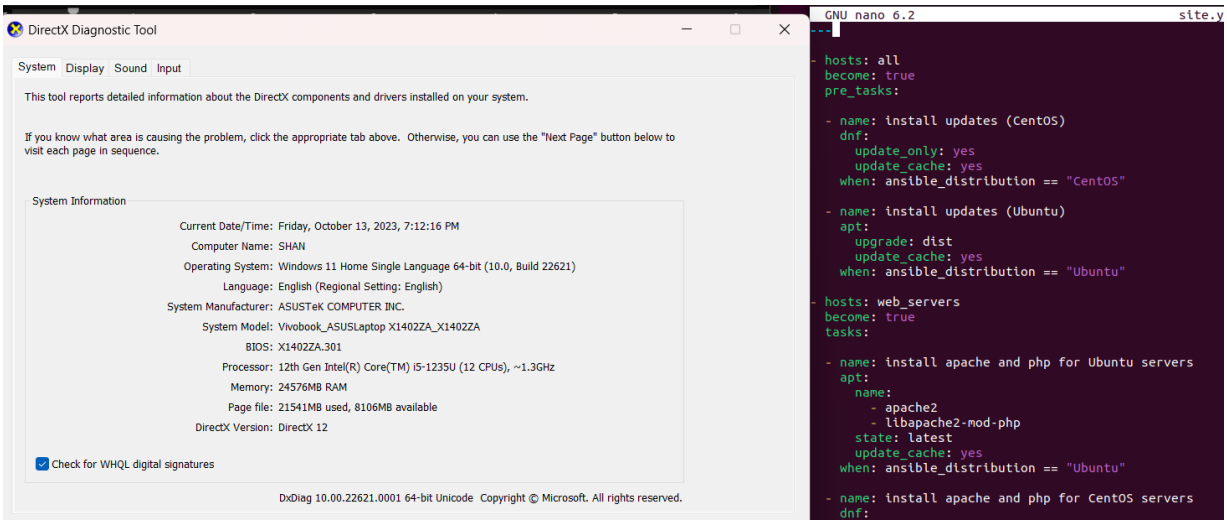
- **Web servers = Ubuntu Servers 1 and 2**
- **DB servers = CentOS and Ubuntu Server 3**
- **File servers = Ubuntu Server 3**

Right now, we have created groups in our inventory file and put each server in its own group. In other cases, you can have a server be a member of multiple groups, for example you have a test server that is also a web server.

3. Edit the *site.yml* by following the image below:

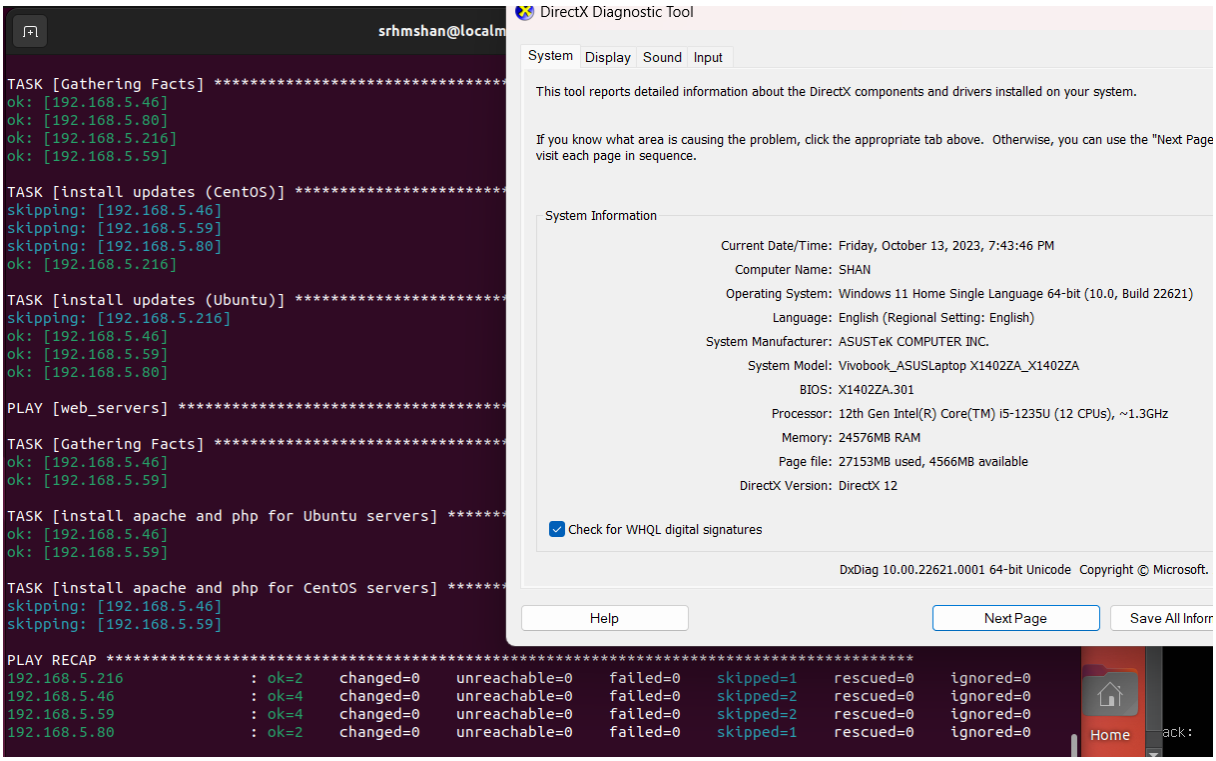
```
---

- hosts: all
  become: true
  pre_tasks:

  - name: install updates (CentOS)
    dnf:
      update_only: yes
      update_cache: yes
    when: ansible_distribution == "CentOS"

  - name: install updates (Ubuntu)
    apt:
      upgrade: dist
      update_cache: yes
    when: ansible_distribution == "Ubuntu"


- hosts: web_servers
  become: true
  tasks:

  - name: install apache and php for Ubuntu servers
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: install apache and php for CentOS servers
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "CentOS"
```

Make sure to save the file and exit.



The *pre-tasks* command tells the ansible to run it before any other thing. In the *pre-tasks*, CentOS will install updates while Ubuntu will upgrade its distribution package. This will run before running the second play, which is targeted at *web_servers*. In the second play, apache and php will be installed on both Ubuntu servers and CentOS servers.

Run the *site.yml* file and describe the result.

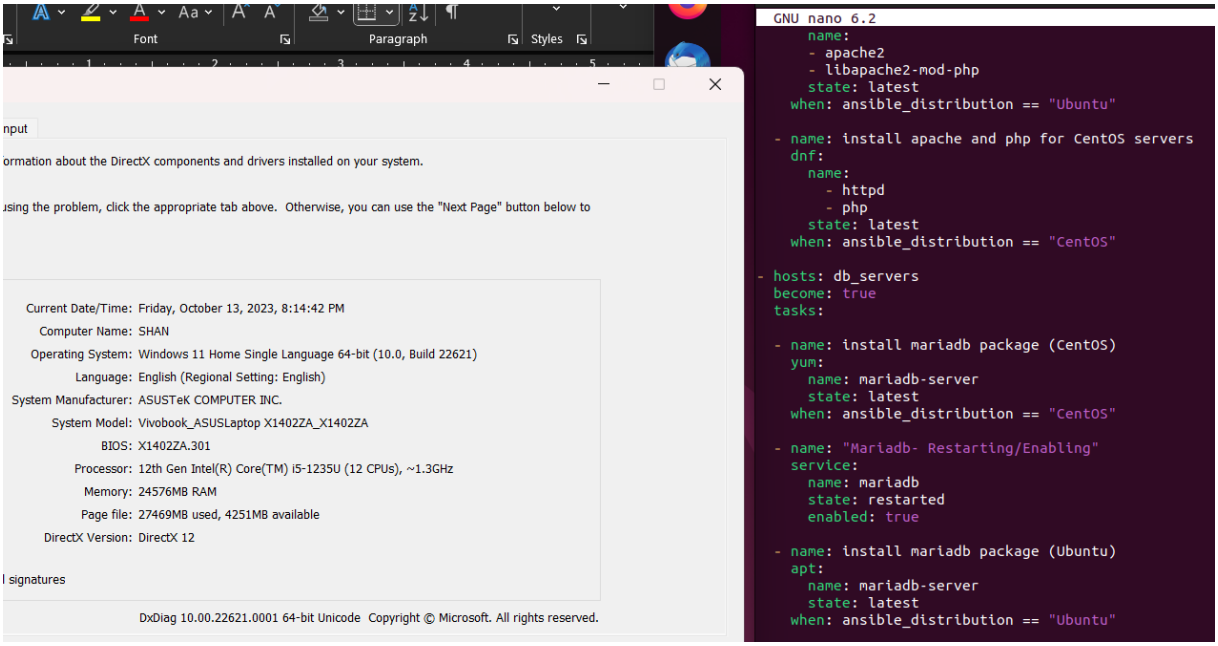


There are different tasks for a specific host, and the results show how these tasks were done successfully. To gather these tasks, it starts by looking at the IP addresses in my inventory file. For instance, when it comes to the "install updates" task for Ubuntu, it skips the CentOS IP addresses because this task is only for Ubuntu servers. The same goes for CentOS updates. It follows this pattern for other tasks too--if it's meant for Ubuntu servers, it will skip the tasks assigned for CentOS; if it's for CentOS, it skips the tasks assigned for Ubuntu.

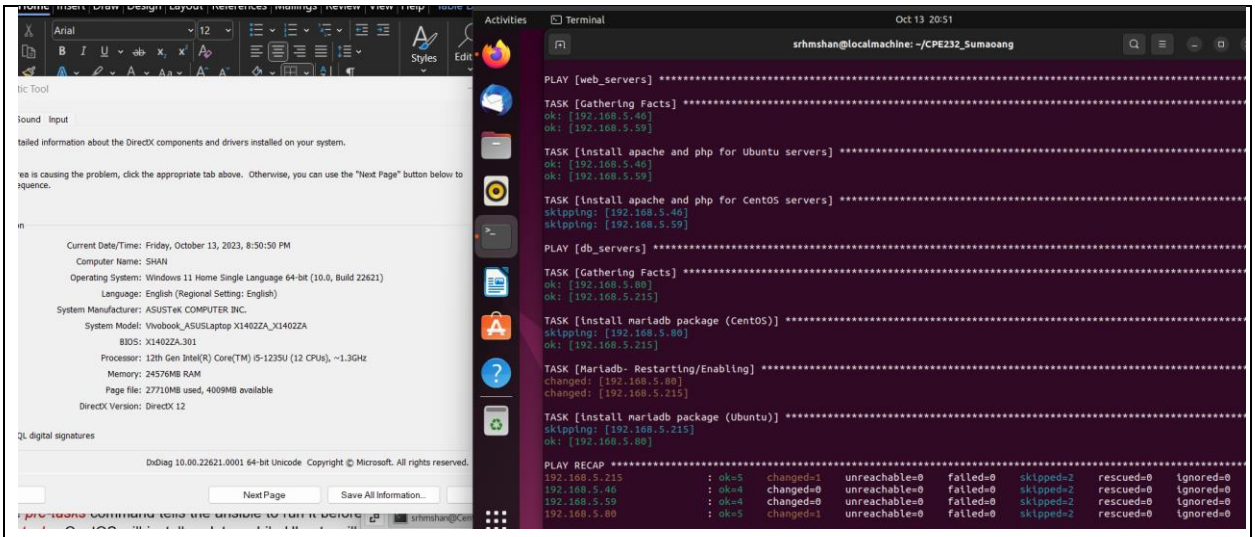
4. Let's try to edit again the *site.yml* file. This time, we are going to add plays targeting the other servers. This time we target the *db_servers* by adding it on the current *site.yml*. Below is an example: (Note add this at the end of the playbooks from task 1.3.

```yaml
- hosts: db_servers
  become: true
  tasks:

  - name: install mariadb package (CentOS)
    yum:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "CentOS"

  - name: "Mariadb- Restarting/Enabling"
    service:
      name: mariadb
      state: restarted
      enabled: true

  - name: install mariadb packege (Ubuntu)
    apt:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "Ubuntu"
```

Make sure to save the file and exit.

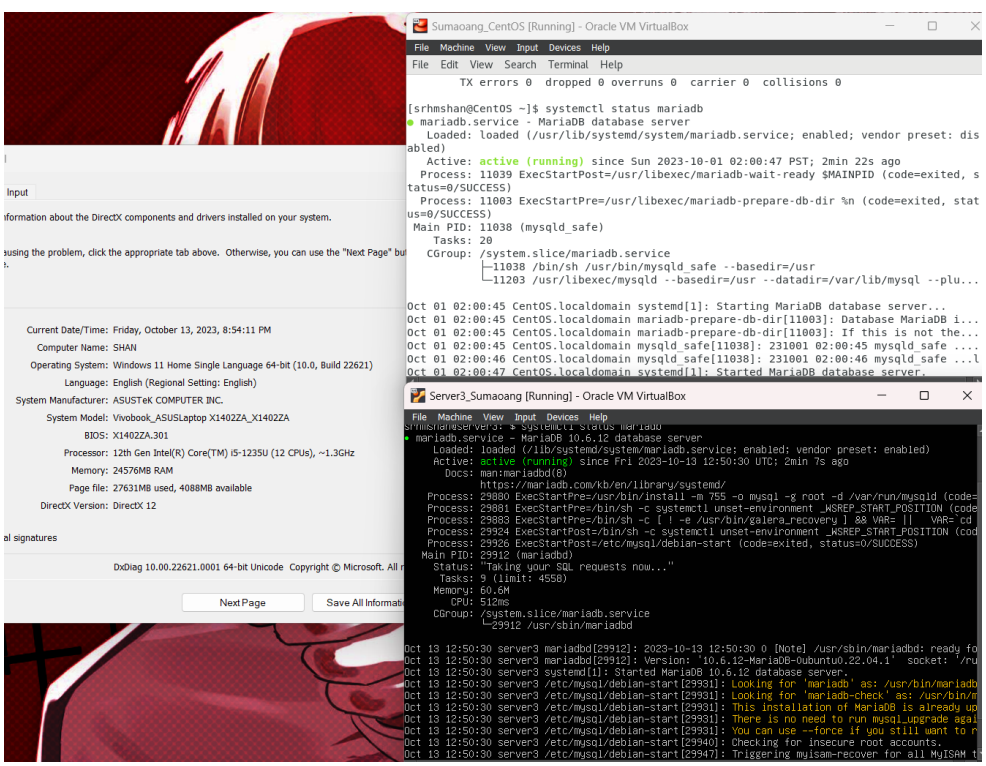


Run the *site.yml* file and describe the result.



I've edited the yaml file and added another play for "[db_servers]" specifically for installing the MariaDB package. As mentioned, I've assigned my CentOS IP address and Server 3 to the "db_servers" group. So, when tasks are collected, it only considers the IP addresses listed within the "db_servers" group.

5. Go to the remote server (Ubuntu) terminal that belongs to the db_servers group and check the status for mariadb installation using the command: *systemctl status mariadb.* Do this on the CentOS server also.



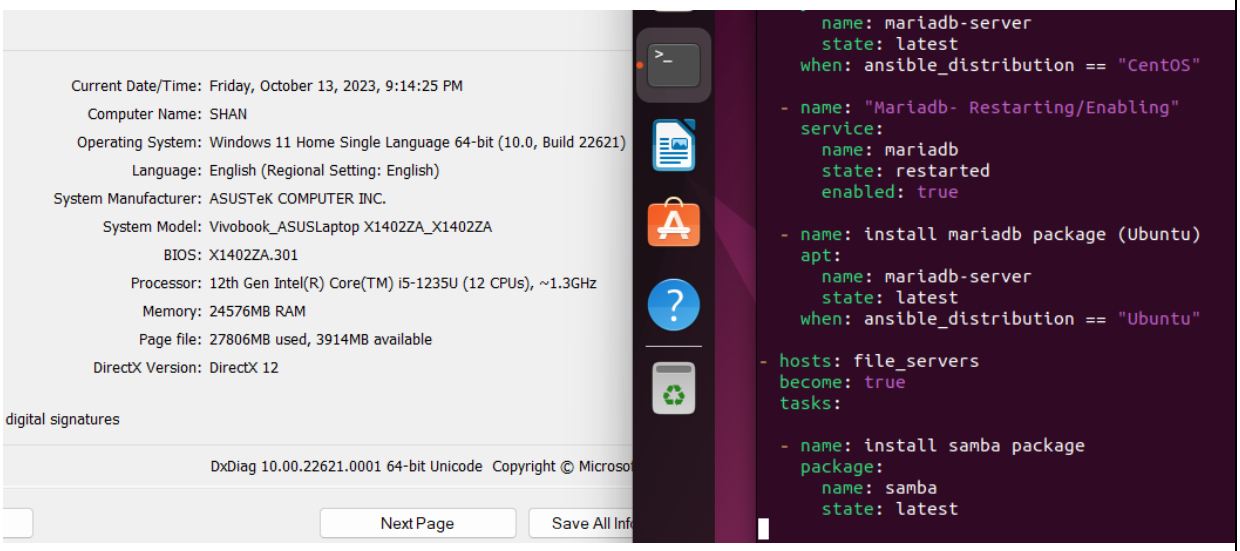
Describe the output.

When I executed the command, I noticed that the Active status was "active (running)". This means that the installation of the MariaDB package was successful for both Ubuntu and CentOS DB servers.

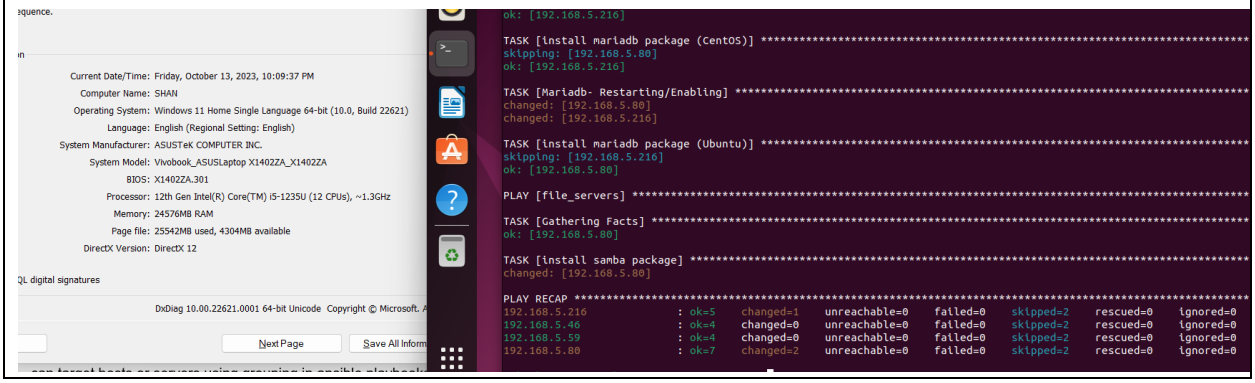
6. Edit the *site.yml* again. This time we will append the code to configure installation on the *file_servers* group. We can add the following on our file.

```
- hosts: file_servers
  become: true
  tasks:

  - name: install samba package
    package:
      name: samba
      state: latest
```

Make sure to save the file and exit.



Run the *site.yml* file and describe the result.



Since I assigned my Ubuntu Server 3 to “file_servers", it only directed the installation to Ubuntu Server 3. If I were to change the IP address and assign the CentOS server, the Samba package will be installed to the CentOS server.

The testing of the *file_servers* is beyond the scope of this activity, and as well as our topics and objectives. However, in this activity we were able to show that we can target hosts or servers using grouping in ansible playbooks.

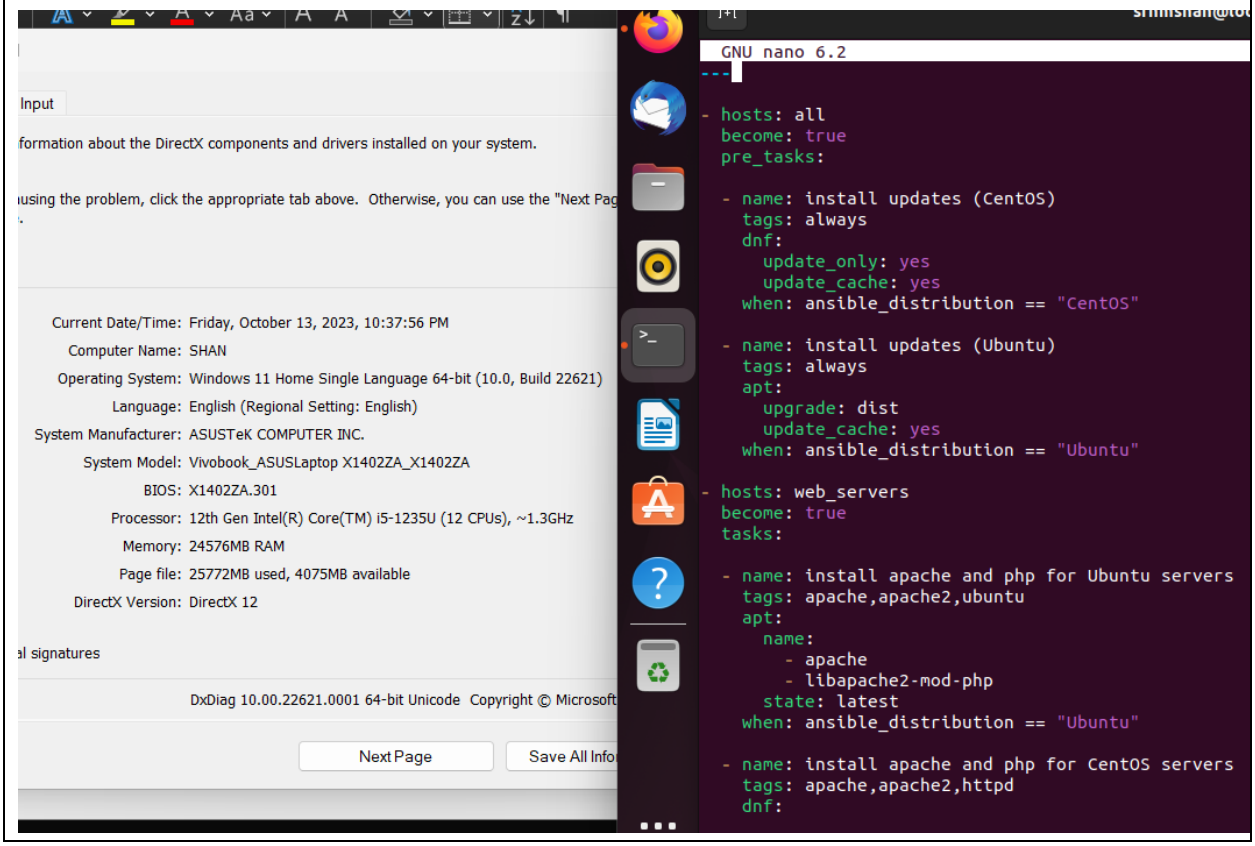
Task 2: Using Tags in running playbooks

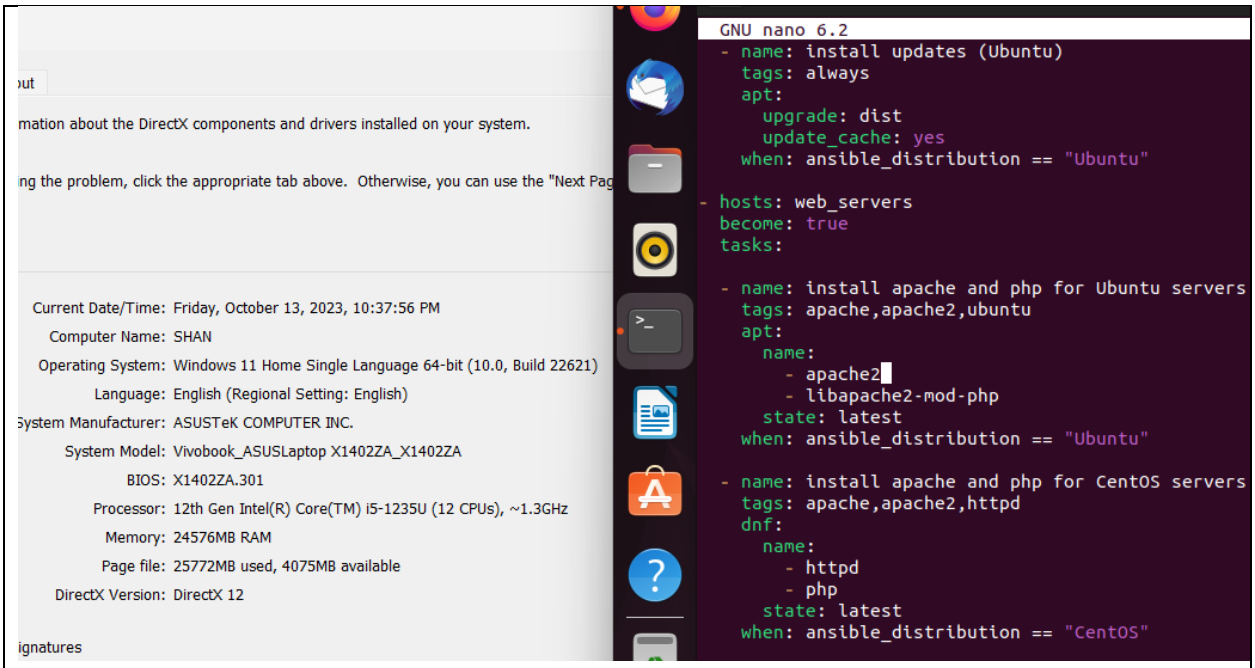
In this task, our goal is to add metadata to our plays so that we can only run the plays that we want to run, and not all the plays in our playbook.

1. Edit the *site.yml* file. Add tags to the playbook. After the name, we can place the tags: *name_of_tag*. This is an arbitrary command, which means you can use any name for a tag.

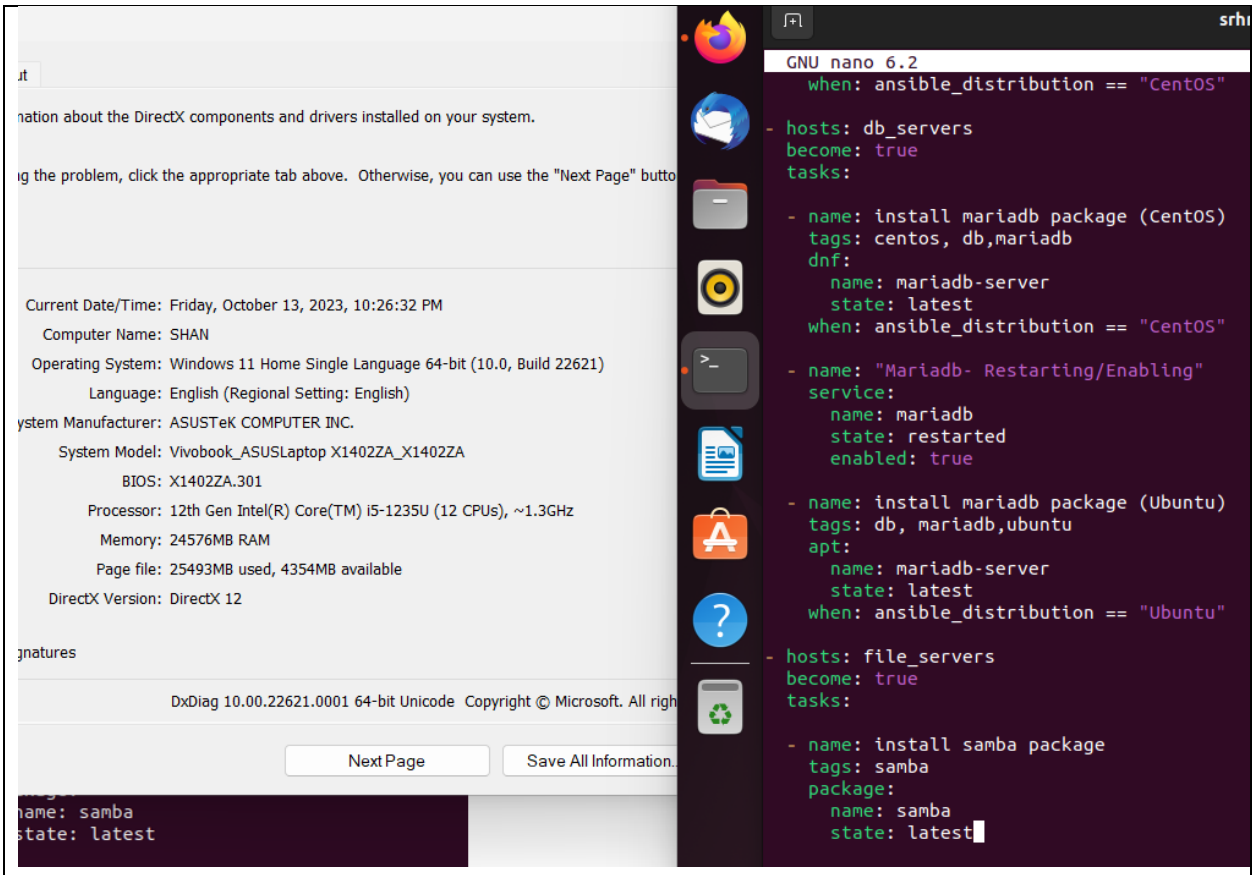
```
---

- hosts: all
  become: true
  pre_tasks:

  - name: install updates (CentOS)
    tags: always
    dnf:
      update_only: yes
      update_cache: yes
    when: ansible_distribution == "CentOS"

  - name: install updates (Ubuntu)
    tags: always
    apt:
      upgrade: dist
      update_cache: yes
    when: ansible_distribution == "Ubuntu"
```

```yaml
- hosts: web_servers
  become: true
  tasks:

  - name: install apache and php for Ubuntu servers
    tags: apache,apache2,ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: install apache and php for CentOS servers
    tags: apache,centos,httpd
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "CentOS"
```



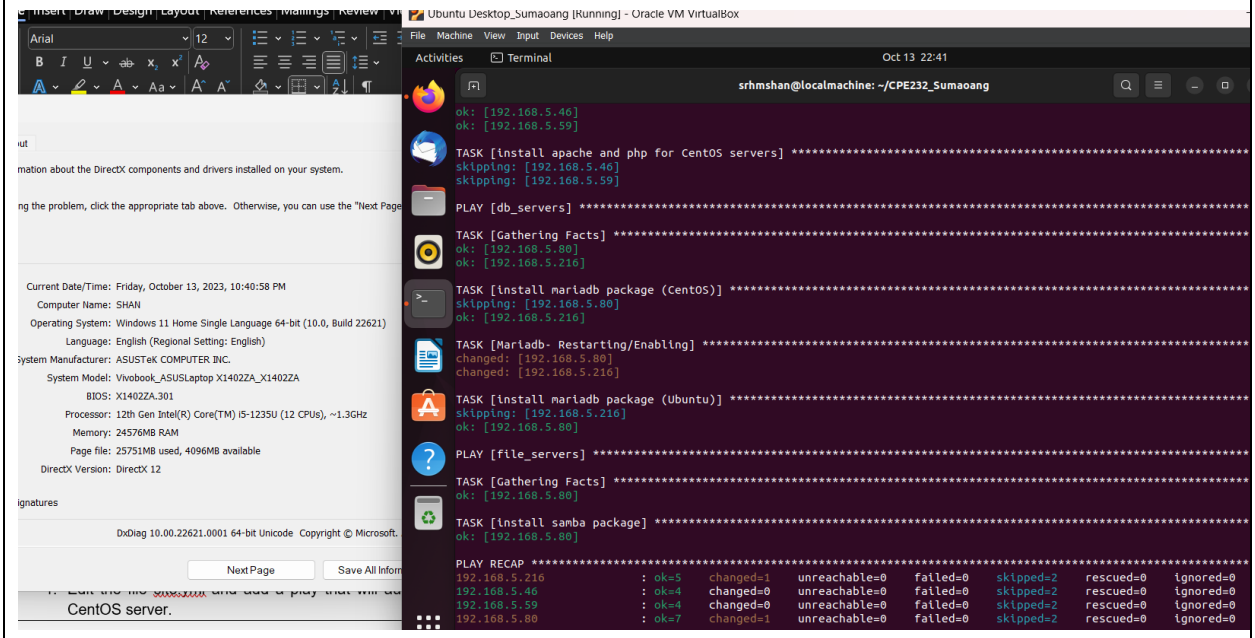



```
- hosts: db_servers
  become: true
  tasks:

  - name: install mariadb package (CentOS)
    tags: centos, db,mariadb
    dnf:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "CentOS"

  - name: "Mariadb- Restarting/Enabling"
    service:
      name: mariadb
      state: restarted
      enabled: true

  - name: install mariadb packege (Ubuntu)
    tags: db, mariadb,ubuntu
    apt:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "Ubuntu"

- hosts: file_servers
  become: true
  tasks:

  - name: install samba package
    tags: samba
    package:
      name: samba
      state: latest
```



Make sure to save the file and exit.

Run the *site.yml* file and describe the result.

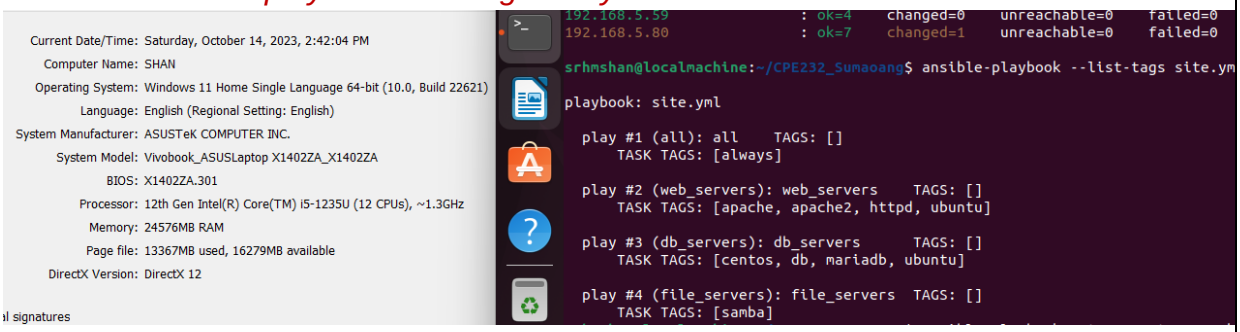


The tags target specific server groups for particular tasks. Adding tags to the

code allows it to install the required packages on the corresponding servers (web_servers, db_servers, or file_servers) while avoiding unnecessary tasks for the other servers. Also, I noticed that the installation was way faster compared to running the playbook with no tags.

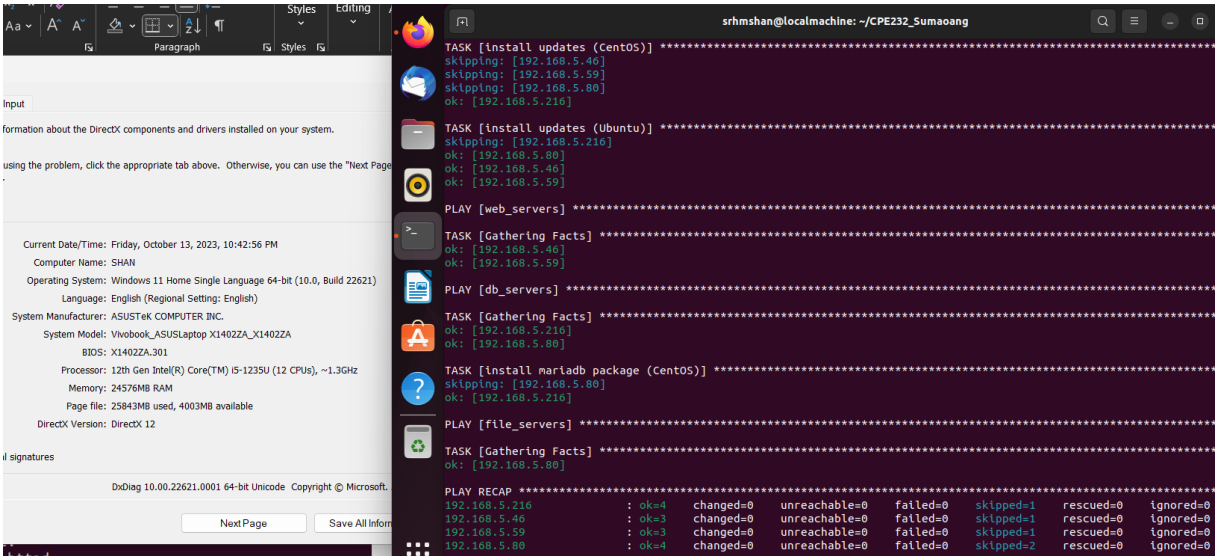
2. On the local machine, try to issue the following commands and describe each result:

   2.1 *ansible-playbook --list-tags site.yml*



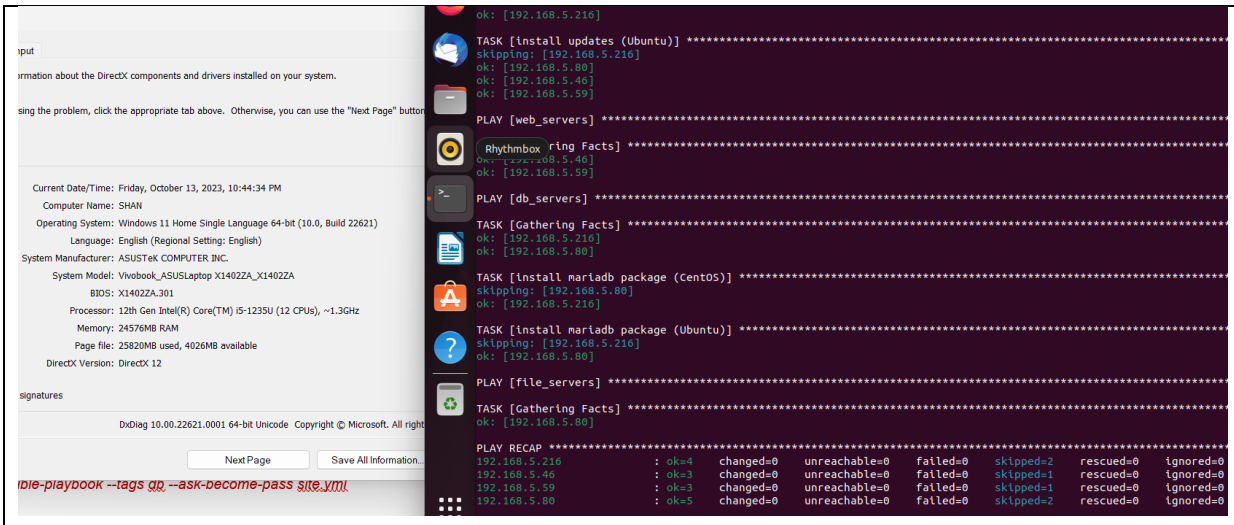
The command executed shows the list of specific tags that I put in each play, in the playbook "site.yml". Since there was no tag in the first play, it displays "always" which means that for every run, it will execute play #1. (I retook the screenshot because I forgot to include dxdiag)

   2.2 *ansible-playbook --tags centos --ask-become-pass site.yml*



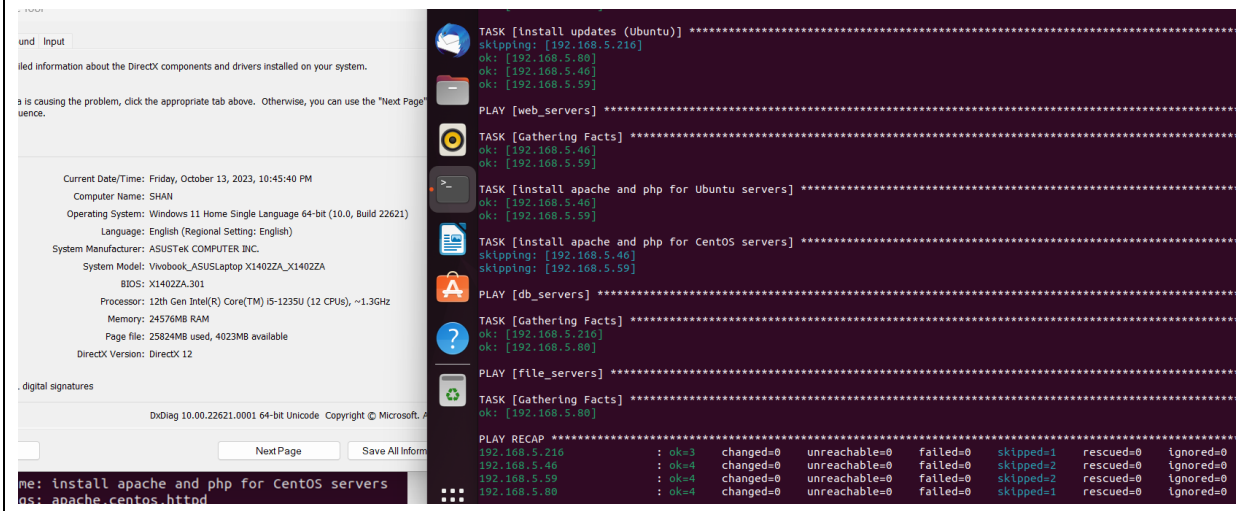
This command runs the plays with tags "centos" using the command "-- tags".

   2.3 *ansible-playbook --tags db --ask-become-pass site.yml*



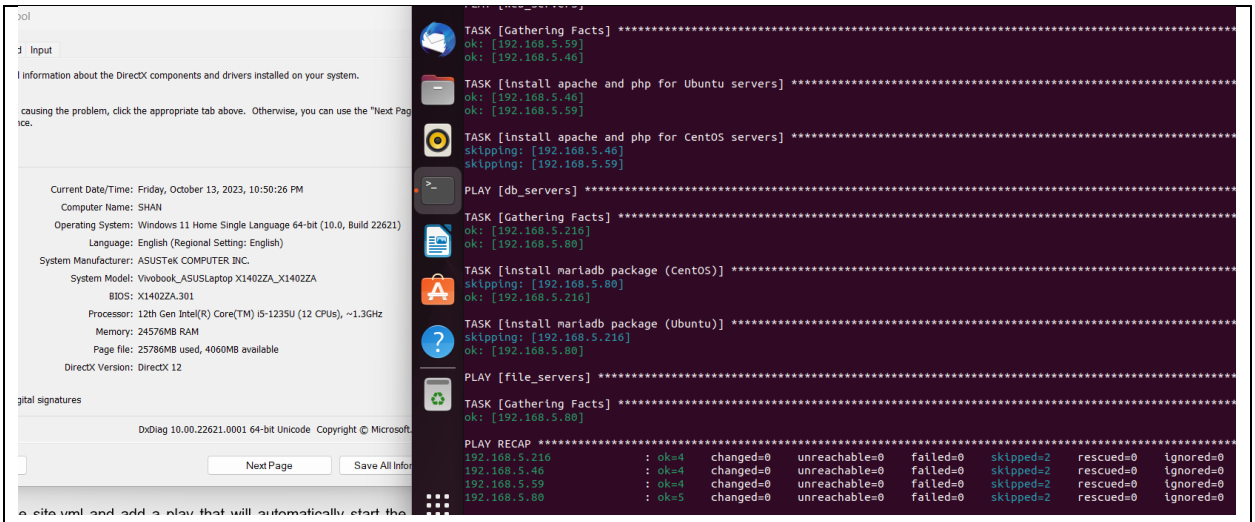
This command runs the playbook and targets the plays with tags “db” using the command “-- tags".

2.4 *ansible-playbook --tags apache --ask-become-pass site.yml*



This command runs and targets the plays tagged “apache” using the command “-- tags".

2.5 *ansible-playbook --tags “apache,db” --ask-become-pass site.yml*



It is possible to execute a command that targets multiple tags at once using "--tags". This command looks for plays that are tagged "apache" and "db".

Task 3: Managing Services

1. Edit the file site.yml and add a play that will automatically start the httpd on CentOS server.

```
- name: install apache and php for CentOS servers
  tags: apache,centos,httpd
  dnf:
    name:
      - httpd
      - php
    state: latest
  when: ansible_distribution == "CentOS"

- name: start httpd (CentOS)
  tags: apache, centos,httpd
  service:
    name: httpd
    state: started
  when: ansible_distribution == "CentOS"
```

Figure 3.1.1

Make sure to save the file and exit.

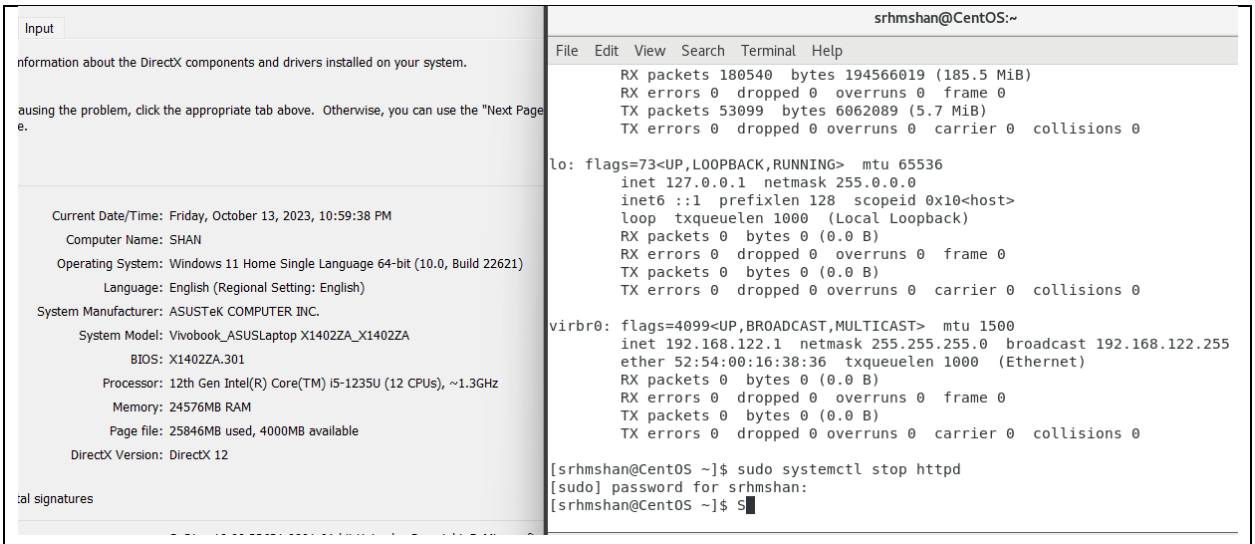
You would also notice from our previous activity that we already created a module that runs a service.

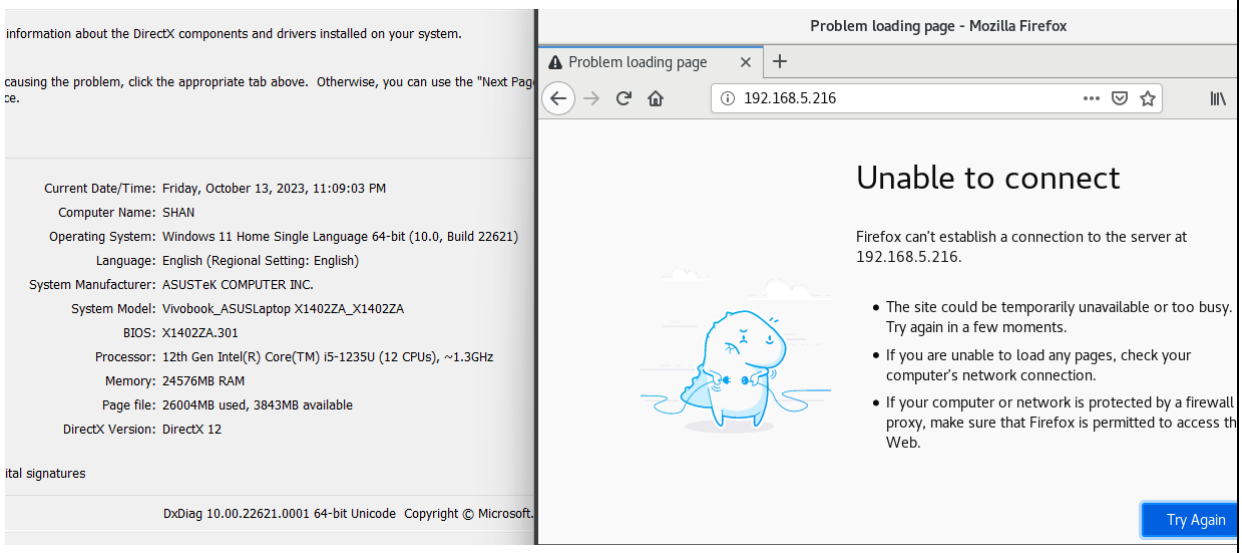
```
- hosts: db_servers
  become: true
  tasks:

  - name: install mariadb package (CentOS)
    tags: centos, db,mariadb
    dnf:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "CentOS"

  - name: "Mariadb- Restarting/Enabling"
    service:
      name: mariadb
      state: restarted
      enabled: true
```

Figure 3.1.2

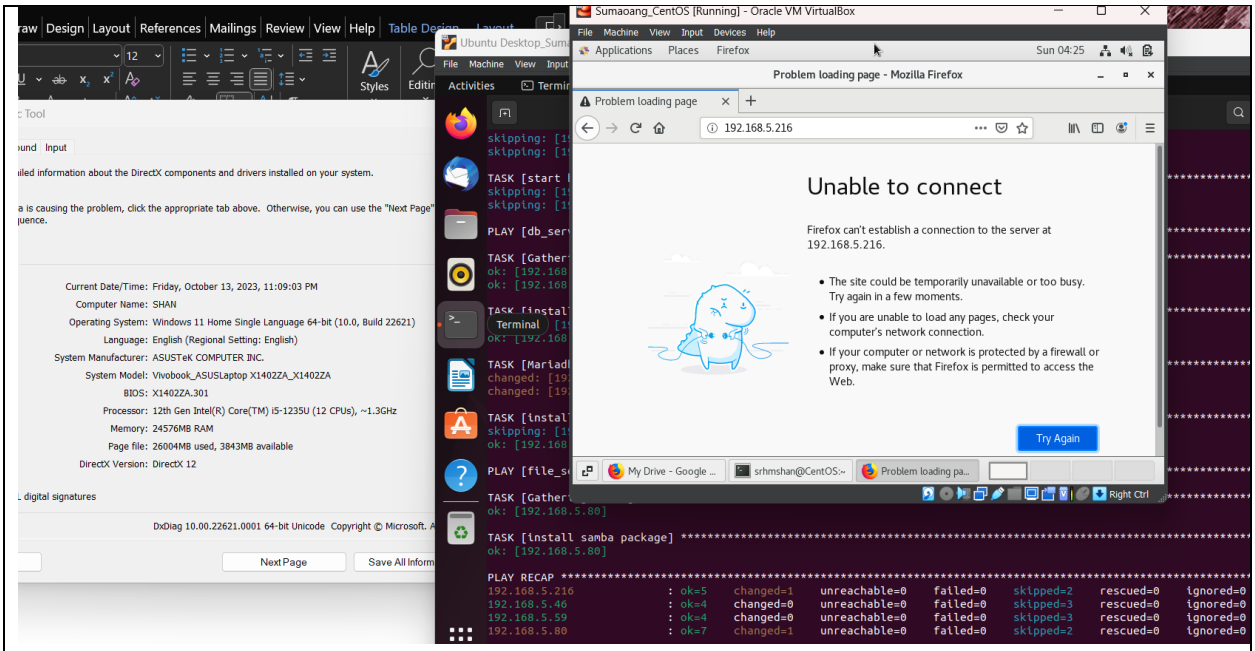
This is because in CentOS, installed packages' services are not run automatically. Thus, we need to create the module to run it automatically.

2. To test it, before you run the saved playbook, go to the CentOS server and stop the currently running httpd using the command *sudo systemctl stop httpd.* When prompted, enter the sudo password. After that, open the browser and enter the CentOS server's IP address. You should not be getting a display because we stopped the httpd service already.



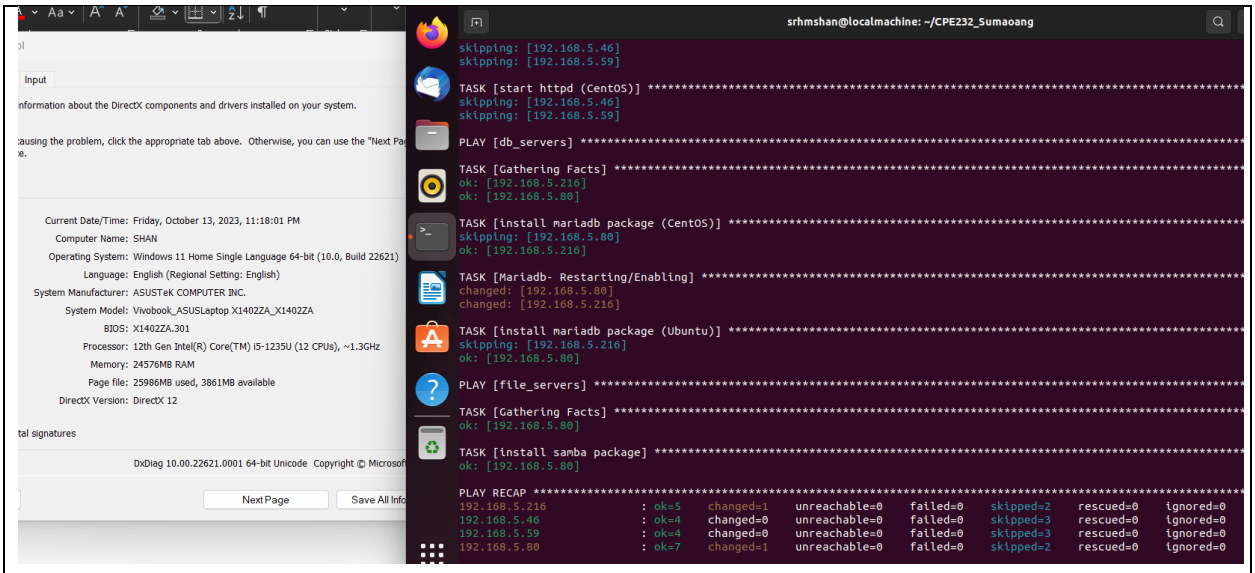


3. Go to the local machine and this time, run the *site.yml* file. Then after running the file, go again to the CentOS server and enter its IP address on the browser. Describe the result.



After running the command, the result obtained was that I could access the web content hosted on the CentOS server, even without it showing that I am connected and even without manually enabling httpd again on the CentOS terminal. This was because the play I added had automatically started the httpd service, which is responsible for serving web content, making it accessible again. As a result, the target plays were "changed" then executed successfully.

To automatically enable the service every time we run the playbook, use the command *enabled: true* similar to Figure 7.1.2 and save the playbook.



Reflections:

Answer the following:

1. What is the importance of putting our remote servers into groups?
   The importance of putting our remote servers into groups is to consider it as a practical way to make managing them easier. It simplifies tasks by allowing you to apply changes to whole groups of servers at once, rather than dealing with each server individually

2. What is the importance of tags in playbooks?
   I would think of tags in playbooks as shortcuts. It lets you choose specific tasks you want to run for easier access to specific plays. They also make troubleshooting faster by focusing only on the parts that need attention. Plus, they help keep your playbooks organized and allow tasks to run simultaneously, making it more efficient.

3. Why do think some services need to be managed automatically in playbooks?
   Automating service management in playbooks is like having a personal assistant for routine tasks. It ensures things are done consistently, saving you time and reducing the chance of errors. It's also flexible and can adapt to different situations, while keeping a record of everything for easy tracking and changes when needed.

Conclusion:

In this activity, I learned practical ways to manage remote servers and streamline playbook execution. I began by organizing servers into groups, making tasks simpler and enhancing security by creating specific rules for different server categories. This is for efficient server management.

I explored the use of tags in playbooks, which are like shortcuts for task selection. Tags enabled me to run specific tasks, making playbook customization and troubleshooting faster, and kept everything organized and efficient. Lastly, I experimented with automating service management through playbooks. Theses automatizations saves time, reduces errors, and maintains adaptability in different plays.

In summary, this activity emphasized the importance of well-structured server grouping, playbook tagging, and automated service management for more efficient server management and configuration.