User Management, MySQL configuration and Setting Python HTTP server with Annsible

Software Construction and Evolution

TABLE OF CONTENTS

1. INTRODUCTION

Configuration management refers to the practice of systematically managing and maintaining changes to a system to ensure its integrity and performance over time. It often involves the use of specialized tools and processes that enable automation, monitoring, and consistency in system environments. Although this concept has roots outside the IT industry, it is widely associated with server configuration management today.

In server environments, configuration management is often synonymous with **IT automation** or **server orchestration**. These terms underscore the essential tasks of managing multiple systems from a single control point and ensuring that configurations are applied consistently across the infrastructure.

This document will explore the advantages of leveraging configuration management tools **to streamline the setup** and **maintenance** of your server infrastructure. Specifically, we will focus on **Ansible**, a widely adopted tool known for **simplifying automation and enhancing contro**l over complex environments.

1.1 Ansible Overview

Ansible is a modern configuration management and automation tool designed to simplify the process of managing and provisioning remote servers. Its minimalist approach allows users to start automating quickly and efficiently. The tool leverages YAML, a simple, human-readable data format, for writing playbooks, making it easier for users to create complex automation tasks without needing extensive programming knowledge.

A key feature of Ansible is its agentless architecture. Unlike many other configuration management tools, Ansible doesn’t require installing any software on the nodes it manages. Instead, it communicates directly with them via standard SSH, using a control machine that runs the Ansible software.

Ansible’s idempotent behavior ensures that tasks are not unnecessarily repeated. For example, if a package is already installed, Ansible will skip the installation in subsequent runs, maintaining the desired system state without redundant actions. This feature ensures that running a playbook multiple times leads to consistent outcomes, helping to avoid potential configuration drift.

To make automation flexible and dynamic, Ansible allows the use of variables, conditionals, and loops. This enables users to tailor their playbooks for different environments or scenarios, improving efficiency. Additionally, Ansible gathers detailed information about managed nodes, such as network configurations and OS details, which are stored as system facts. These facts can be referenced in playbooks to further customize tasks based on the target system's attributes.

Ansible also integrates with the Jinja2 templating engine, allowing for dynamic content generation and variable handling within playbooks. Templates make it easy to configure services or deploy applications consistently across multiple systems. For instance, you can create a reusable template to configure virtual hosts for Apache across different servers.

The tool comes with a large library of built-in modules, which streamline common administrative tasks like package installation (using `apt`, `yum`, etc.), file synchronization (`rsync`), and database management (MySQL, PostgreSQL, MongoDB). Moreover, Ansible's functionality can be extended through custom plugins and modules, providing more flexibility.

2. SETTING UP ANSIBLE

2.1 Requirements

To execute the automated setup described in this guide, you'll need the following:

- Ansible Control Node: This should be an Ubuntu 20.04 machine with

Ansible installed and configured to connect to remote servers via SSH keys. Ensure the control node has a user with sudo privileges and that a firewall is enabled. For detailed instructions, refer to the "How to Install and Configure Ansible on Ubuntu 20.04" guide.

- Remote Server (Ansible Host): A fresh Ubuntu 20.04 installation is required on the remote server, with no prior setup necessary. However, SSH access from the Ansible control node must be configured, as this server will be targeted for automated provisioning.

This setup enables smooth communication between the control node and the target server for efficient configuration management.

2.2 Installation Guide

For users seeking to install Ansible, the following platform-specific guides are provided to assist with the setup process. These methods are current and applicable with today’s technology:

- Windows: Ansible can be installed via the Windows Subsystem for Linux (WSL), Cygwin, or a virtual machine. Detailed instructions are available at:
  https://phoenixnap.com/kb/install-ansible-on-windows.
- MacOS: Installation on MacOS can be performed through the Homebrew package manager. The step-by-step guide can be found here:
  https://www.ansiblepilot.com/articles/how-to-install-ansible-in-macos-ansible-install/.
- Ubuntu 20.04: Ubuntu users can install Ansible via the APT package manager. For the full instructions, visit:
  https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-20-04.

These resources provide comprehensive and reliable methods for installing Ansible on various platforms.

2.2.1 VM on Window 11

Setting up a Virtual Machine (VM) on Windows 11 is an effective approach to creating a Linux-like environment where we can install and run Ansible, especially when direct installation is challenging on Windows. This section outlines the steps required to configure a VM for this purpose.

1. Enabling Virtualization

To begin, ensure that virtualization is enabled on your system. Virtualization technology must be turned on in your system’s BIOS or UEFI settings. You can check if it's enabled by going to the "Task Manager" under the "Performance" tab, where it will indicate the status of virtualization.

2. Installing a Hypervisor (e.g., Hyper-V or VirtualBox)

Windows 11 includes Hyper-V, a built-in virtualization tool that can be used to create and manage VMs. To enable Hyper-V:

- Go to "Turn Windows features on or off" in the Control Panel.
- Enable the Hyper-V checkbox and restart your system.

Alternatively, you can use VirtualBox, a free and open-source hypervisor. Download and install VirtualBox from the official site, as it is a commonly used option for managing VMs in both personal and professional settings.

3. Setting Up a Linux-Based VM

Since Ansible runs natively on Linux systems, you can install a Linux distribution (such as Ubuntu) within the VM. For Hyper-V:

- Open Hyper-V Manager and create a new virtual machine by selecting the desired Linux distribution (like Ubuntu).
- Allocate system resources (CPU, memory, and storage) to the VM according to your hardware capacity.

If using VirtualBox:

- Start VirtualBox and click "New" to create a VM, specifying Ubuntu or another Linux-based OS.
- Adjust system settings, such as assigning memory and storage, and then install the Linux OS within the VM.

3. ANSIBLE BEHIND THE SCENE

3.1 Control Node: A control node is any system where Ansible is installed and used to manage other servers. It can be a personal computer or any device running a Linux or Unix-based OS. Although Ansible cannot be directly installed on Windows, you can bypass this by using a Linux virtual machine within Windows to run Ansible.

3.2 Managed Nodes: Managed nodes are the systems that Ansible controls, and they must be accessible via SSH with Python (version 2.6+ or 3.5+) installed. Ansible supports various operating systems, including Windows servers as managed nodes.

3.3 Inventory: The inventory is a file listing the hosts managed by Ansible. While a default inventory is created during installation, it's best to use custom inventories for specific projects. Inventories can be static (.ini files) or dynamically generated in any language that outputs JSON.

3.4 Tasks: A task is a single action Ansible performs on a managed node. Tasks can be run via ad-hoc commands or included in playbooks, where multiple tasks are executed in sequence as part of an automation script.

3.5 Playbook: A playbook is a collection of tasks that define automation processes. Playbooks specify the target hosts, whether privilege escalation is required, and any variables or file inclusions. Written in YAML, playbooks allow for step-by-step execution of tasks on managed nodes.

3.6 Handlers: Handlers are actions that respond to triggers within tasks, such as restarting a service. They execute after all tasks in a play are complete, ensuring efficiency. However, handlers can be forced to execute immediately if necessary.

3.7 Roles: Roles are predefined sets of playbooks and files organized for specific purposes, like setting up a web server or a MySQL database. They allow for reusable, modular automation across different projects, making playbook management more efficient.

4. ANSIBLE COMMANDS AND TESTING

4.1 Getting started

In this section, we will start by running a simple playbook for the purpose of introducing basic Ansible commands and routine.This particular playbook is designed to perform essential setup tasks on your Ansible hosts. The tasks include installing Aptitude, creating a new sudo user, configuring SSH, installing system packages, and setting up a firewall.

1. Install Aptitude
   To install Aptitude, the playbook includes a task using the `apt` module with the command `aptitude`.
2. Create a Sudo User
   The playbook creates a new user with sudo privileges using the `user` module and sets up password-less sudo by modifying the `/etc/sudoers` file with the `lineinfile` module.
3. Configure SSH
   The playbook adds a local SSH public key to the new user's `authorized_keys` file using the `authorized_key` module. It also disables root password authentication by updating the `/etc/ssh/sshd_config` file and restarting the SSH service using the `service` module.
4. Install System Packages
   Essential system packages are installed by listing them in the `apt` module under the `name` parameter, ensuring they are present on the host.
5. Configure Firewall
   The playbook configures the UFW firewall by allowing SSH connections and denying all other traffic. This is achieved with the `ufw` module.

Steps to Prepare and Run Your Ansible Playbook

Prepare Your Ansible Control Node

Ensure you are logged in as a user with sudo privileges on your Ansible control node.

Create Your Playbook File

Define the tasks in a YAML file, such as `setup.yml`, which will be executed on the managed hosts.

Add Aptitude Installation Task

Include a task to install Aptitude:

```
- name: Install Aptitude

  apt:

    name: aptitude

    state: present
```

Add Sudo User Setup

Add tasks to create a new user with sudo rights and configure password-less sudo:

```
- name: Create a new sudo user

  user:

    name: newuser

    groups: sudo

    append: yes

    state: present

- name: Set up password-less sudo
```

```
  lineinfile:

    path: /etc/sudoers

    line: 'newuser ALL=(ALL) NOPASSWD:ALL'

    validate: '/usr/sbin/visudo -cf %s'
```

Configure SSH Key and Disable Root Password

Include tasks to copy the SSH key and disable root password authentication:

```
- name: Copy SSH key for new user

  authorized_key:

    user: newuser

    state: present

    key: "{{ lookup('file',
'/path/to/your/public/key') }}"

- name: Disable root password authentication

  lineinfile:

    path: /etc/ssh/sshd_config

    regexp: '^PermitRootLogin'

    line: 'PermitRootLogin no'
  notify:

    - restart ssh



- name: Restart SSH service

  service:

    name: ssh

    state: restarted
```

Install System Packages

Add a task to install necessary packages:

```
- name: Install system packages

  apt:

    name:

      - vim

      - curl

      - git

    state: present
```

Configure UFW Firewall

Add tasks to configure the UFW firewall:

```
- name: Allow SSH through UFW

  ufw:

    rule: allow

    name: OpenSSH

- name: Deny all other traffic

  ufw:

    rule: deny

    name: 'Any'
```

Review the Complete Playbook

Verify that all tasks are correctly included in the `setup.yml` file and are properly configured.

Run Your Playbook

Execute the playbook using the command:

```
ansible-playbook -i inventory setup.yml
```

4.2 Real-world Testing

User-management with Ansible

In this section, we will test user and group management on managed nodes using Ansible. The testing environment consists of:

1. Control Node: Running on Ubuntu (WSL on Windows 11).

2. Remote Servers: Ubuntu servers running on Oracle VirtualBox, managed via Ansible from the control node.

3. Inventory File: Contains the information of the remote servers.

Steps for Testing

We will use three playbooks to perform different tasks related to user and group management on the remote servers:

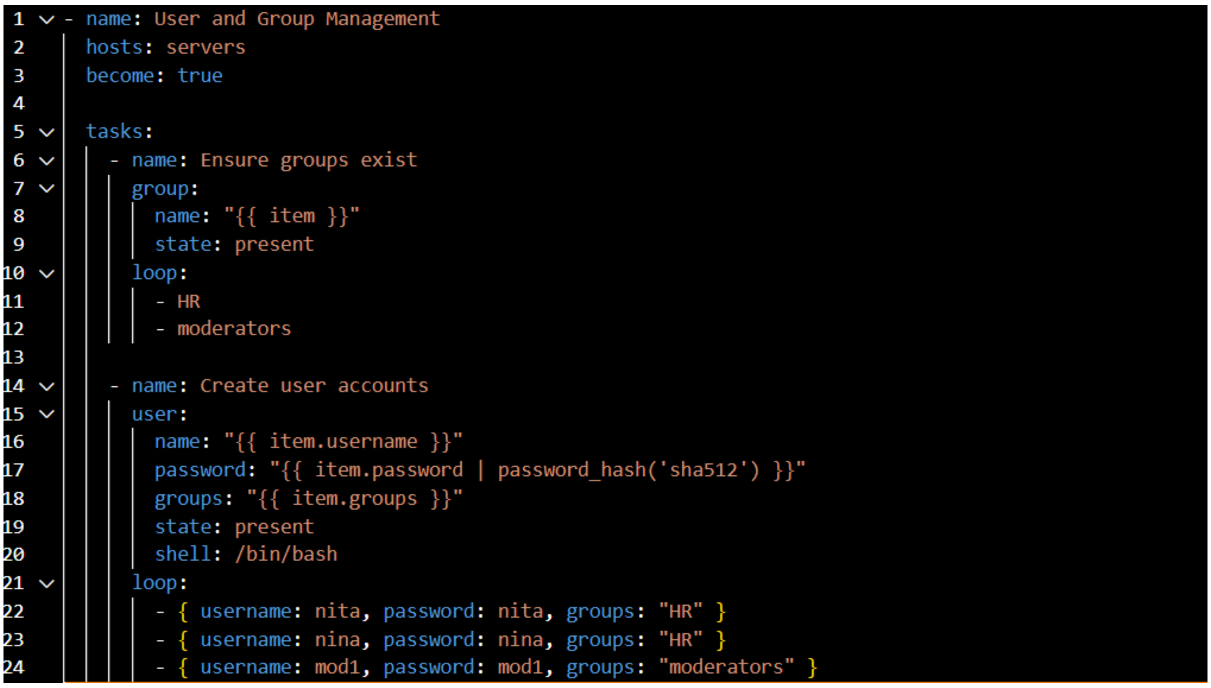
1. Creating Users and Groups: The first playbook (`create_loop.yml`) ensures the creation of user groups and corresponding users with specific privileges.

2. Removing Users and Groups: The second playbook (`remove_loop.yml`) removes users and groups created during testing.

3. Managing a User with Sudo Privileges: The third playbook (`sudo_pri.yml`) creates a user with administrative privileges and configures the sudoers file.

1. Creating Users and Groups

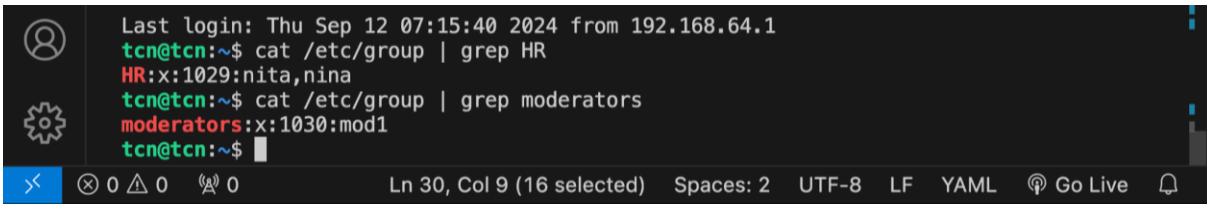
The following playbook (`create_loop.yml`) creates two groups—HR and moderators—and assigns users to each group:



```yaml
- name: User and Group Management
  hosts: servers
  become: true

  tasks:
    - name: Ensure groups exist
      group:
        name: "{{ item }}"
        state: present
      loop:
        - HR
        - moderators

    - name: Create user accounts
      user:
        name: "{{ item.username }}"
        password: "{{ item.password | password_hash('sha512') }}"
        groups: "{{ item.groups }}"
        state: present
        shell: /bin/bash
      loop:
        - { username: nita, password: nita, groups: "HR" }
        - { username: nina, password: nina, groups: "HR" }
        - { username: mod1, password: mod1, groups: "moderators" }
```

Figure 4.2.1

- This playbook will ensure that the groups **HR** and **moderators** exist on the servers, and users such as **nita**, **nina**, and **mod1** are created and assigned to their respective groups.

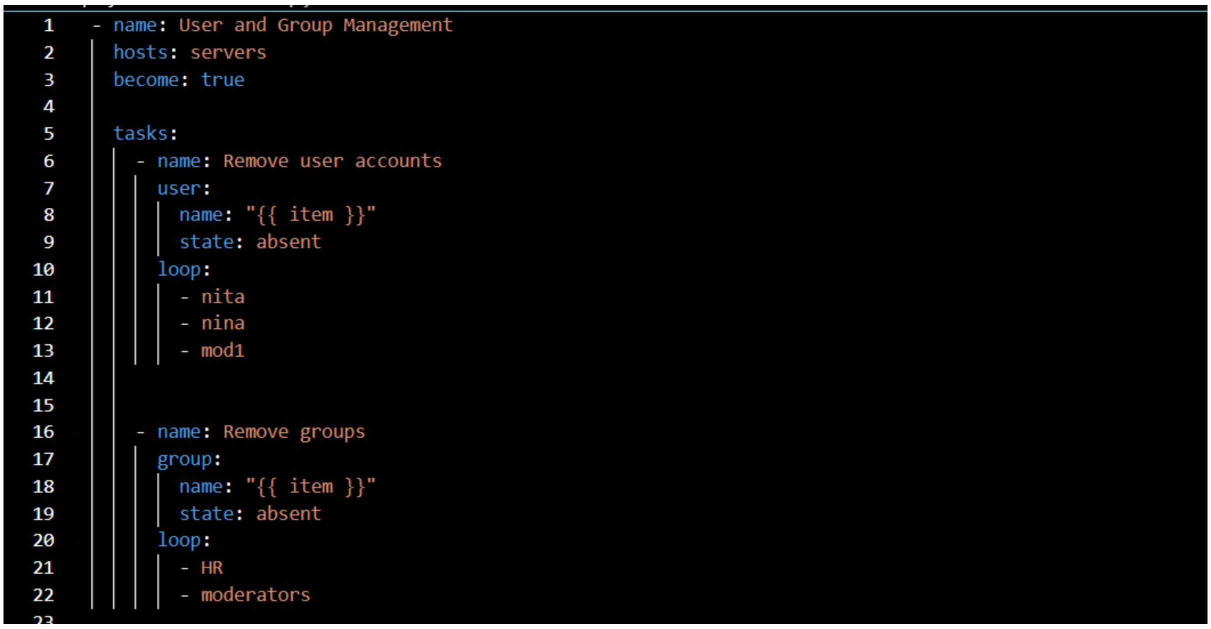


```
Last login: Thu Sep 12 07:15:40 2024 from 192.168.64.1
tcn@tcn:~$ cat /etc/group | grep HR
HR:x:1029:nita,nina
tcn@tcn:~$ cat /etc/group | grep moderators
moderators:x:1030:mod1
tcn@tcn:~$ 
```

Figure 4.2.2 - Result of created group and user

2. Removing Users and Groups

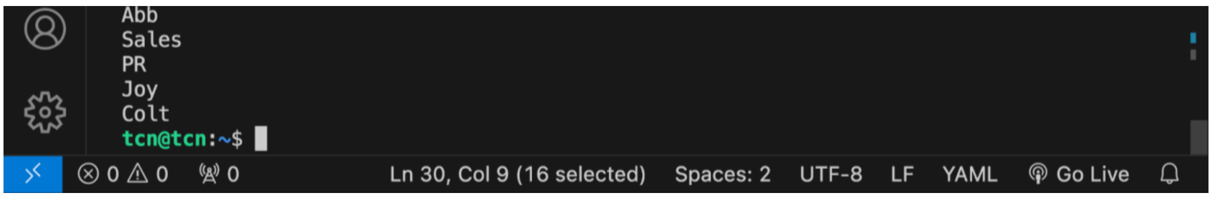
To clean up after testing, we use the ‘remove_loop.yml’ playbook to remove the created users and groups:



```yaml
- name: User and Group Management
  hosts: servers
  become: true

  tasks:
    - name: Remove user accounts
      user:
        name: "{{ item }}"
        state: absent
      loop:
        - nita
        - nina
        - mod1


    - name: Remove groups
      group:
        name: "{{ item }}"
        state: absent
      loop:
        - HR
        - moderators
```

Figure 4.2.3

- This playbook will remove users **nita**,**nina**, and **mod1** and delete the **HR** and **moderators** groups.

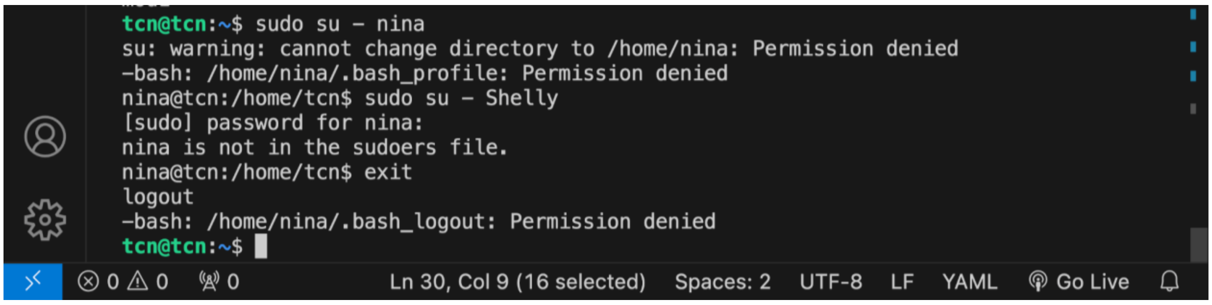


```
Abb
Sales
PR
Joy
Colt
tcn@tcn:~$
```

Figure 4.2.4: Result of removing users and groups

3. Managing a User with Sudo Privileges

As shown in Figure 4.2.5, without granting necessary permissions, attempting to create a user or perform actions on their behalf will result in permission denied errors. This is because the system is unable to verify the user's authority or locate the required resources.

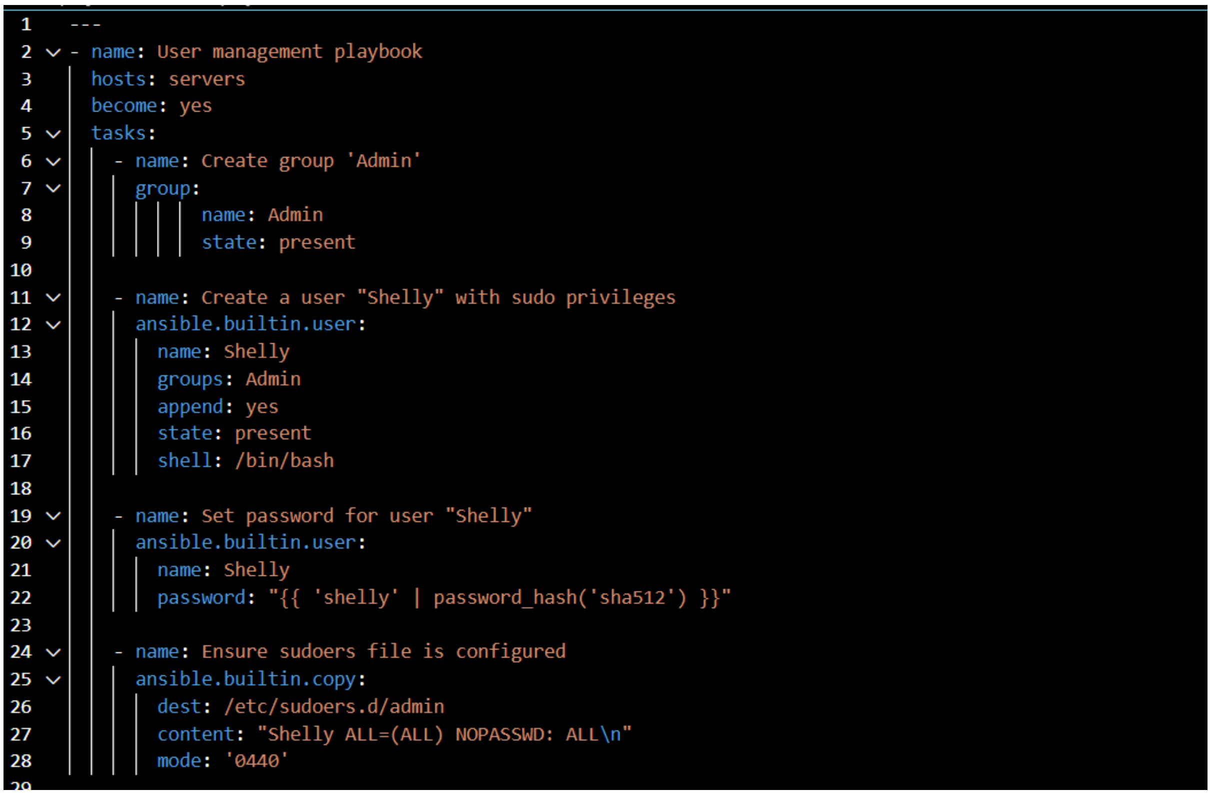


```
tcn@tcn:~$ sudo su - nina
su: warning: cannot change directory to /home/nina: Permission denied
-bash: /home/nina/.bash_profile: Permission denied
nina@tcn:/home/tcn$ sudo su - Shelly
[sudo] password for nina:
nina is not in the sudoers file.
nina@tcn:/home/tcn$ exit
logout
-bash: /home/nina/.bash_logout: Permission denied
tcn@tcn:~$
```

Figure 4.2.5

To prevent such issues, it is imperative to ensure that appropriate permissions are assigned to users or groups based on their roles and responsibilities within the system.

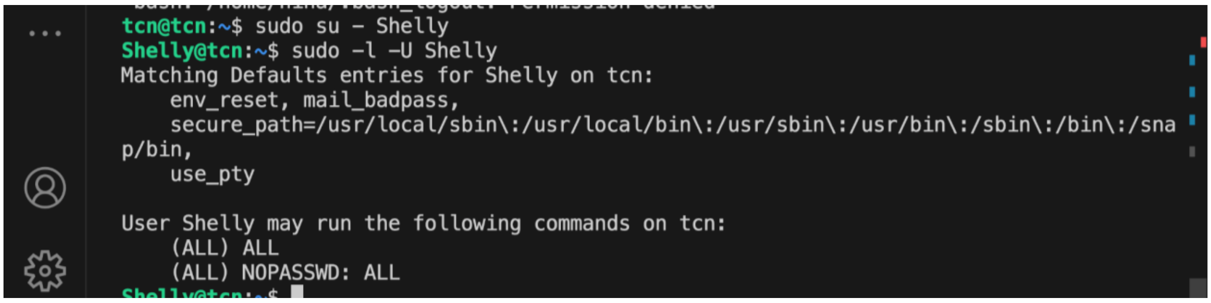
The `sudo_pri.yml` playbook will create a user named **Shelly**, add her to the **Admin** group, and configure sudo privileges:



```yaml
---
- name: User management playbook
  hosts: servers
  become: yes
  tasks:
    - name: Create group 'Admin'
      group:
          name: Admin
          state: present

    - name: Create a user "Shelly" with sudo privileges
      ansible.builtin.user:
        name: Shelly
        groups: Admin
        append: yes
        state: present
        shell: /bin/bash

    - name: Set password for user "Shelly"
      ansible.builtin.user:
        name: Shelly
        password: "{{ 'shelly' | password_hash('sha512') }}"

    - name: Ensure sudoers file is configured
      ansible.builtin.copy:
        dest: /etc/sudoers.d/admin
        content: "Shelly ALL=(ALL) NOPASSWD: ALL\n"
        mode: '0440'
```

Figure 4.2.6

This playbook ensures that **Shelly** is created, added to the **Admin** group, and granted sudo privileges without requiring a password. The result is shown in Figure 4.2.7.



```
tcn@tcn:~$ sudo su - Shelly
Shelly@tcn:~$ sudo -l -U Shelly
Matching Defaults entries for Shelly on tcn:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/sna
p/bin,
    use_pty

User Shelly may run the following commands on tcn:
    (ALL) ALL
    (ALL) NOPASSWD: ALL
Shelly@tcn:~$
```

Figure 4.2.7

Execution

To execute the playbooks, run the following commands from control node:

ansible-playbook -i inventory.init create_loop.yml

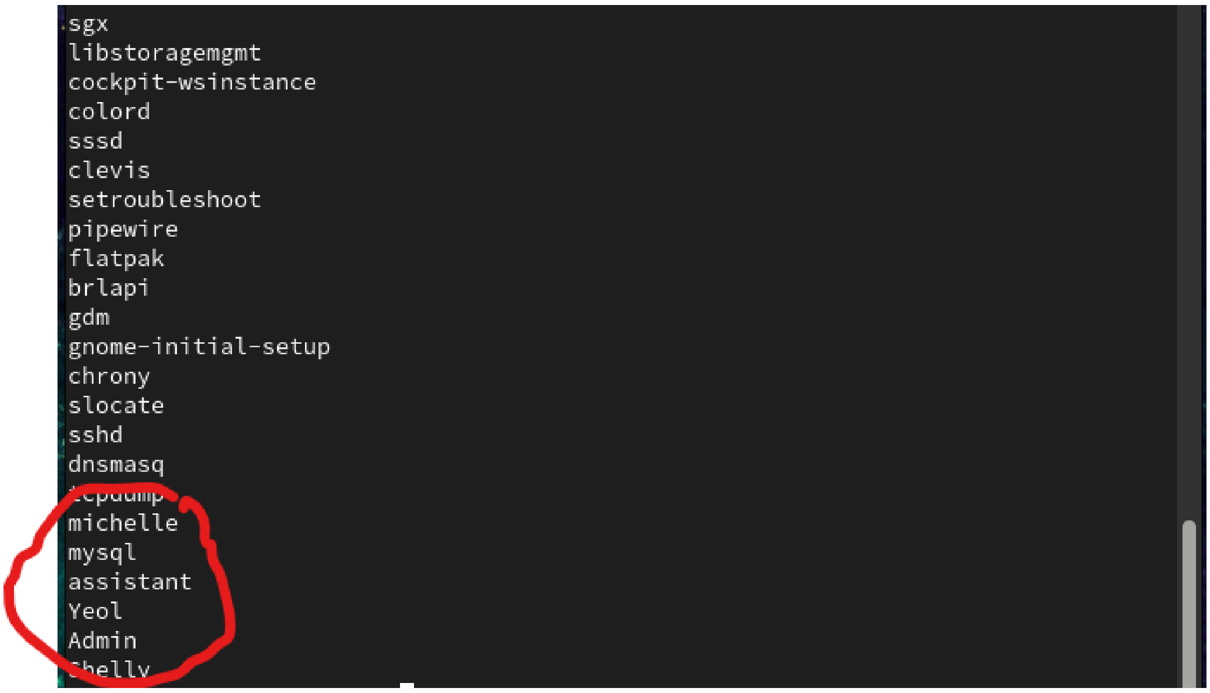
ansible-playbook -i inventory.init sudo_pri.yml

After testing, to remove the users and groups, run:

ansible-playbook -i inventory.init remove_loop.yml

Result

Check whether groups were created by using command “compgen -g”



```
sgx
libstoragemgmt
cockpit-wsinstance
colord
sssd
clevis
setroubleshoot
pipewire
flatpak
brlapi
gdm
gnome-initial-setup
chrony
slocate
sshd
dnsmasq
tcpdump
michelle
mysql
assistant
Yeol
Admin
Shelly
```

Figure 4.2.8

MySQL Configuration with Ansible

In this section, we will test the configuration of MySQL databases and user management on remote servers using Ansible.

Steps for Testing

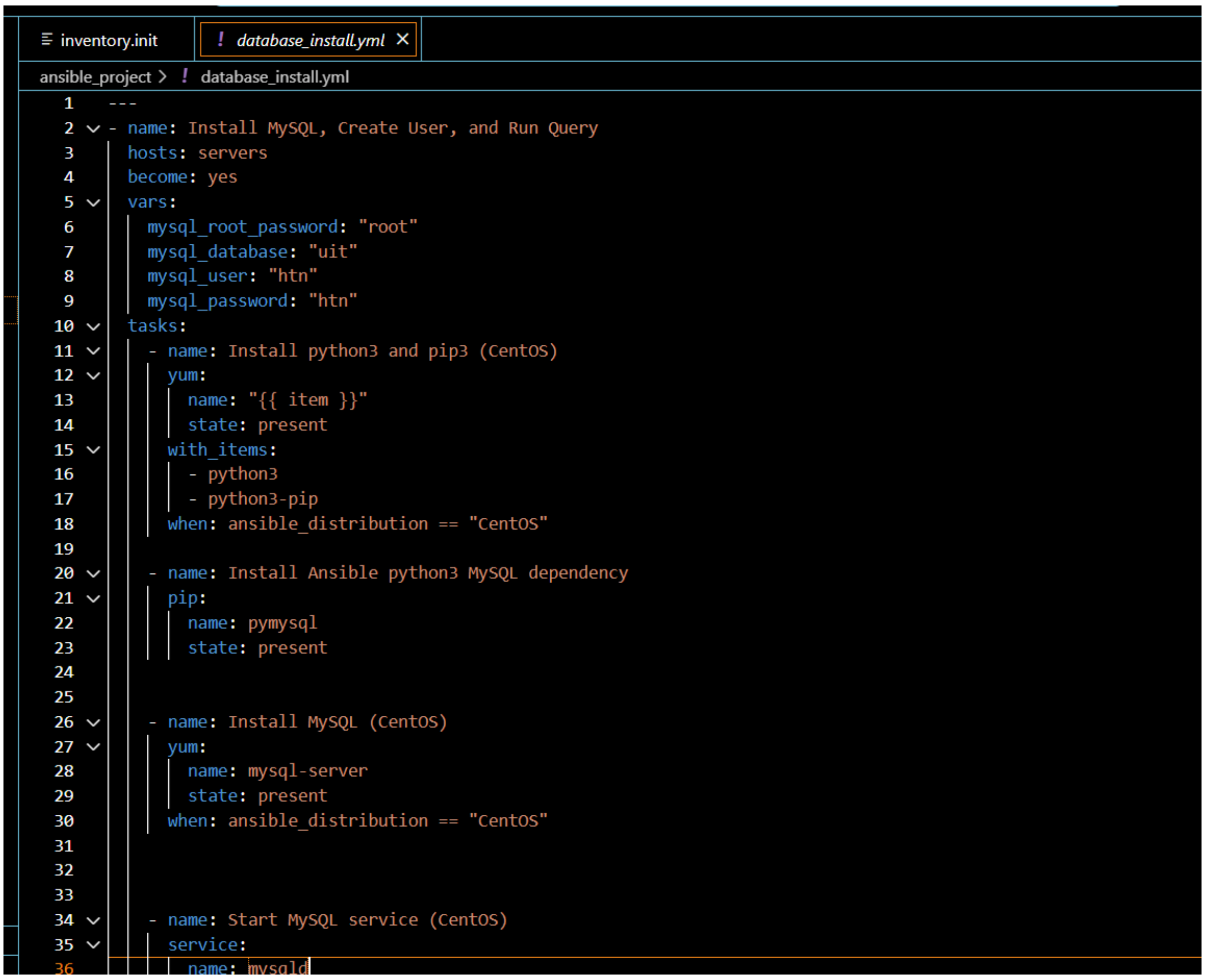
1. Installing MySQL: The first playbook (`database_install.yml`) installs MySQL on the remote servers, ensuring that the necessary Python dependencies for MySQL management are present.

2. Creating MySQL Users and Databases: The second playbook (`database_create_new_user.yml`) creates a new database and a MySQL user with privileges.

3. Running SQL Queries: The third playbook (`database_run_query.yml`) creates a table and inserts sample data into it.

1. Installing MySQL

The following playbook (`database_install.yml`) installs MySQL along with the necessary dependencies:



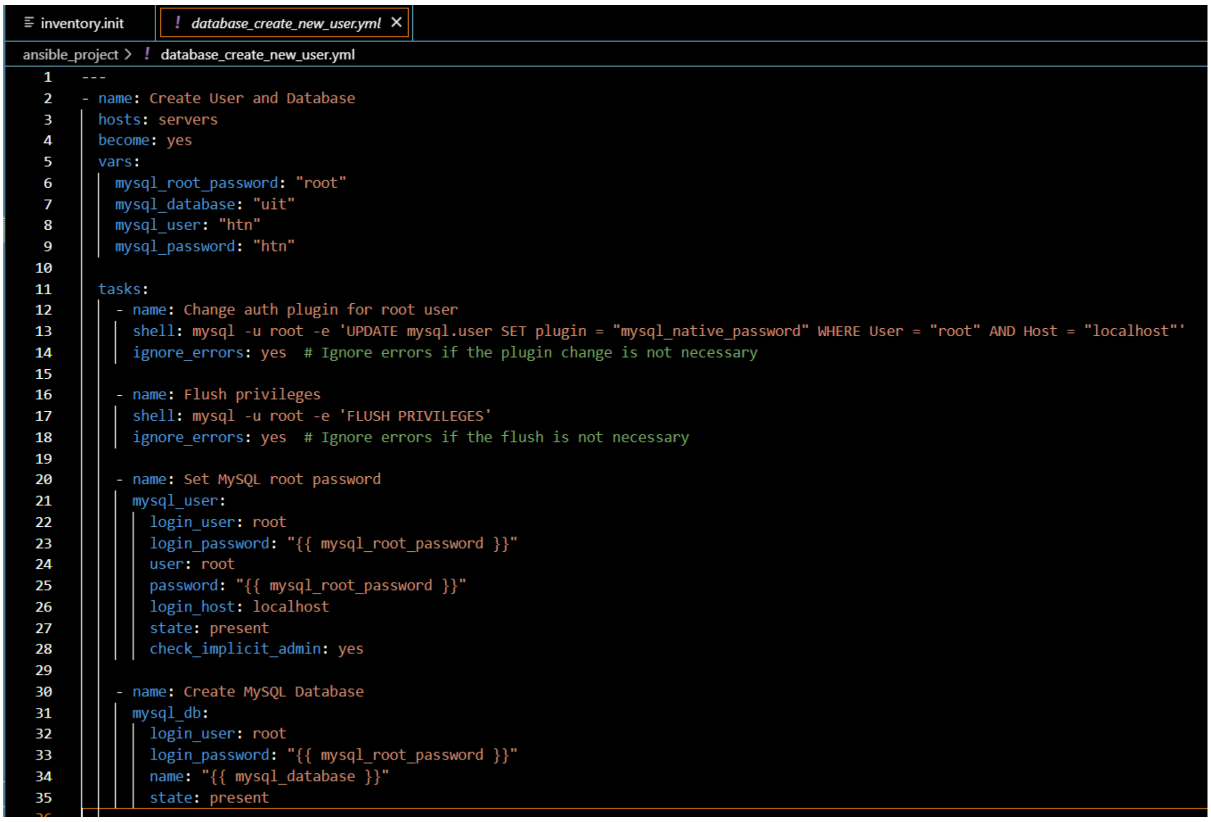
```yaml
---
- name: Install MySQL, Create User, and Run Query
  hosts: servers
  become: yes
  vars:
    mysql_root_password: "root"
    mysql_database: "uit"
    mysql_user: "htn"
    mysql_password: "htn"
  tasks:
    - name: Install python3 and pip3 (CentOS)
      yum:
        name: "{{ item }}"
        state: present
      with_items:
        - python3
        - python3-pip
      when: ansible_distribution == "CentOS"

    - name: Install Ansible python3 MySQL dependency
      pip:
        name: pymysql
        state: present


    - name: Install MySQL (CentOS)
      yum:
        name: mysql-server
        state: present
      when: ansible_distribution == "CentOS"



    - name: Start MySQL service (CentOS)
      service:
        name: mysqld
```

Figure 4.2.9

This playbook ensures that Python and MySQL are installed on the managed nodes, and it starts the MySQL service.

2. Creating MySQL Users and Databases

The `database_create_new_user.yml` playbook creates a MySQL database and user, and assigns the necessary privileges:


inventory.init
! database_create_new_user.yml
ansible_project > ! database_create_new_user.yml

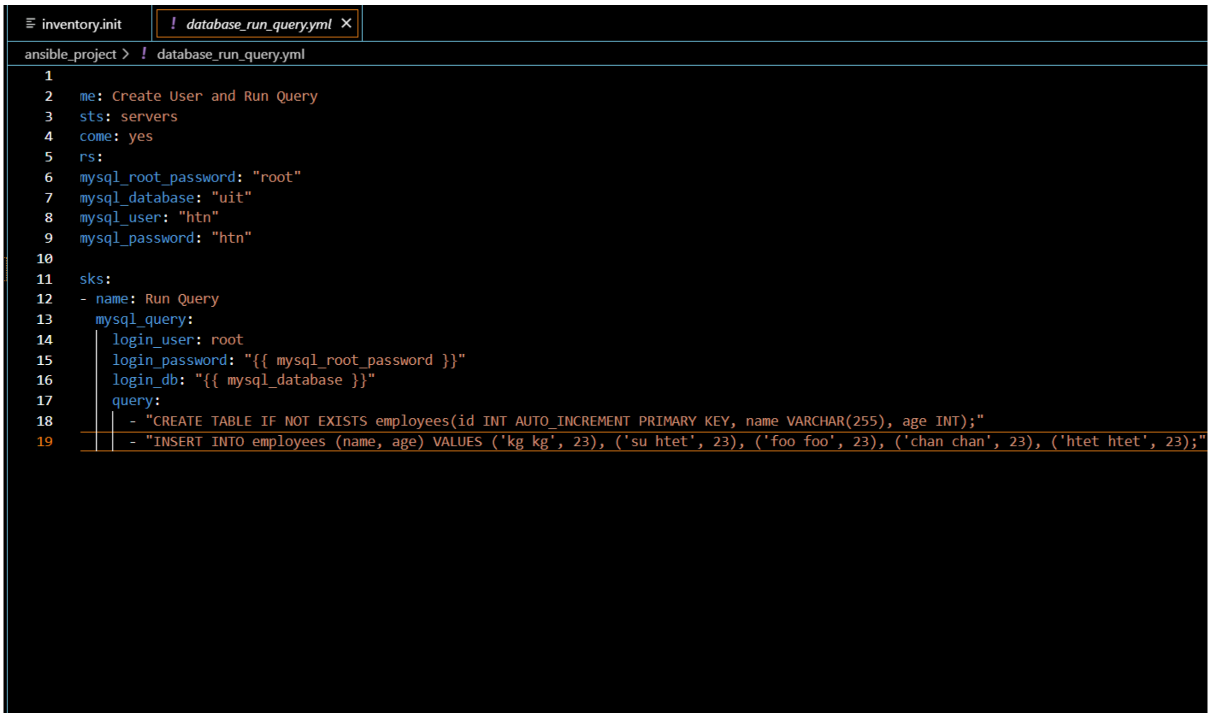
```yaml
---
- name: Create User and Database
  hosts: servers
  become: yes
  vars:
    mysql_root_password: "root"
    mysql_database: "uit"
    mysql_user: "htn"
    mysql_password: "htn"

  tasks:
    - name: Change auth plugin for root user
      shell: mysql -u root -e 'UPDATE mysql.user SET plugin = "mysql_native_password" WHERE User = "root" AND Host = "localhost"'
      ignore_errors: yes  # Ignore errors if the plugin change is not necessary

    - name: Flush privileges
      shell: mysql -u root -e 'FLUSH PRIVILEGES'
      ignore_errors: yes  # Ignore errors if the flush is not necessary

    - name: Set MySQL root password
      mysql_user:
        login_user: root
        login_password: "{{ mysql_root_password }}"
        user: root
        password: "{{ mysql_root_password }}"
        login_host: localhost
        state: present
        check_implicit_admin: yes

    - name: Create MySQL Database
      mysql_db:
        login_user: root
        login_password: "{{ mysql_root_password }}"
        name: "{{ mysql_database }}"
        state: present
```

Figure 4.2.10

This playbook ensures that a new database called `uit` and a user `htn` with all privileges are created.

3. Running SQL Queries

Finally, the ‘database_run_query.yml’ playbook runs a SQL query to create a table and insert data into it:



```
me: Create User and Run Query
sts: servers
come: yes
rs:
mysql_root_password: "root"
mysql_database: "uit"
mysql_user: "htn"
mysql_password: "htn"

sks:
- name: Run Query
  mysql_query:
    login_user: root
    login_password: "{{ mysql_root_password }}"
    login_db: "{{ mysql_database }}"
    query:
      - "CREATE TABLE IF NOT EXISTS employees(id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), age INT);"
      - "INSERT INTO employees (name, age) VALUES ('kg kg', 23), ('su htet', 23), ('foo foo', 23), ('chan chan', 23), ('htet htet', 23);"
```

Figure 4.2.11

This playbook creates an `employees` table in the `uit` database and inserts sample records into it.

Execution: To execute the playbooks, run the following commands from your control node:

ansible-playbook -i inventory.init database_install.yml

ansible-playbook -i inventory.init database_create_new_user.yml

ansible-playbook -i inventory.init database_run_query.yml

Python HTTP Server Setup

In this section, we will test setting up a basic Python HTTP server on a managed node using Ansible.

Steps for Testing

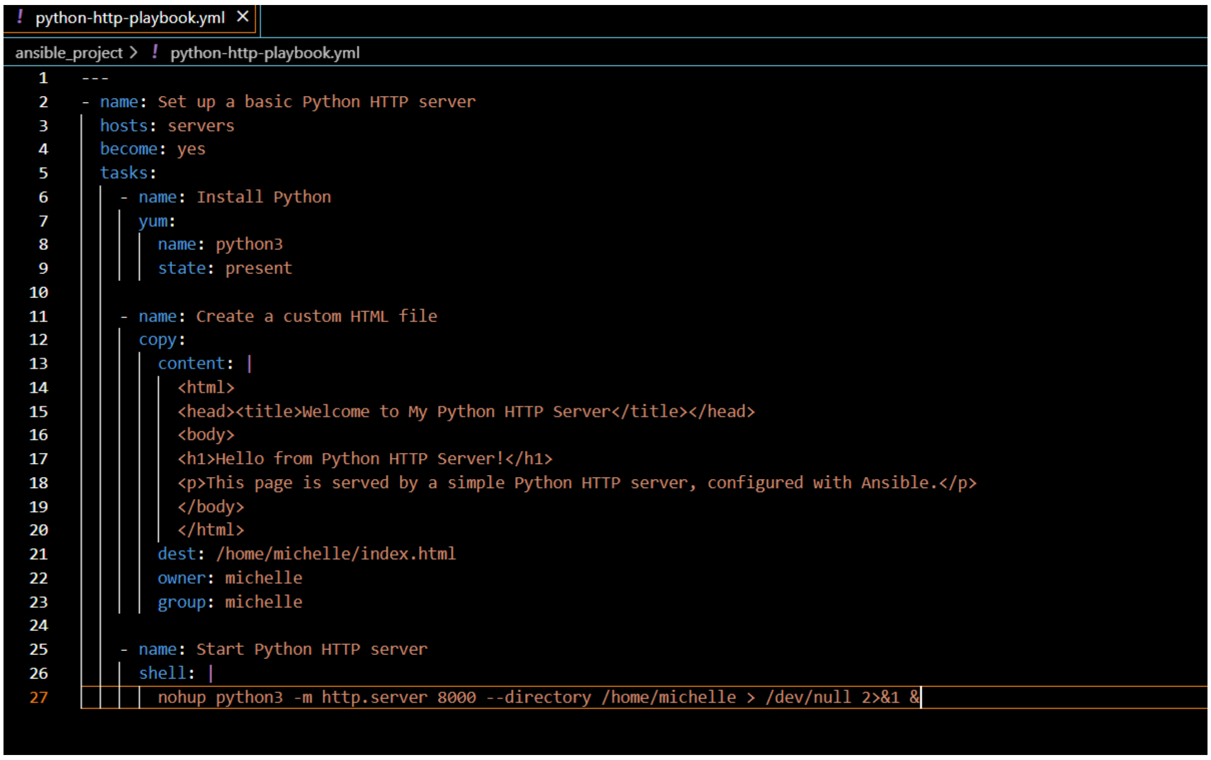
1. Installing Python: The playbook will first ensure that Python3 is installed on the managed node.

2. Creating HTML File: The playbook will create a custom HTML file in the home directory of the user “michelle.”

3. Starting the Python HTTP Server: Finally, the playbook will run the Python HTTP server to serve the created HTML page.

Python HTTP Server Configuration

This is the content of the playbook `python_http_server.yml`:

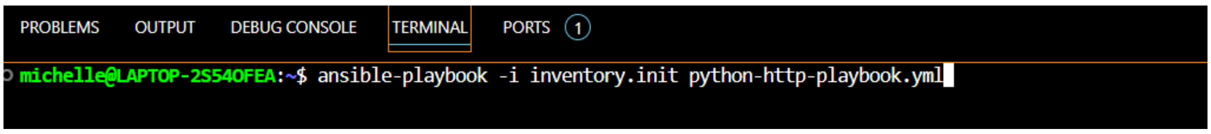

! python-http-playbook.yml ✕
ansible_project > ! python-http-playbook.yml

```yaml
---
- name: Set up a basic Python HTTP server
  hosts: servers
  become: yes
  tasks:
    - name: Install Python
      yum:
        name: python3
        state: present

    - name: Create a custom HTML file
      copy:
        content: |
          <html>
          <head><title>Welcome to My Python HTTP Server</title></head>
          <body>
          <h1>Hello from Python HTTP Server!</h1>
          <p>This page is served by a simple Python HTTP server, configured with Ansible.</p>
          </body>
          </html>
        dest: /home/michelle/index.html
        owner: michelle
        group: michelle

    - name: Start Python HTTP server
      shell: |
        nohup python3 -m http.server 8000 --directory /home/michelle > /dev/null 2>&1 &
```

Figure 4.2.12

Execution

To run this playbook, execute the following command from the control node:



```
michelle@LAPTOP-2S54OFEA:~$ ansible-playbook -i inventory.init python-http-playbook.yml
```

Figure 4.2.13

Once executed, the Python HTTP server will be running on port ‘8000’ of the remote server, serving the HTML file.

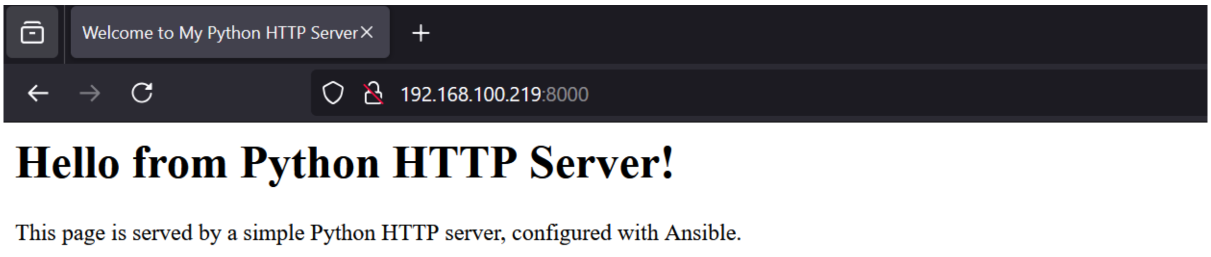
Verification

Checking if the server is running by accessing the remote server’s IP in your browser:

```
http://<remote_server_ip>:8000
```

Figure 4.2.14

The page should display:



Hello from Python HTTP Server!

This page is served by a simple Python HTTP server, configured with Ansible.

Figure 4.2.15

5. CONCLUSION

Ansible stands out as a powerful and versatile configuration management tool designed to simplify and automate the administration of IT infrastructure. Its minimalist design and use of YAML for playbooks make it accessible and intuitive, allowing users to create sophisticated automation scripts with ease. By leveraging SSH for communication and maintaining an idempotent approach to task execution, Ansible ensures that systems reach and maintain their desired state without unnecessary redundancy.

The flexibility of Ansible extends across multiple platforms, from various Linux distributions to Windows servers as managed nodes. Its support for dynamic inventories and modular roles enhances its adaptability and efficiency in managing complex environments. With its focus on simplicity, performance, and extensive community support, Ansible is a compelling choice for automating and orchestrating tasks, making it an invaluable tool for both small-scale and enterprise-level IT operations.