Project configuration management using Ansible

What is Ansible?

Ansible is an open-source IT automation tool that automates provisioning, configuration management, application deployment, orchestration, and many other manual IT processes. Ansible is the simplest solution for automating routine IT tasks.

- ➔ Ansible can be installed in specific operating systems i.e., CentOS or FEDORA, ubuntu, Debian, windows

Installing Ansible on Fedora:

$sudo dmf install ansible

Installing Ansible on CentOS:

$sudo yum install epel-release

$sudo yum install ansible

Installing Ansible on ubuntu:

$sudo apt update

$sudo apt install software-properties-common

$sudo add-apt-repository --yes --update ppa:ansible/ansible

$sudo apt install ansible

Installing Ansible on windows:

You cannot use a window operating system for ansible control node

Why Ansible is preferred?

Ansible makes it easy to prepare infrastructure such as cloud platforms or servers for application installation and configuration. It removes the strain associated with provisioning hundreds of servers manually and allows for swift and reliable scaling of IT infrastructure.

Ansible Automation Platform makes it possible for users across an organization to create, test, and manage automation content through a powerful and agentless framework. It is a more secure, stable, and flexible foundation for deploying end-to-end automation solutions, from IT processes, to hybrid cloud, to the edge.

How it works?

Ansible works by connecting to your nodes and pushing out small programs called modules to these nodes. Modules are used to accomplish automation tasks in Ansible. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules and removes them when finished.

Without modules, you'd have to rely on ad-hoc commands and scripting to accomplish tasks. Ansible can be installed on Red hat Enterprise Linux, CentOS, or Fedora; Ubuntu; Debian; and many other operating systems.

Ansible contains built-in modules that you can use to automate tasks, or you can write your own. Ansible modules can be written in any language that can return JSON, such as Ruby, Python, or bash. Windows automation modules are even written in PowerShell.

Nodes:

Ansible control nodes are primarily used to run tasks on managed hosts. You can use any machine with Python installed as an Ansible control node. However, you cannot use Windows as an Ansible control node.

1.Control node

2.Managed node

Control node:

The machine from which you run the Ansible CLI tool. You can use any computer that meets the software requirements as a control node - laptops, shared desktops, and servers can all run Ansible. Multiple control nodes are possible, but Ansible itself does not coordinate across them.

Managed nodes:

Also referred to as 'hosts', these are the target devices (servers, network appliances or any computer) you aim to manage with Ansible. Ansible is not normally installed on managed nodes, unless you are using ansible-pull, but this is rare and not the recommended setup.

Inventory:

A list of managed nodes provided by one or more 'inventory sources. Your inventory can specify information specific to each node, like IP address. It is also used for assigning groups, that both allow for node selection in the Play and bulk variable assignment. Sometimes an inventory source file is also referred to as a 'host file'.

Modules:

A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. Each module can be used by the Ansible API, or by the ansible or ansible-playbook programs. A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.

If you need functionality that is not available in any of the thousands of Ansible modules found in collections, you can easily write your own custom module. When you write a module for local use, you can choose any programming language and follow your own rules. After you create a module, you must add it locally to the appropriate directory so that Ansible can find and execute it.

Tasks

In Ansible, a task is an individual unit of work to execute on a managed node. Each action to perform is defined as a task. Tasks can be executed as a one-off action via ad-hoc commands, or included in a playbook as part of an automation script.

Playbook

A playbook contains an ordered list of tasks, and a few other directives to indicate which hosts are the target of that automation, whether or not to use a privilege escalation system to run those tasks, and optional sections to define variables or include files. Ansible

executes tasks sequentially, and a full playbook execution is called a play. Playbooks are written in YAML format.

Handlers

Handlers are used to perform actions on a service, such as restarting or stopping a service that is actively running on the managed node's system. Handlers are typically triggered by tasks, and their execution happens at the end of a play, after all tasks are finished. This way, if more than one task triggers a restart to a service, for instance, the service will only be restarted once and after all tasks are executed. Although the default handler behaviour is more efficient and overall, a better practice, it is also possible to force immediate handler execution if that is required by a task.

Roles

A role is a set of playbooks and related files organized into a predefined structure that is known by Ansible. Roles facilitate reusing and repurposing playbooks into shareable packages of granular automation for specific goals, such as installing a web server, installing a PHP environment, or setting up a MySQL server.

YAML:

YAML is a human-readable data serialization language that is often used for writing configuration files. Depending on whom you ask, YAML stands for yet another markup language or YAML ain't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents.

YAML is a popular programming language because it is designed to be easy to read and understand. It can also be used in conjunction with other programming languages. Because of its flexibility and accessibility, YAML is used by the Ansible automation tool to create automation processes, in the form of Ansible Playbooks.

- YAML files use a .yml or .yaml extension
- There are no usual format symbols, such as braces, square brackets, closing tags, or quotation marks.
- And YAML files are simpler to read as they use Python-style indentation to determine the structure and indicate nesting
- Tab characters are not allowed by design, to maintain portability across systems, so whitespaces-literal space characters-are used instead.
- Comments can be identified with a pound or hash symbol (#).
- YAML does not support multi-line comments
- 3 dashes (---) are used to signal the start of a document, while each document ends with three dots (...). Example YAML CODE:

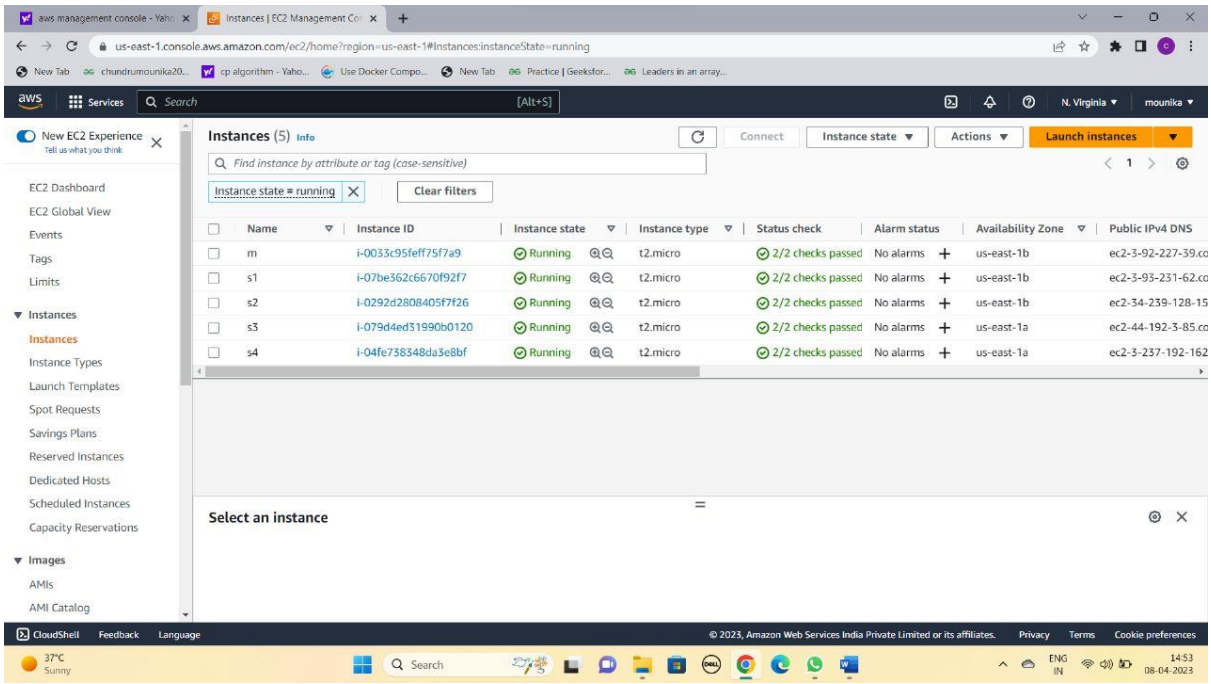
```
#comment: This is a supermarket list using YAML

---

Food:

 -   Vegetables: tomatoes
 -   Fruits:
         Critics:  oranges
         Tropical: bananas
         Nuts: peanuts
         Sweets: raisins

…
```

PROJECT CONFIGURATION WITH ANSIBLE

STEP-1: Creation of EC2 instances

Launch 5 instances with ubuntu AMI, set up the security group with SSH and All traffic and select your key pair.

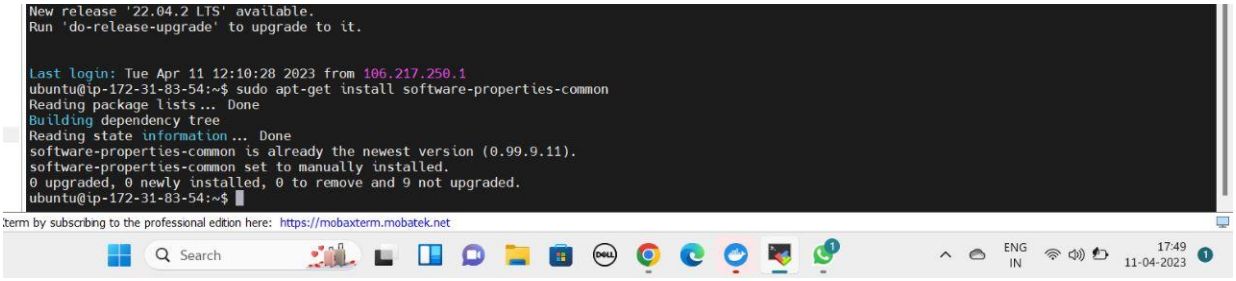
In the 5 instance one instance act as master and another 4 instances act slave or worker.



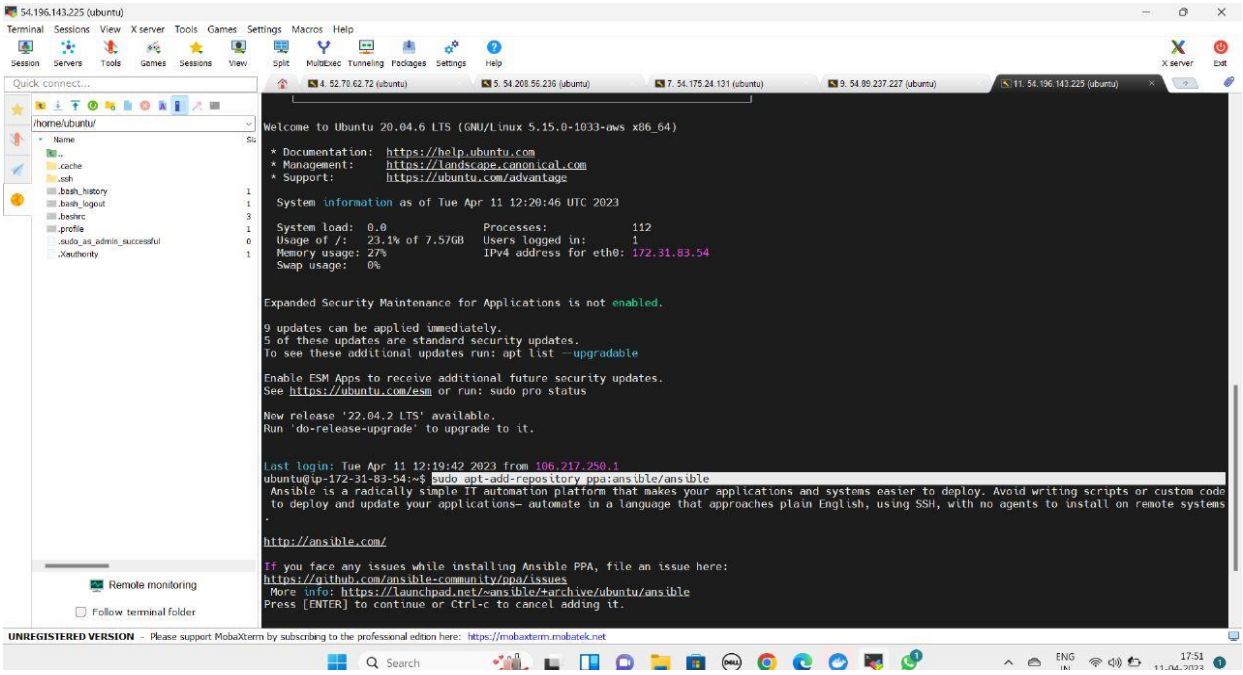
STEP-2: connect the instances using mobaXterm or putty.

STEP-3: Installation of ansible on master by using this commands. $sudo apt update

$sudo apt-get install software-properties-common

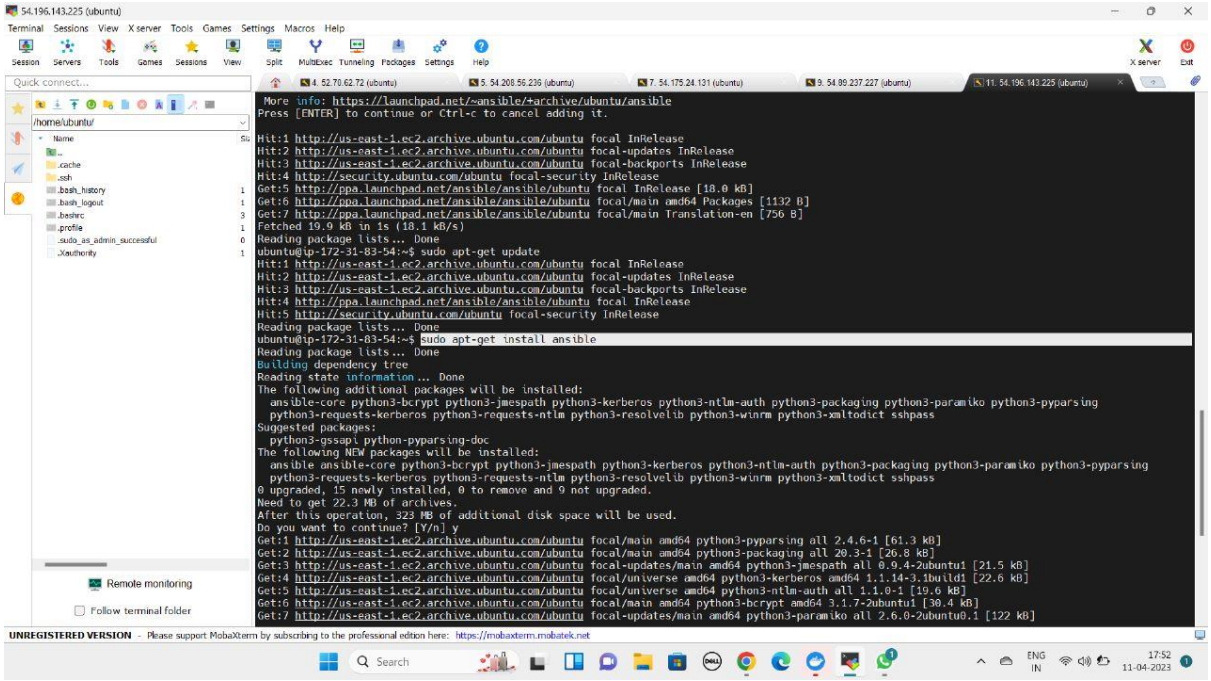


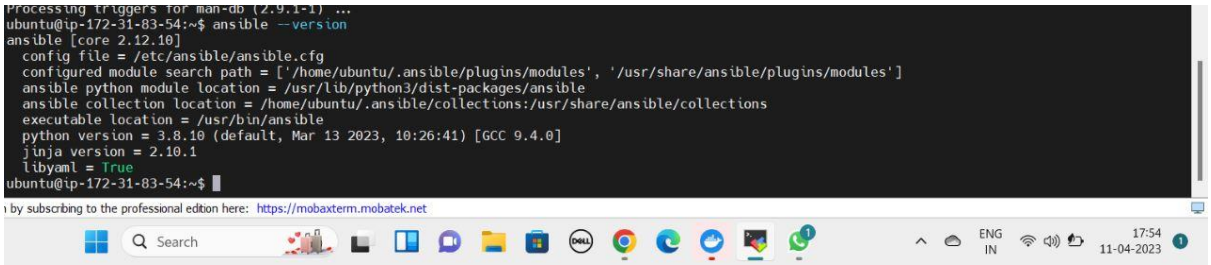
$sudo apt-add-repository ppa:ansible/ansible



$sudo apt-get update

$sudo apt-get install ansible



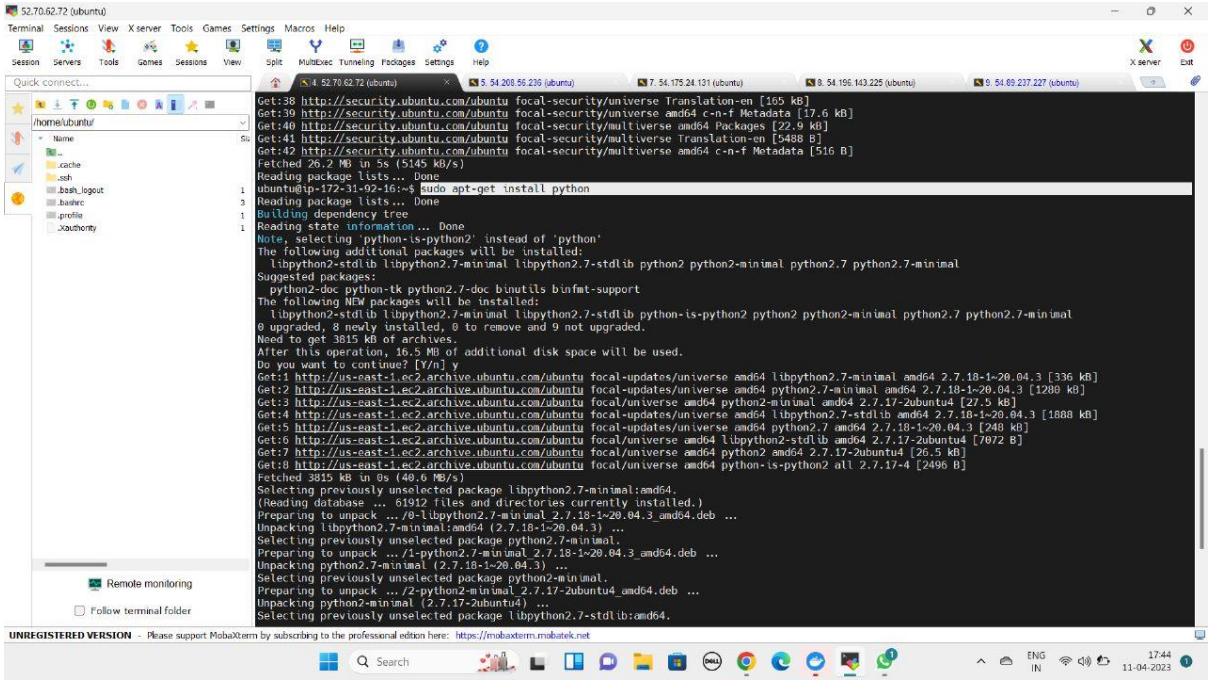


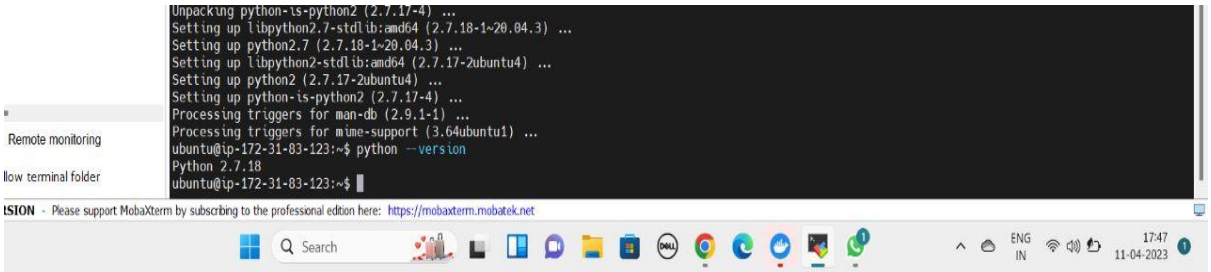
Installation of phyton on hosts by using this commands

$sudo apt-get update

$sudo apt-get install python

If you want check the python version then the command is “$python –version”

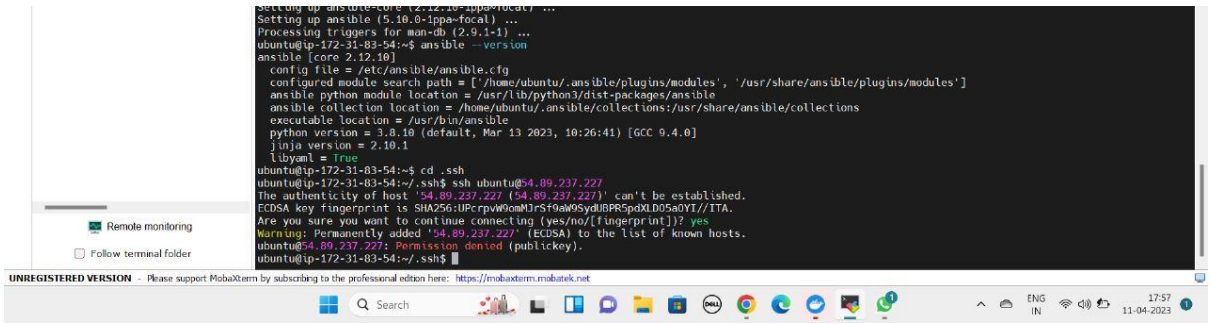




STEP-4: Configure SSH access to ansible host

Commands on master:

$ssh ubuntu@<public Ip address of host>

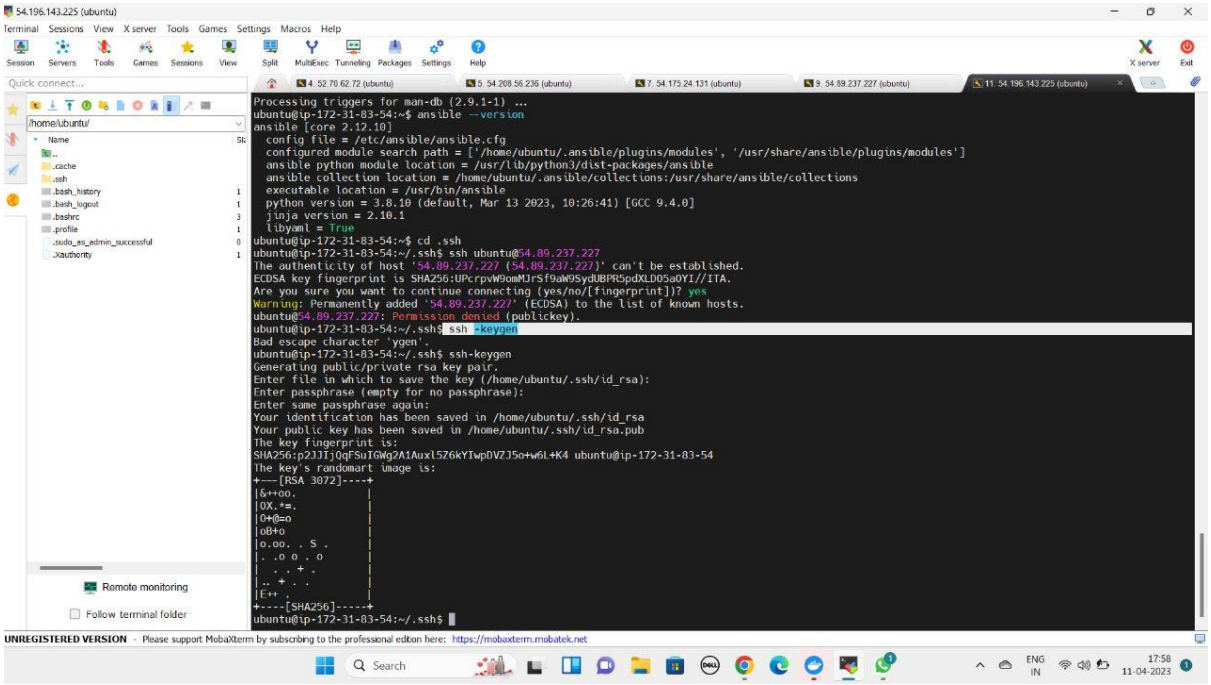


The output of this command shows permission denied of connecting master to the slave. To connect master to the slave, we need to add the master SSH key to the slave.

STEP-5: Enable keyless access to accept the connection from the master node.

5.1: Go inside the SSH directory by using this command “$cd .ssh” and do “$ls”, there are two files exists one is authorized_keys and another file is known_hosts.

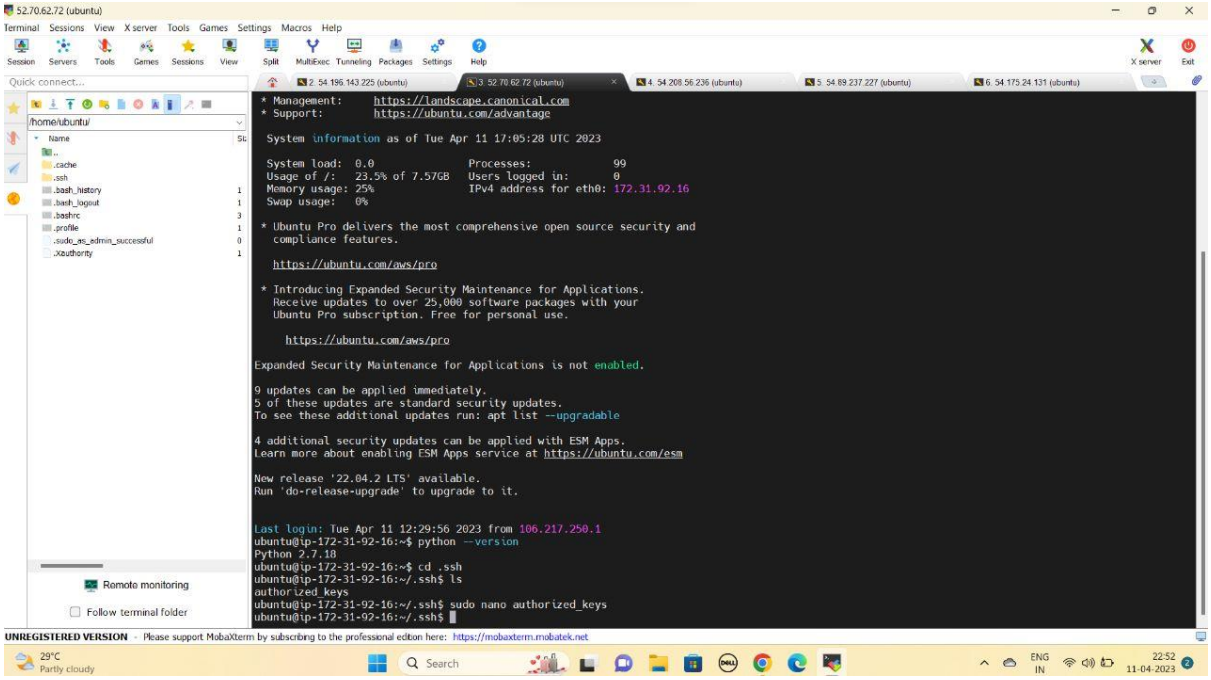
5.2: Generation of key in a system “$ssh-keygen” This command creates the unique key for the machine



5.3: Do “$ls” and we can see two more files are added i.e., “id_rsa” and “id_rsa.pub”

5.4: Copy the code inside the id_rsa.pub using “$cat id_rsa.pub”

5.5: Now in the slave node go to SSH directory and do ls and go inside the authorized_keys using “$sudo nano authorized_keys” and paste the code which is copied previous and save the file and exit



5.6: Now we can configure SSH access in master machine using the command "$ssh ubuntu@<ip_address>" by this command we can successfully login to the slave machine in master and do exit

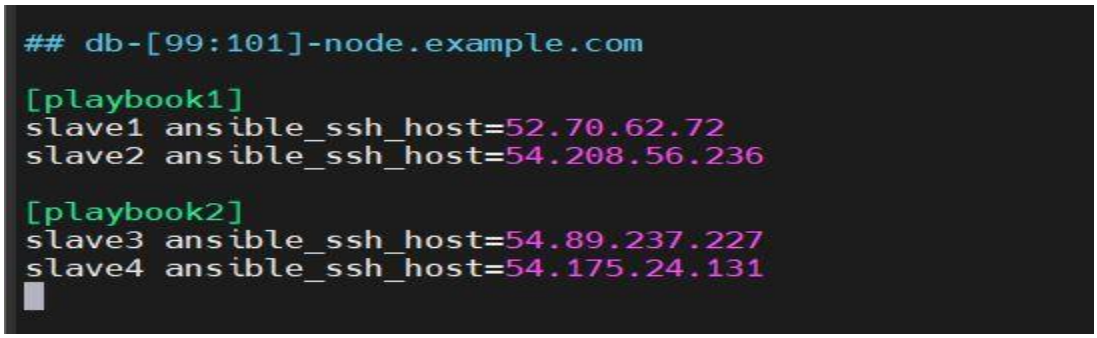
STEP-6: Setting up ansible host and testing connection

For setting up ansible host we need to modify the inventory file, the location of this file is /etc/ansible/hosts. All the commented lines inside the inventory file are templates

$sudo nano /etc/ansible/hosts

[<group_name>]

<slave name> ansible_ssh_host=<ip_address>



We can ‘n’ number slave hosts to the ansible host

$ansible -m ping all

Or $ansible -m ping <group name>

```
ubuntu@ip-172-31-83-54:~/.ssh$ cd ..
ubuntu@ip-172-31-83-54:~$ ansible -m ping all
slave3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
slave2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
slave1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
slave4 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ip-172-31-83-54:~$
```

This command used to verify whether the master is communicating with the slave

STEP-6: Creation of playbooks

We can create playbooks using .yml or .yaml extension and inside the playbook the tasks or actions are written in sequential order using YAML code

6.1 Installing nginx using yaml

```
ubuntu@ip-172-31-83-54:~$ cat ngnix.yml
---
- name: nginx install & start services
  hosts: all
  become: true


  tasks:
  - name: install nginx
    apt:
      name: nginx
      state: latest

  - name:  start nginx
    service:
      name: nginx
      state: started
```

- name: nginx install & start services

  hosts: all

  become: true

  tasks:

  - name: install nginx

    apt:

      name: nginx

      state: latest

  - name: start nginx

    service:

      name: nginx

      state: started

6.2: Installing git using yaml code

```
- name: Install Git on hosts
  hosts: squad
  become: true
  tasks:
    - name: Install Git
      apt:
        name: git
        state: present
```

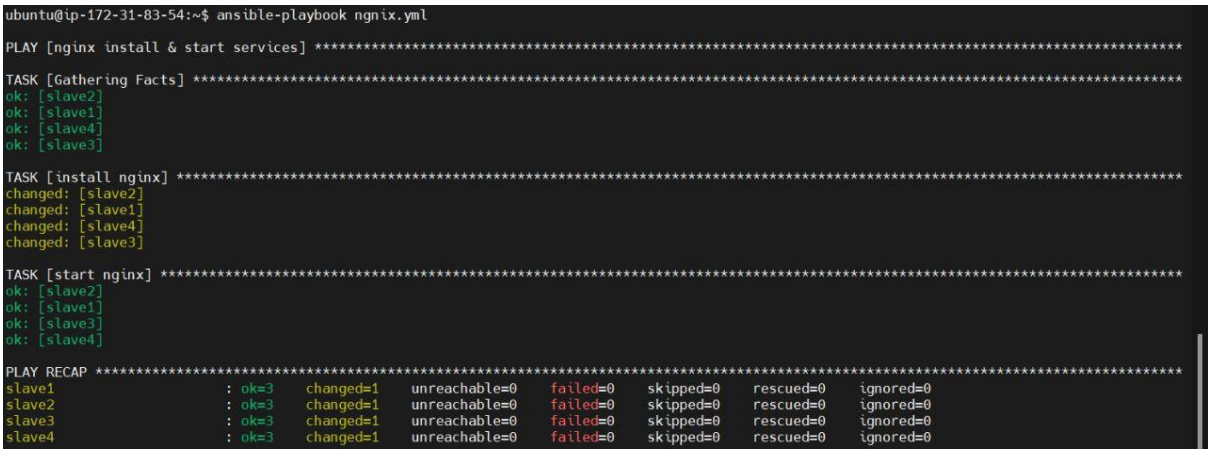
6.3: Installing Apache using yaml code

```
cat: apache,yml: No such file or directory
ubuntu@ip-172-31-83-54:~$ cat apache.yml
---
- hosts: playbook1
  become: yes
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest
    - name: enabled mod_rewrite
      apache2_module: name=rewrite state=present
      notify:
        - restart apache2
  handlers:
    - name: restart apache2
      service: name=apache2 state=restarted
```

```
---
- hosts: playbook1
  become: yes
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest
    - name: enabled mod_rewrite
      apache2_module: name=rewrite state=present
      notify:
        - restart apache2
  handlers:
    - name: restart apache2
      service: name=apache2 state=restarted
```

STEP-7: Run the playbook by using this command

$ansible-playbook filename.yml



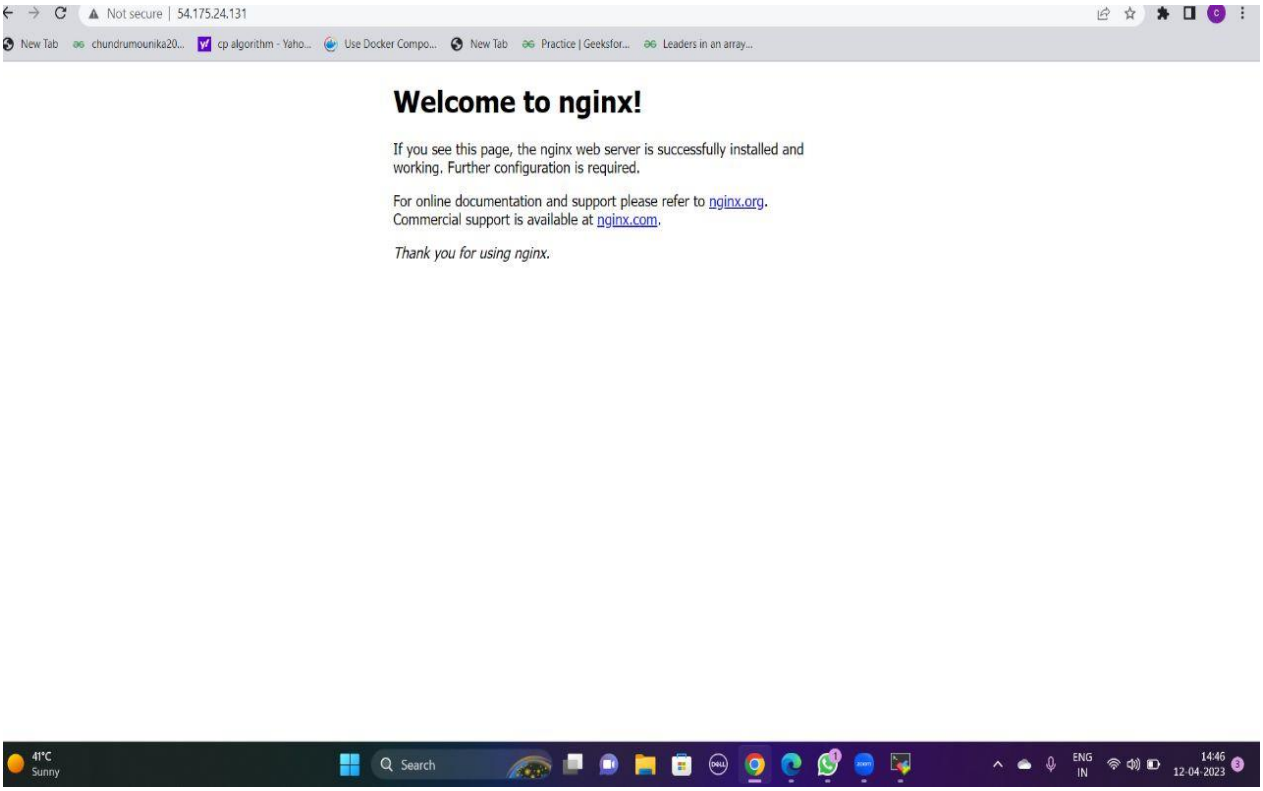
```
ubuntu@ip-172-31-83-54:~$ ansible-playbook ngnix.yml

PLAY [nginx install & start services] ***********************************************************************************

TASK [Gathering Facts] ***************************************************************************************************
ok: [slave2]
ok: [slave1]
ok: [slave4]
ok: [slave3]

TASK [install nginx] *****************************************************************************************************
changed: [slave2]
changed: [slave1]
changed: [slave4]
changed: [slave3]

TASK [start nginx] *******************************************************************************************************
ok: [slave2]
ok: [slave1]
ok: [slave3]
ok: [slave4]

PLAY RECAP ***************************************************************************************************************
slave1                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
slave2                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
slave3                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
slave4                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fig: running nginx playbook

```
ubuntu@ip-172-31-83-54:~$ ansible-playbook apache.yml

PLAY [playbook1] ***********************************************************************************************

TASK [Gathering Facts] *****************************************************************************************
ok: [slave2]
ok: [slave1]

TASK [install apache2] *****************************************************************************************
ok: [slave1]
ok: [slave2]

TASK [enabled mod_rewrite] *************************************************************************************
ok: [slave1]
ok: [slave2]

PLAY RECAP *****************************************************************************************************
slave1                     : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fig: running Apache playbook

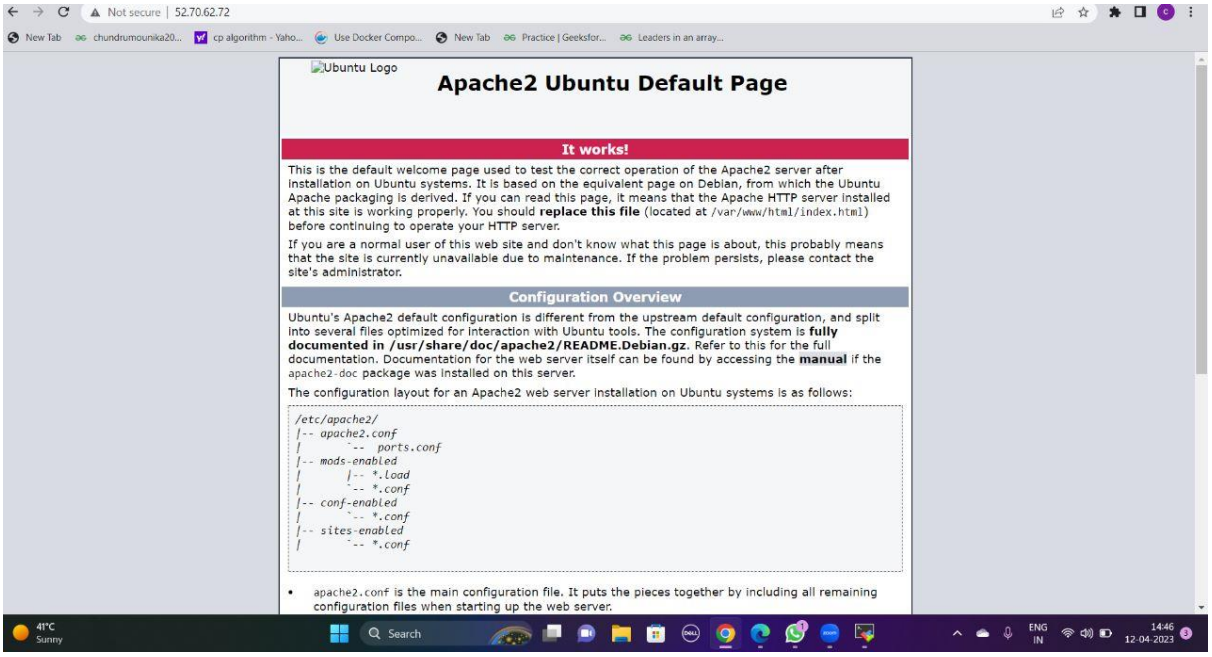
Outputs:



This page can be shown when any slave Ip address is copied and browse we get this page because while writing playbook we give host as all

<u>Nginx default page</u>

This page can be shown when only slave-1 and slave-2 Ip address is copied and browse we



get this page because while writing playbook we give host as playbook1

<u>Apache default page</u>

Webhosting using ansible

Step1- Initially we need to push the required html files into the git repository

Step2- Now in master node we need to clone the files to our desired location using the command

Git clone <repository URL>

```
ubuntu@ip-172-31-83-54:~$ git clone https://github.com/Sandhya-Akula/amazon.git
Cloning into 'amazon'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 34.96 KiB | 11.65 MiB/s, done.
ubuntu@ip-172-31-83-54:~$
```

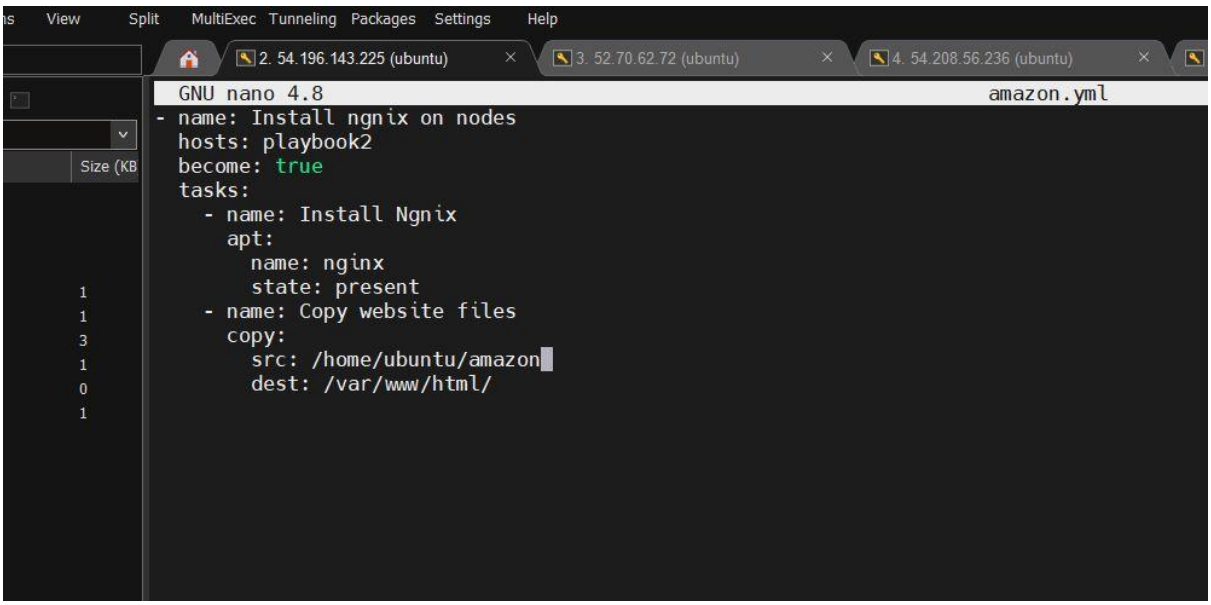
Step3- we need copy the path of the source file by using “pwd” command

```
ubuntu@ip-172-31-83-54:~$ cd amazon
ubuntu@ip-172-31-83-54:~/amazon$ pwd
/home/ubuntu/amazon
ubuntu@ip-172-31-83-54:~/amazon$
```

Step4- Now we need to go .ssh location using “cd .ssh” and create playbook using .yaml or .yml extension

Yaml code:

```
- name: Install ngnix on nodes
  hosts: playbook1
  become: true
  tasks:
    - name: Install Ngnix
      apt:
        name: nginx
        state: present
    - name: Copy website files
      copy:
```




```
        src: /home/ubuntu/amazon
        dest: /var/www/html/
```

step-4- Run the playbook by using command

ansible-playbook <filename>

step5- copy the slave node public Ip address copy in the new tab

```
ubuntu@ip-172-31-83-54:~/.ssh$ sudo nano amazon.yml
ubuntu@ip-172-31-83-54:~/.ssh$ ansible-playbook amazon.yml

PLAY [Install ngnix on nodes] ***********************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************************************
ok: [slave3]
ok: [slave4]

TASK [Install Ngnix] ********************************************************************************************************************
ok: [slave4]
ok: [slave3]

TASK [Copy website files] ***************************************************************************************************************
changed: [slave4]
changed: [slave3]

PLAY RECAP ******************************************************************************************************************************
slave3                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
slave4                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-83-54:~/.ssh$
```

“ <ipaddress>/souce-foldername” the output will the html page.