

# **PROJECT-4 ;**

## **JENKINS MASTER SLAVE CONFIGURATIONS**

### **Master Slave Configurations:**

In Jenkins, master-slave configuration (now referred to as controller-agent configuration) is a setup that allows you to distribute the build and test workload across multiple nodes (machines). This setup improves performance, scalability, and allows better resource utilization in CI/CD pipelines.

### **Advantages:**

#### **Better Resource Utilization:**

- By distributing workloads, you maximize the resource usage of the available machines. The master only handles the management, while the agents run the actual builds, freeing the master from performing all tasks.

#### **Improved Performance:**

- Distributing tasks reduces the time needed to complete jobs. Agents can run builds in parallel, reducing the overall build time in CI/CD pipelines.

#### **Scalability:**

- Adding more agent machines helps increase the capacity of your Jenkins environment without requiring upgrades to the master machine. This makes Jenkins highly scalable to meet increasing project demands.

#### **Separation of Concerns:**

- You can isolate environments for different purposes (e.g., building for Android/iOS, Java, or Python). This separation ensures the jobs run smoothly without interfering with each other.

#### **Fault Tolerance:**

- If one agent goes down, the workload can be shifted to other available agents, ensuring minimal disruption to your build process.

#### **Customizable Hardware for Specific Jobs:**

- You can tailor specific agents to run certain jobs that require different hardware or software (e.g., a high-performance agent for resource-intensive builds).

## **Step-by-Step Setup:**

### **Install Jenkins on the Master (Controller):**

- ❖ Download and install Jenkins on your primary machine (the controller node).
- ❖ Start Jenkins and set up the basic configurations

### **Prepare the Slave Machine (Agent):**

- ❖ Install the necessary software and dependencies on the slave machine (e.g., Java, Docker, or any other tools your jobs require).
- ❖ Ensure the slave machine can communicate with the master over the network (TCP/IP, usually on port 22 for SSH).

### **Create a New Agent in Jenkins:**

- ❖ Go to **Manage Jenkins > Manage Nodes > New Node**.
- ❖ Provide a name for the agent and select the type of agent you are setting up (usually “Permanent Agent”).
- ❖ Configure the connection details (remote directory, usage, launch method). Common methods include:
  - i. SSH: If the agent is a Linux/Unix-based system.
  - ii. Jenkins Swarm Plugin: If the agent machine uses a custom plugin to connect.
  - iii. Windows Agent: For Windows machines, you may need to use the JNLP protocol.

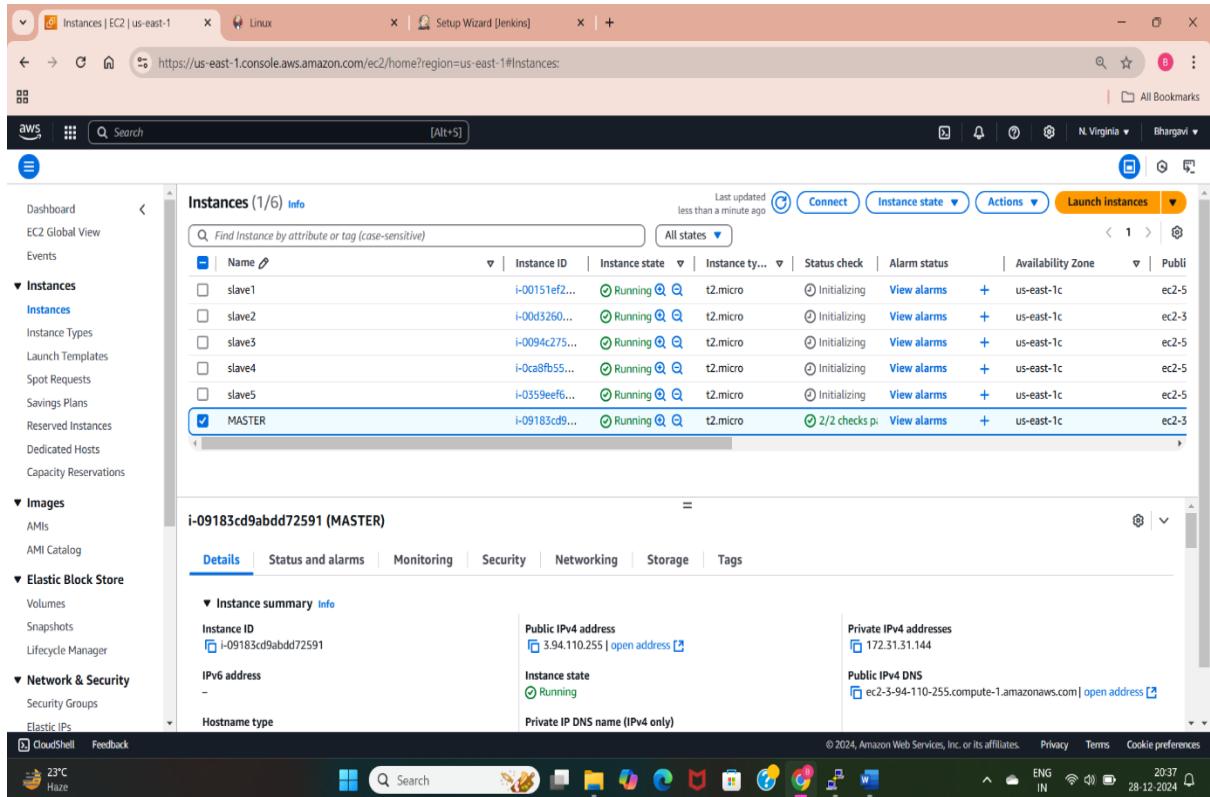
### **Launch the Agent:**

- ❖ After configuring the agent, you can either launch it manually or wait for it to connect automatically based on the configuration.
- ❖ You can verify the connection from the Manage Nodes section, where the agent will appear as connected.

## METHOD-1:

**Create One Master And Five Slave Nodes And Install Nginx (Slave-1), Httpd (Slave-2), Sonarqube (Slave-3), Tomcat (Slave-4) And Deploy Any One Static Application In Any One Of The Web Servers (Httpd & Nginx) (Slave-5)**

Launch an EC2 instance for the master server. Install Jenkins on the master (default port: 8080). Enable Jenkins service (sudo systemctl start jenkins).



The screenshot shows the AWS Management Console interface for the EC2 service. The left sidebar navigation includes 'Dashboard', 'EC2 Global View', 'Events', 'Instances' (selected), 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', and 'Capacity Reservations'. Under 'Instances', there are sub-options for 'Images', 'AMIs', 'AMI Catalog', 'Elastic Block Store' (with 'Volumes', 'Snapshots', and 'Lifecycle Manager'), and 'Network & Security' (with 'Security Groups' and 'Elastic IPs'). The main content area displays a table titled 'Instances (1/6) Info' with the following data:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
slave1	i-00151ef2...	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-5
slave2	i-00d3260...	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-3
slave3	i-0094c275...	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-5
slave4	i-0cafb55...	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-5
slave5	i-0359eef6...	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-5
<b>MASTER</b>	i-09183cd9...	Running	t2.micro	2/2 checks p	View alarms	us-east-1c	ec2-3

The 'MASTER' instance is highlighted with a blue selection bar. Below the table, a detailed view for the 'MASTER' instance is shown, including 'Details', 'Status and alarms', 'Monitoring', 'Security', 'Networking', 'Storage', and 'Tags'. The 'Details' tab is active, displaying information such as Instance ID (i-09183cd9abdd72591), Public IPv4 address (3.94.110.255), Instance state (Running), and Private IP DNS name (IPv4 only).

Here I have created five slave servers to setup master slave configuration. I have taken 4 amazon-linux, with three t2.micro instances types, and only slave3 is with t2.medium.

- After that install the java 17 and Jenkins in master server & start the Jenkins.
- Generate a key in the master server.
- Use the public IP of instance and with Jenkins port 8080 paste in browser and we can see the Jenkins setup there.

Run the command ssh-keygen on the master server to generate a public-private key pair.

```

Verifying : giflib-4.1.6-9.amzn2.0.2.x86_64
Verifying : libXinerama-xmono-fonts.noarch 0:2.0.2.x86_64
Verifying : libXext-xmono-fonts.noarch 0:2.33-6.amzn2
Verifying : log4j-cve-2021-44228-hotpatch-1.3+11-1.amzn2.noarch
Verifying : libXslt-1.1.28-6.amzn2.x86_64
Verifying : libxml-3.2.1+4.amzn2.0.6.x86_64
Verifying : python-javapackages-tools.noarch 0:3.4.1-11.amzn2
Verifying : libXext-x86_64 0:1.3+11-1.amzn2.x86_64
Verifying : alsa-lib-1.1.4.1-2.amzn2.x86_64
Verifying : libICE-1.0.9-9.amzn2.0.2.x86_64
Verifying : dejavu-sans-fonts.noarch 0:2.33-6.amzn2
dejavu-fonts-common.noarch 0:2.33-6.amzn2
dejavu-serif-fonts.noarch 0:2.33-6.amzn2
giflib-x86_64 0:4.1.6-9.amzn2.0.2
libICE.x86_64 0:1.0.9-9.amzn2.0.2
libX11-common.noarch 0:1.6.7-3.amzn2.0.5
libX11-x86_64 0:1.6.7-3+1.amzn2.0.2
libXinerama-x86_64 0:2.3.1-2.amzn2.0.2
libXext-x86_64 0:1.2.3-1.amzn2.0.2
libXtst-x86_64 0:1.2.3-1.amzn2.0.2
python-javapackages.noarch 0:3.4.1-11.amzn2
python-javapackages-tools.noarch 0:3.4.1-11.amzn2
log4j-cve-2021-44228-hotpatch.noarch 0:1.3-7.amzn2

Dependency Installed:
alsa-lib.x86_64 0:1.1.4.1-2.amzn2
dejavu-sans-fonts.noarch 0:2.33-6.amzn2
fontconfig-filesystem.noarch 0:1.44-8.amzn2
javapackages-tools.noarch 0:3.4.1-11.amzn2
libX11.x86_64 0:1.6.7-3.amzn2.0.5
libXext.x86_64 0:1.3+11-1.amzn2.0.2
libXinerama-x86_64 0:2.3.1-2.amzn2.0.2
libXtst-x86_64 0:1.2.3-1.amzn2.0.2
log4j-cve-2021-44228-hotpatch.noarch 0:1.3-7.amzn2

Completed!
[ec2-user@ip-172-31-18-14 ~]$ sudo hostname slave1
[ec2-user@ip-172-31-18-14 ~]$ eexec bash
-bash: eexec: command not found
[ec2-user@ip-172-31-18-14 ~]$ exec bash
[ec2-user@slave1 ~]$ cd .ssh/
[ec2-user@slave1 .ssh]$ ll
total 4
-rw----- 1 ec2-user ec2-user 389 Dec 28 16:01 authorized_keys
[ec2-user@slave1 .ssh]$ sudo vi authorized_keys
[ec2-user@slave1 .ssh]$ sudo visudo
[ec2-user@slave1 .ssh]$ history
 1 sudo yum install java-17
 2 sudo hostname slave1
 3 eexec bash
 4 exec bash
 5 .ssh/
 6 11
 7 sudo vi authorized_keys
 8 sudo visudo
 9 history
[ec2-user@slave1 .ssh]$ 

```

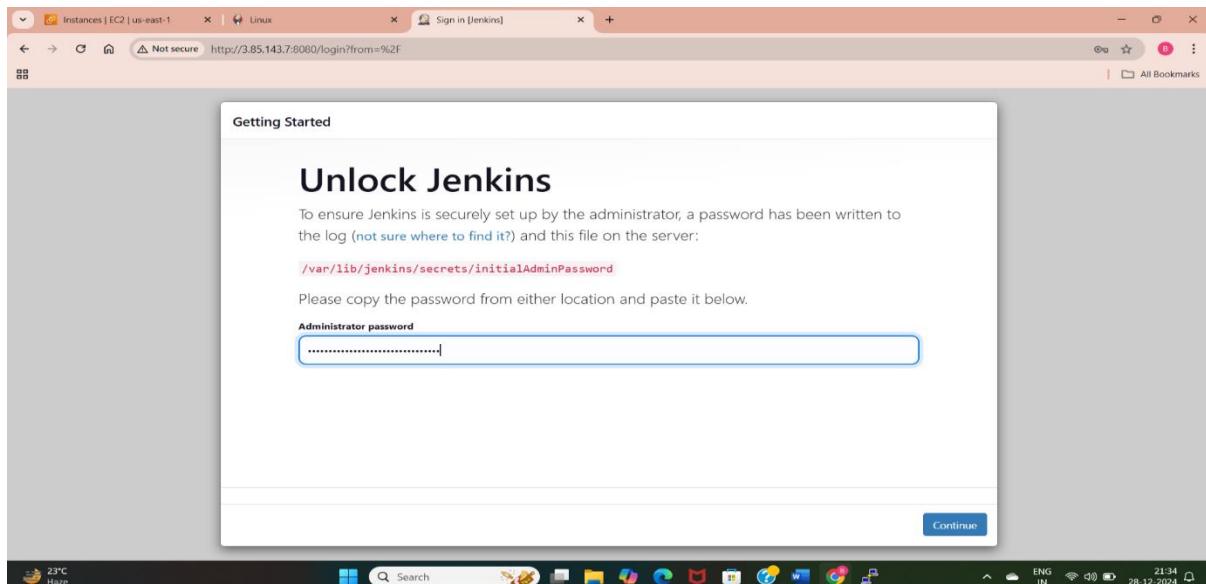
The terminal shows the execution of a Java update command and its dependencies. It also shows the creation of an SSH key pair and the configuration of the slave server's hostname.

This creates:

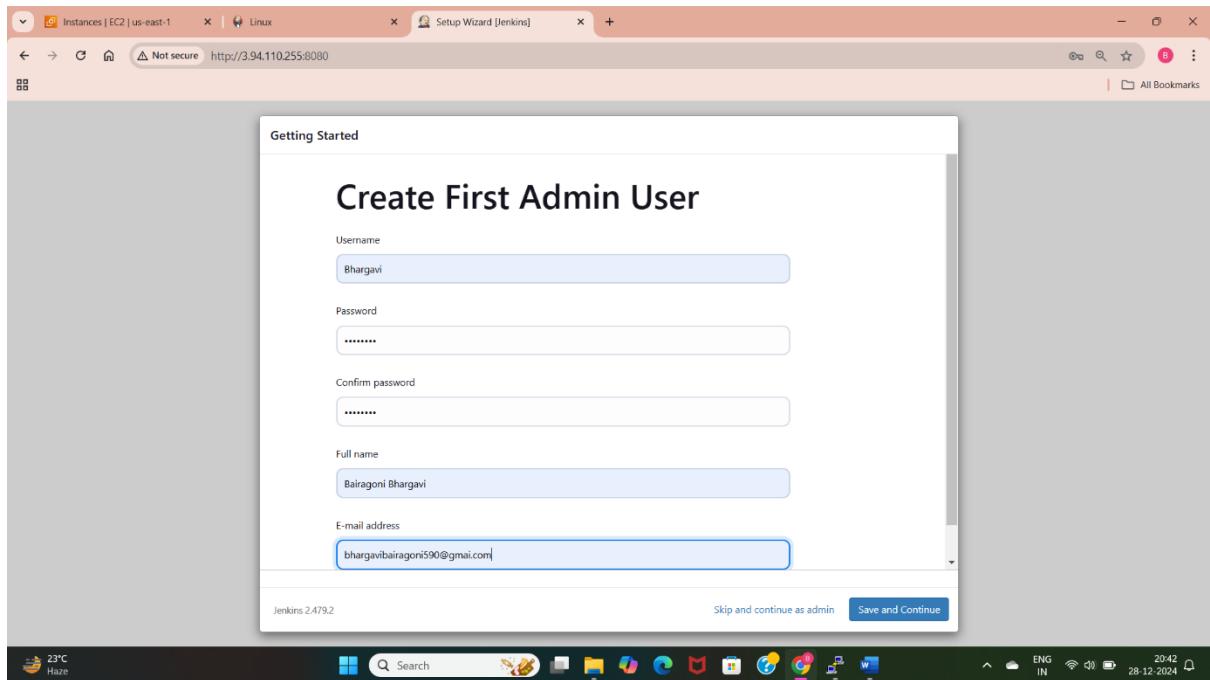
- Public key: Stored in `~/.ssh/id_rsa.pub`.
- Private key: Stored in `~/.ssh/id_rsa`.

Launch EC2 instances for the slave servers. Assign appropriate names using `<sudo hostname slave4>` and `<exec bash>`.

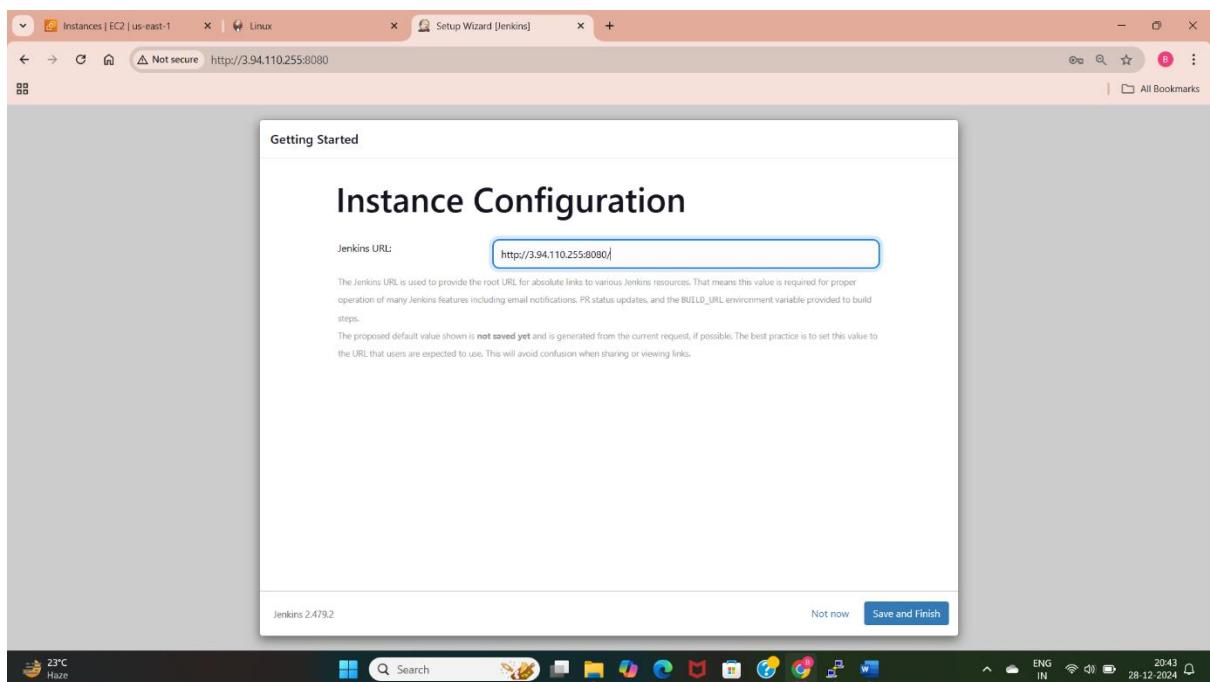
- On the master server, copy the contents of the public key `<cat .ssh/id_rsa.pub>`.
- On each slave server, paste the public key into the `.ssh/authorized_keys` file using `< sudo vi .ssh/authorized_keys>`. Use shift + A to append and paste the key, then save the file.
- On all servers (master and slaves), install Java as Jenkins requires it to run using `<sudo yum install java-17-amazon-corretto -y>`.
- On the master server, open Jenkins in your browser: `http://<master-server-public-ip>:8080`.



- First we need to give the password there using the above file path in the terminal as `<sudo cat filepath>` we will get a password there copy and paste in Jenkins.



- Create a Jenkins user and password so we can build jobs using this Jenkins account.



- This page is showing the Jenkins url which is “`http://public IP: jenkins port(8080)`”, save and finish the settings to start using Jenkins.

## Slave-1:

```

Verifying : giflib=4.1.6-9.amzn2.0.2.x86_64
Verifying : libXinerama=1.1.2-1.amzn2.0.x86_64
Verifying : dejavu-sans-mono-fonts=2.33-6.amzn2.noarch
Verifying : log4j-cve-2021-44228-hotpatch=1.3-7.amzn2.noarch
Verifying : libXcursor=1.1.1-1.amzn2.0.x86_64
Verifying : libXslt=1.1.28-6.amzn2.x86_64
Verifying : python-lxml=3.2.1-4.amzn2.0.x86_64
Verifying : python-javapackages=3.4.1-11.amzn2.noarch
Verifying : libXtst=1.2.3-1.amzn2.0.x86_64
Verifying : alsalib=1.1.4.1-2.amzn2.x86_64
Verifying : libICE=1.0.9-9.amzn2.0.x86_64
Verifying : javapackages-tools=3.4.1-11.amzn2.noarch

Installed:
  java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1

Dependency Installed:
  alsalib.x86_64 0:1.1.4.1-2.amzn2
  dejavu-sans-fonts.noarch 0:2.33-6.amzn2
  fontpackages-filesystem.noarch 0:1.44-8.amzn2
  javapackages-tools.noarch 0:3.4.1-11.amzn2
  libX11.x86_64 0:1.6.7-3.amzn2.0
  libXcursor.x86_64 0:1.1.5-1.amzn2.0
  libXrandr.x86_64 0:1.5.1-2.amzn2.0.3
  libXtst.x86_64 0:1.2.3-1.amzn2.0.2
  log4j-cve-2021-44228-hotpatch.noarch 0:1.3-7.amzn2

Complete!
[ec2-user@ip-172-31-18-14 ~]$ sudo hostname slave1
[ec2-user@ip-172-31-18-14 ~]$ exec bash
-bash: exec: command not found
[ec2-user@ip-172-31-18-14 ~]$ exec bash
[ec2-user@slave1 ~]$ cd /ssh/
[ec2-user@slave1 .ssh]$ ll
total 4
-rw----- 1 ec2-user ec2-user 389 Dec 28 16:01 authorized_keys
[ec2-user@slave1 .ssh]$ sudo vi authorized_keys
[ec2-user@slave1 .ssh]$ sudo visudo
[ec2-user@slave1 .ssh]$ history
 1 sudo yum install java-17 -y
 2 sudo hostname slave1
 3 exec bash
 4 exec bash
 5 cd .ssh/
 6 ll
 7 sudo vi authorized_keys
 8 sudo visudo
 9 history
[ec2-user@slave1 .ssh]$ 

```

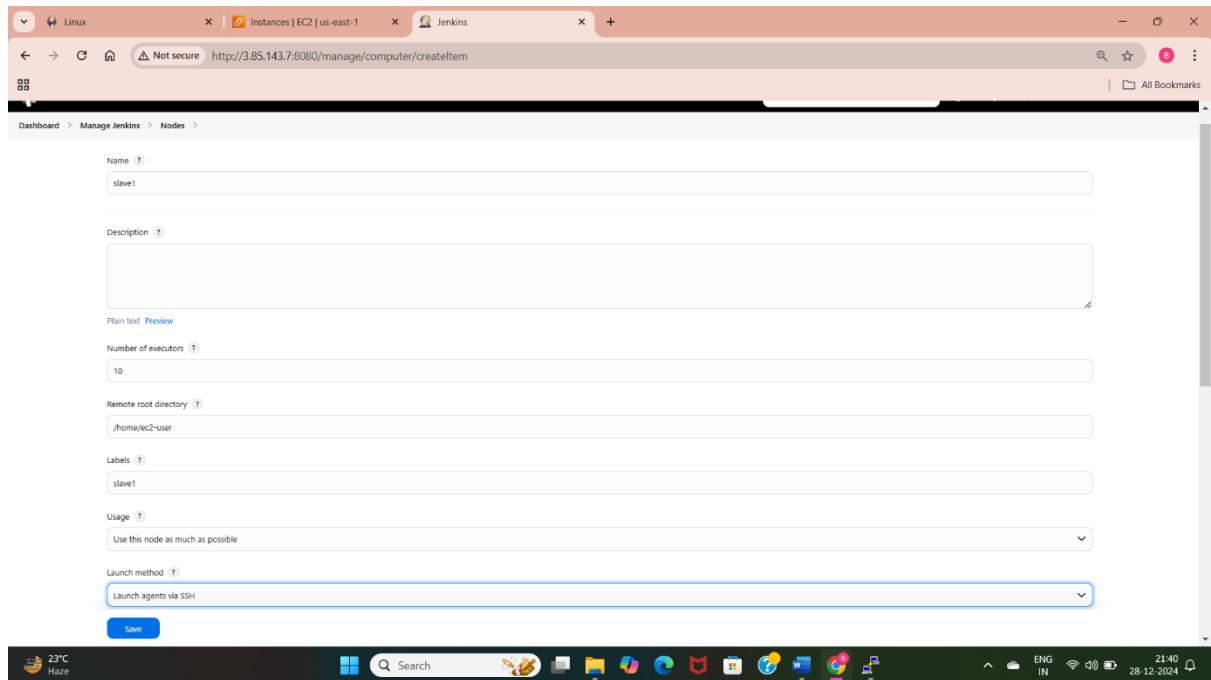
- Now I have connected the slave 1 server with the terminal. Install the Jenkins dependency (java 17) here to set the master slave configurations.
- Here I have changed the hostname in terminal to slave1 using <sudo hostname slave1> and updated the changes to terminal using <exec bash>.
- Now we need to connect the master with slave by copying the public key in the authorized key of slave1 server.

```

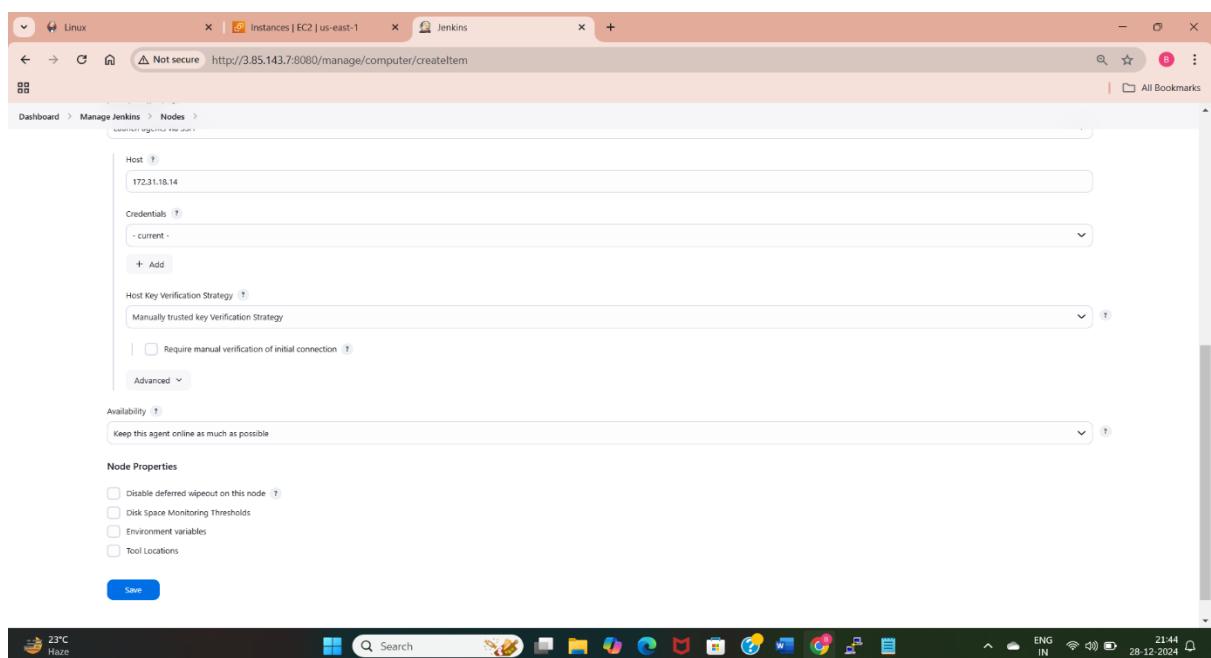
ssh-rsa AAAAB3NzaC1yc2EAAAQABQJQm+37T84142d/enh1Osuo/MUPApSkrVcmhCPk7d1YrKjQs9F9xlgWn/hQmcLSQxbj6REz2h8zVQ0dCEh6RtSB8lm2cKaZ9fdHR0sXmZpOFW03UUSoRaemMwiaiUiVjtqdxsUffBFBM1/N0wJscQRk125kKa1pAN9+bpdi6b613P2mhM+01t/P2j22DyMbpl+i+17zz3z/EdboVtc926tHrM2gOy8KUqd46FJYBLkfVnIY+a9grxxvzNHqPmUsv5kLWMMOFVmL/gyN36LfvyJ5zFmYyvGKwqfLgtPQQ7jaofPFXt5efEKvxzeDBDpSuXBbuZKzSLqb7Rm+hJyrd9r/B3 new-key
ssh-rsa AAAAB3NzaC1yc2EAAAQABQGKVUin1Cs1bz/R4cUW05xemb6exhv2W5VnKY70hnvlz21WH6b6UEf6K30mnEZ0cjFCE1T+2ai1T7ikulMzI20GJXgFNTcy2a3n2VnHM6EcSTfgVVN2mlkBcbacyw3RvDjclscFWA5v12NsJyvSN/8FrIt7WnSt5GW0GicKc335tphbnizZN1JNvVOs3HUGR0swvKd8BMSUQf1rSV/J3jT2AcQH03JxEohkb7jerSp/jnSE6+QgKTy4eqBY7ooyW+yk8SiRmPxG0C8tBNE4b7kGeis8HEA3jggZx6Khv0fIKWE1H4InVuNiflICp41zTuAwj1kYb
ec2-user@ip-172-31-19-44:ec2.internal

```

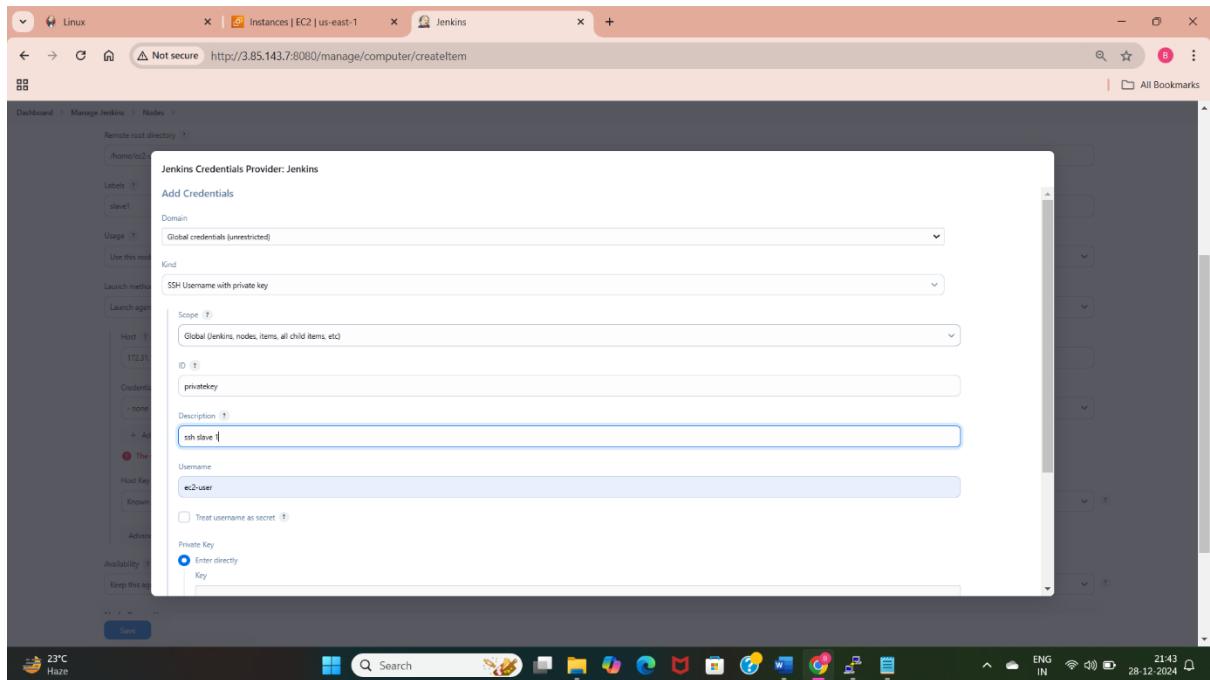
- To connect the master with the slave we need to copy the public key of instance and paste in slave server authorized\_keys and save.



- ✓ Next we need to create a node in Jenkins using the above slave1 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave1”, select the number of executors(no. of builds) according to requirement.
- ✓ Give the remote root directory of the slave1 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



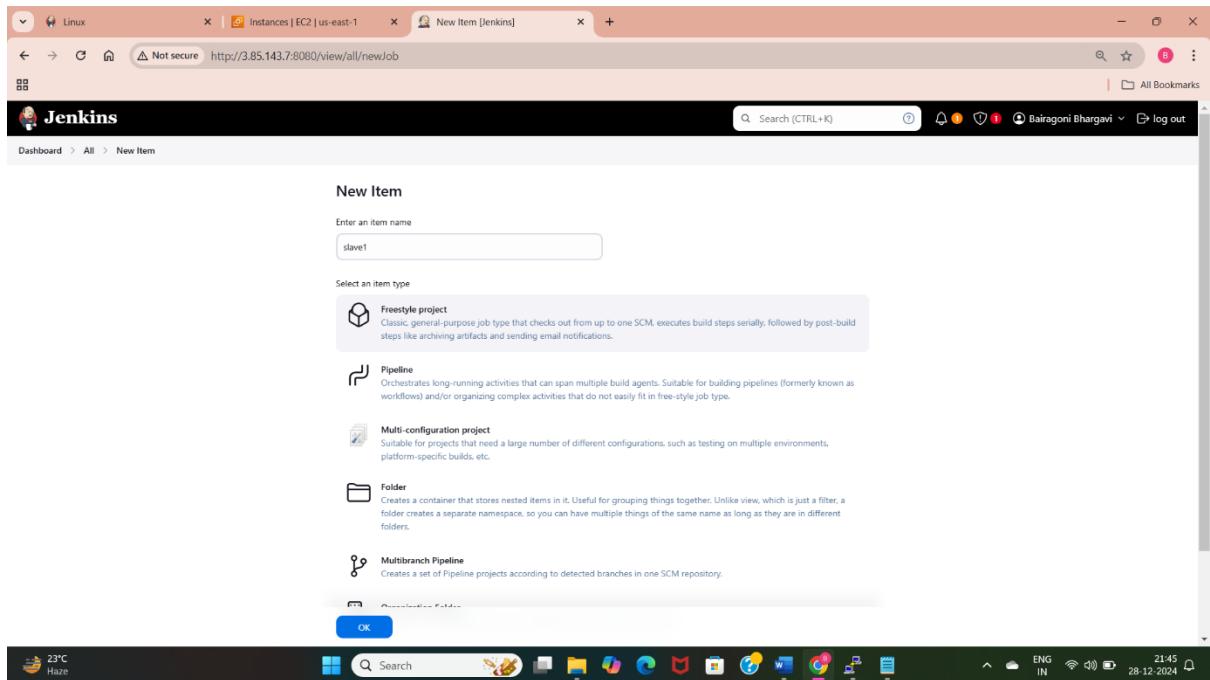
- Next select the host as private IP of slave1 server.
- Add credentials by following the below method.



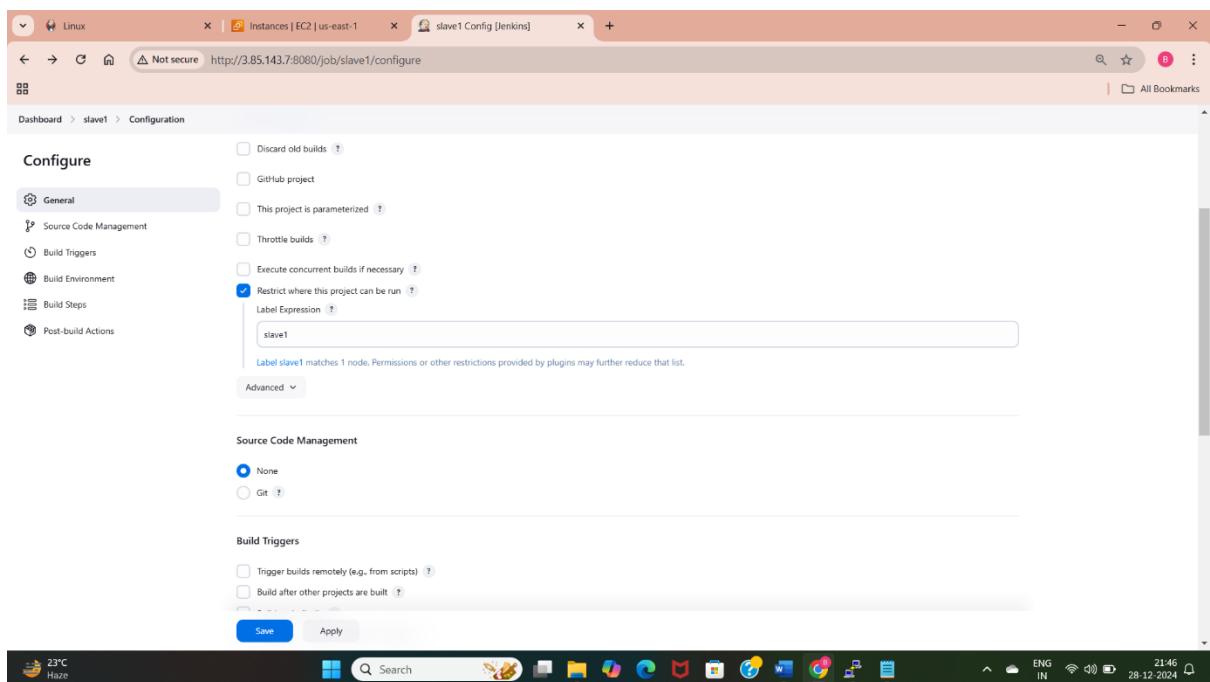
- Here am creating the credentials for this slave1 node. Select kind as “SSH Username with private key”.
- Next give any ID or description for this.
- Give the username. Here we need to give the username of our instance i.e ec2-user.
- Select “private key” and add the key using add option. Copy the private key which is stored in our local machine to Jenkins credentials, and save the changes.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.70 GB	0 B	5.70 GB	0ms
	slave1	Data obtained	N/A	N/A	N/A	N/A	N/A
			10 min	10 min	10 min	10 min	10 min

- This is the slave1 node which is created by me using master slave configurations with some configurations.



- ✓ Create a slave1 job by selecting new item, give the name for job ans select any one of the type like freestyle project.



- ✓ Select the above created slave1 by using "Restrict where this project can be run" and select the label name which is created above.

The screenshot shows the Jenkins job configuration interface. Under the 'Build Steps' section, there is a 'Execute shell' step with the following commands:

```
sudo amazon-linux-extras install nginx -y
#sudo yum install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx
```

Below the build steps, there is a 'Post-build Actions' section with a 'Save' button.

- ✓ Now go to build steps> select Execute shell and give the commands to install nginx , start nginx and save and build the job.

The screenshot shows a terminal window displaying the contents of the /etc/sudoers file. The file contains various sudo permissions, including one for the Jenkins user:

```
Defaults env_reset
Defaults env_keep += "MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE"
Defaults env_keep += "LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES"
Defaults env_keep += "LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE"
Defaults env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY"

# Allows anyone to run any command as root
# commands via sudo.

# Defaults: www-data ALL=(ALL) NOPASSWD:ALL

Defaults secure_path = /sbin:/bin:/usr/sbin:/usr/bin

## Root entries (the main part) which defines what software on
## which machines (the hosts) file can be shared between multiple
## systems).
## Entries:
##   user    machine=command
## The COMMAND section may have other options added to it.
## Allow root to run any command anywhere
root    ALL=(ALL)      ALL

## Allows members of the 'sys' group to run networking, software,
## service management, and more.
# Allow ALL=(ALL) NOPASSWD:ALL

## Allows members of the users group to run all commands
#wheel  ALL=(ALL)      ALL

## Some things require a password
#wheel      ALL=(ALL)      NOPASSWD:ALL

## Allows members of the users group to mount and unmount the
## cdrom as root.
# Allow ALL=(ALL) NOPASSWD:/sbin/mount /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# Allow ALL=(ALL) NOPASSWD:/sbin/shutdown -h now

## Need drop-in files from /etc/sudoers.d (item # does not need a comment)
#includedir /etc/sudoers.d
jenkins All=(ALL) NOPASSWD: ALL
```

- ✓ Now give sudo permissions for Jenkins using “sudo visudo” and give the permission using “Jenkins ALL=(ALL) NOPASSWD: ALL”.

```

Started by user Bairagoni Bhargavi
Running as SYSTEM
Building remotely on slave1 in workspace /home/ec2-user/workspace/slave1-job
[slave1-job] $ /bin/sh -xe /tmp/jenkins15226867589073338435.sh
+ sudo amazon-linux-extras install nginx -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Clearing repos: amzn2-core amzn2extra-docker amzn2extra-kernel-5.10
: amzn2extra-nginx
17 metadata files removed
6 sqlite files removed
0 metadata files removed
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package nginx.x86_64 1:1.22.1-1.amzn2.0.4 will be installed
--> Processing Dependency: nginx-com = 1:1.22.1-1.amzn2.0.4 for package: 1:nginx-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: nginx-filesystem = 1:1.22.1-1.amzn2.0.4 for package: 1:nginx-1.22.1-1.amzn2.0.4.x86_64
--> Running transaction check
--> Package nginx-core.x86_64 1:1.22.1-1.amzn2.0.4 will be installed
--> Processing Dependency: libcrypto.so.1.1(OPTIONSSL_1_1_0)(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: libssl.so.1.1(OPTIONSSL_1_1_0)(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: libss1.so.1.1(OPTIONSSL_1_1_0)(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: libcrypto.so.1.1()(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: libprofiler.so.0()(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64
--> Processing Dependency: liblber.so.1.1.1(64bit) for package: 1:nginx-core-1.22.1-1.amzn2.0.4.x86_64

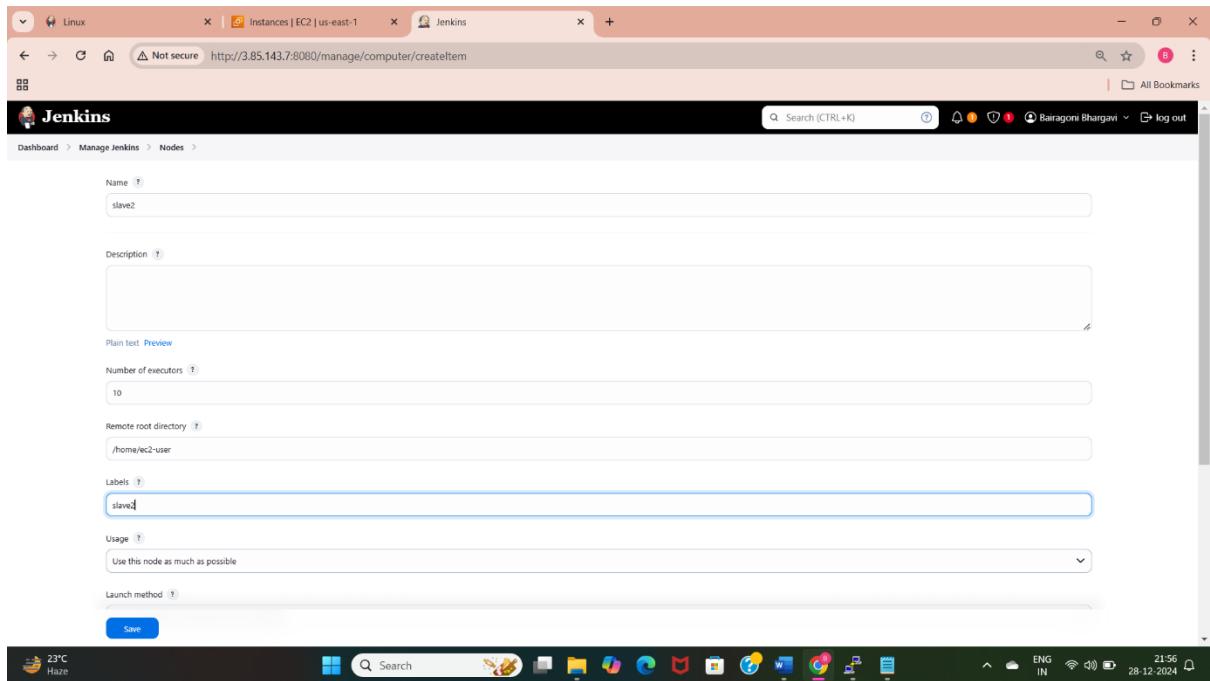
```

- ✓ Now we can see the build is started by the user Bairagoni Bhargavi. Building on the workspace of the Jenkins by moving into the job and running the above mentioned commands.

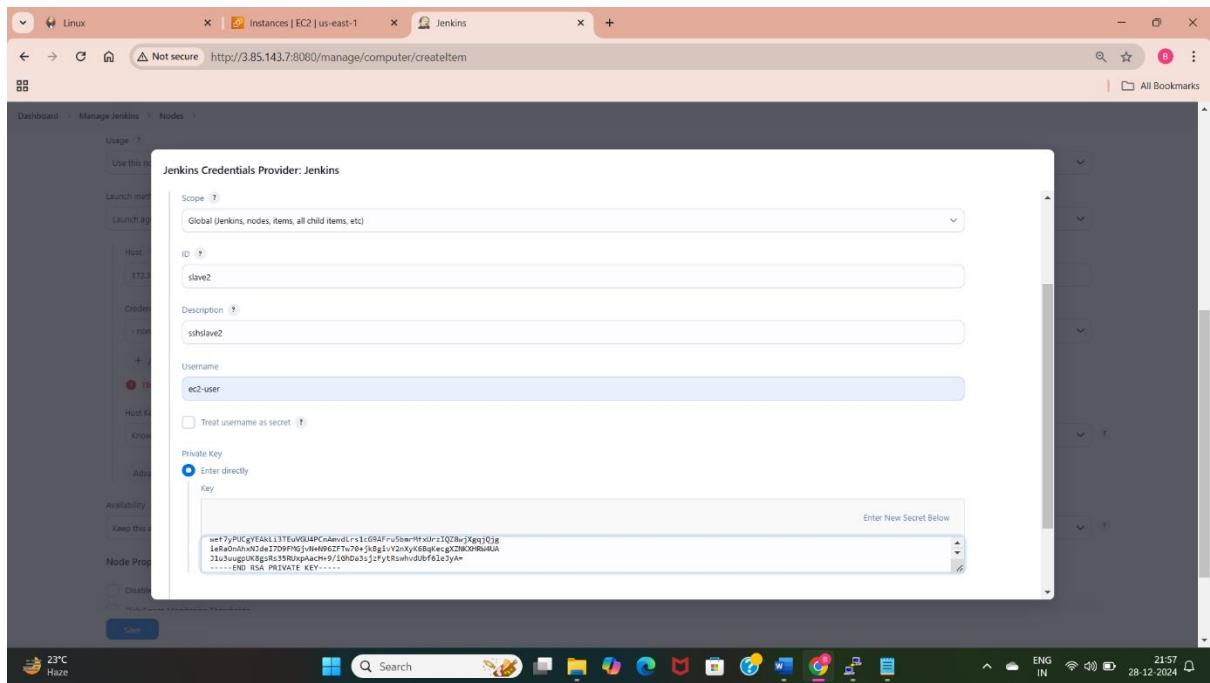


- ✓ After build is success we use the public IP address of instance in the browser so we can able to access the nginx webpage.

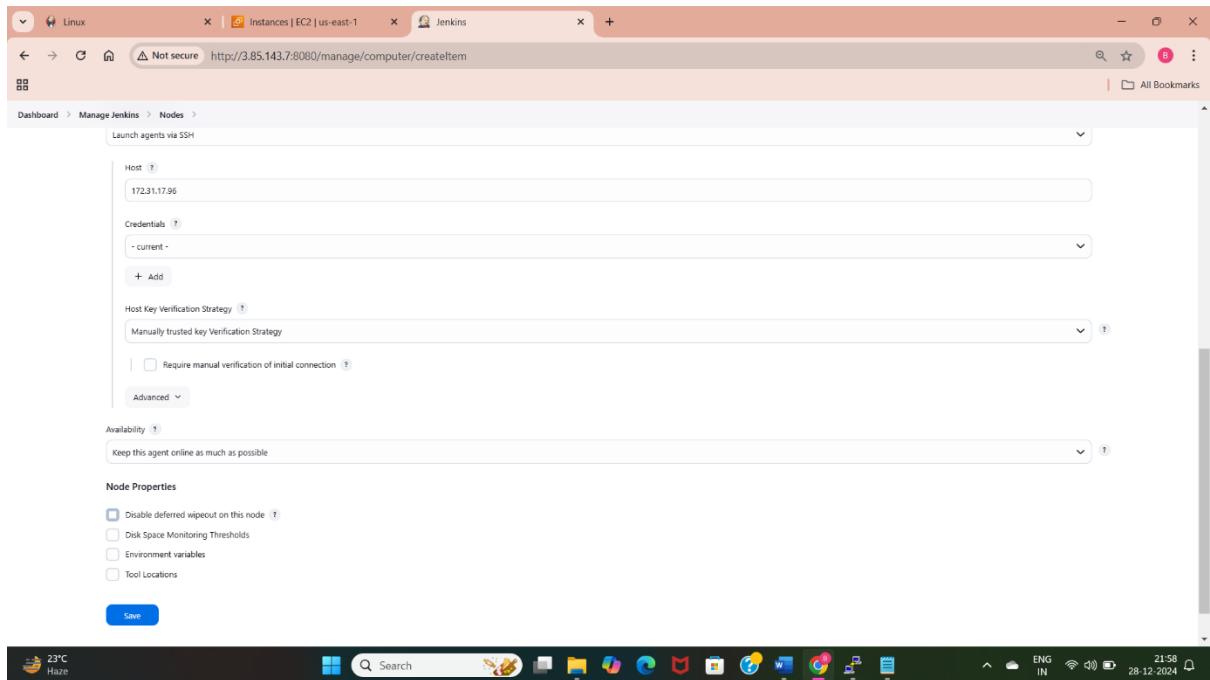
## Slave-2:



- ✓ Now am setting the master slave configurations between master and slave2. Here I have provided the name, selected 10 executors, root directory of the server, give any label name, select usage.



- ✓ Now provide the credentials to use the slave2 by using the SSH Username with Private key and provide private key there.



- ✓ Give the private IP address of the instance and add the created credentials, select host key verification strategy as “Require trusted key Verification Strategy”, select keep this agent online as much as possible.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	Built-In Node	Linux (amd64)	In sync	5.52 GiB	0 B	5.52 GiB	0ms
2	slave1	Linux (amd64)	In sync	5.88 GiB	0 B	5.88 GiB	34ms
3	slave2	Linux (amd64)	In sync	5.90 GiB	0 B	5.90 GiB	66ms
	Data obtained	9 min 28 sec	9 min 28 sec	9 min 28 sec	9 min 28 sec	9 min 28 sec	9 min 28 sec

The screenshot shows the Jenkins 'Nodes' management page. It lists the following nodes:

- Built-In Node
- slave1
- slave2
- Data obtained

The slave2 node has 10 executors. The table provides details for each node, including architecture, clock difference, free disk space, free swap space, free temp space, and response time.

- ✓ We can see the slave2 node is created successfully with 10 executors.

```

ec2-user@ip-172-31-17-96:~$ Verifying : libXt-1.1.5-3.amzn2.0.2.x86_64
Verifying : giflib-4.1.6-9.amzn2.0.2.x86_64
Verifying : libXinerama-1.1.3-2.1.amzn2.0.2.x86_64
Verifying : libavu-sans-mono-fonts-2.33-6.amzn2.noarch
Verifying : log4j-cve-2021-44228-hotpatch-1.3-7.amzn2.noarch
Verifying : libX11-common-2.3.0-13+11-1.amzn2.1.x86_64
Verifying : libXext-1.3.0-6.amzn2.x86_64
Verifying : libXtst-1.2.3-1.amzn2.0.2.x86_64
Verifying : python-javapackages-3.4.1-11.amzn2.noarch
Verifying : libICE-1.0.9-9.amzn2.0.2.x86_64
Verifying : libXtst-tools-3.4.1-11.amzn2.noarch
Installed:
java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1

Dependency Installed:
alsa-lib.x86_64 0:1.4.1-2.amzn2
dejavu-sans-mono-fonts.noarch 0:2.33-6.amzn2
fontpackages-fsfsystem.noarch 0:1.44-8.amzn2
javapackages-tools.noarch 0:3.4.1-11.amzn2
libXiL.x86_64 0:1.6.7-3.amzn2.0.5
libXext.x86_64 0:1.3.3-3.amzn2.0.5
libXrandr.x86_64 0:1.5.1-2.amzn2.0.3
libXtst.x86_64 0:1.2.3-1.amzn2.0.2
log4j-cve-2021-44228-hotpatch.noarch 0:1.3-7.amzn2

Complete!
[ec2-user@ip-172-31-17-96 ~]$ sudo hostname slave2
[ec2-user@ip-172-31-17-96 ~]$ exec bash
[ec2-user@slave2 ~]$ cd .ssh/
[ec2-user@slave2 .ssh]$ ll
total 4
-rw-r--r-- 1 ec2-user ec2-user 389 Dec 28 16:23 authorized_keys
[ec2-user@slave2 .ssh]$ sudo vi authorized_keys
[ec2-user@slave2 .ssh]$ sudo visudo
[ec2-user@slave2 .ssh]$ cd
[ec2-user@slave2 ~]$ history
1 sudo yum install java-17 -y
2 sudo hostname slave2
3 exec bash
4 cd .ssh/
5 ll
6 sudo vi authorized_keys
7 sudo visudo
8 cd
9 history
[ec2-user@slave2 ~]$ 

```

23°C Haze ENG IN 22:03 28-12-2024

- ✓ Now am installing the java dependency in slave2 server.

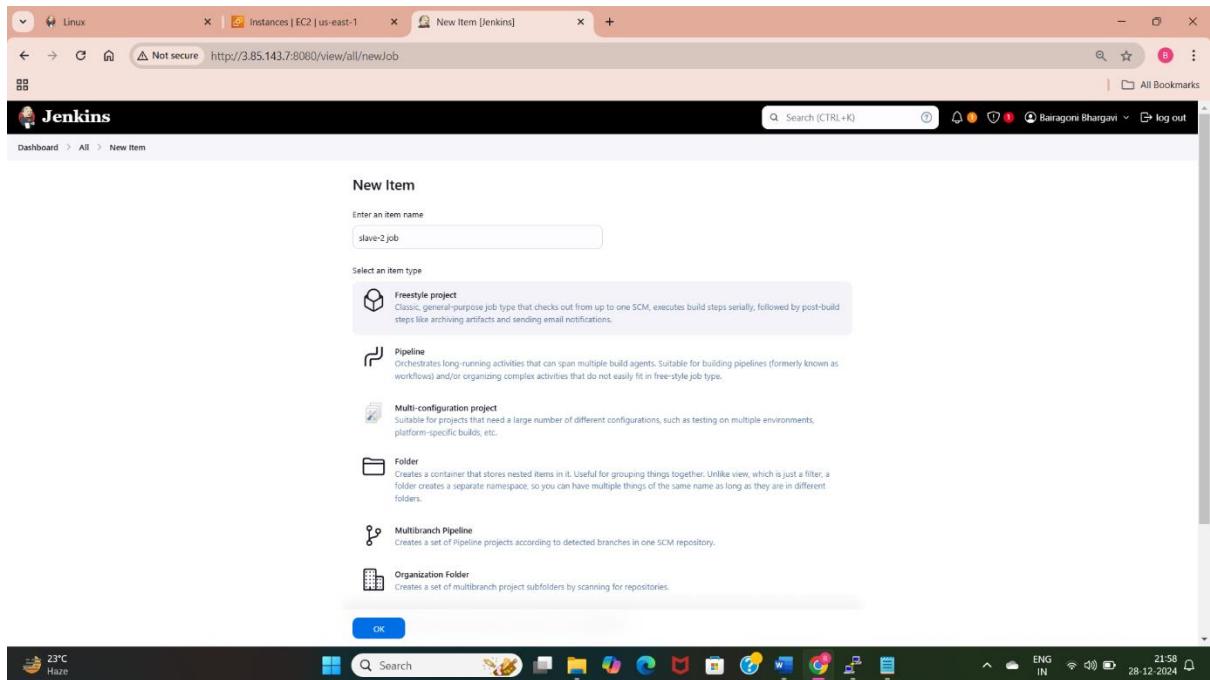
```

ec2-user@ip-172-31-17-86:~$ ssh-copy-id -i /root/.ssh/id_rsa.pub ec2-user@ip-172-31-19-44
[ec2-user@ip-172-31-17-86:~]$ 

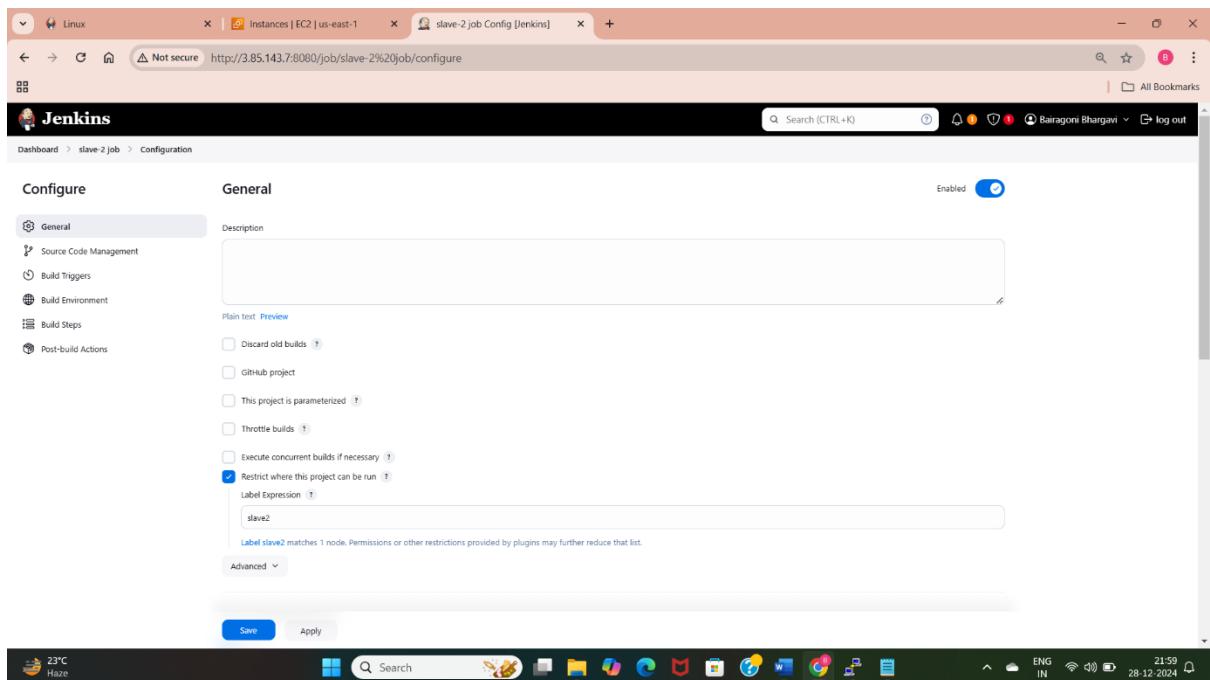
```

23°C Haze ENG IN 22:07 28-12-2024

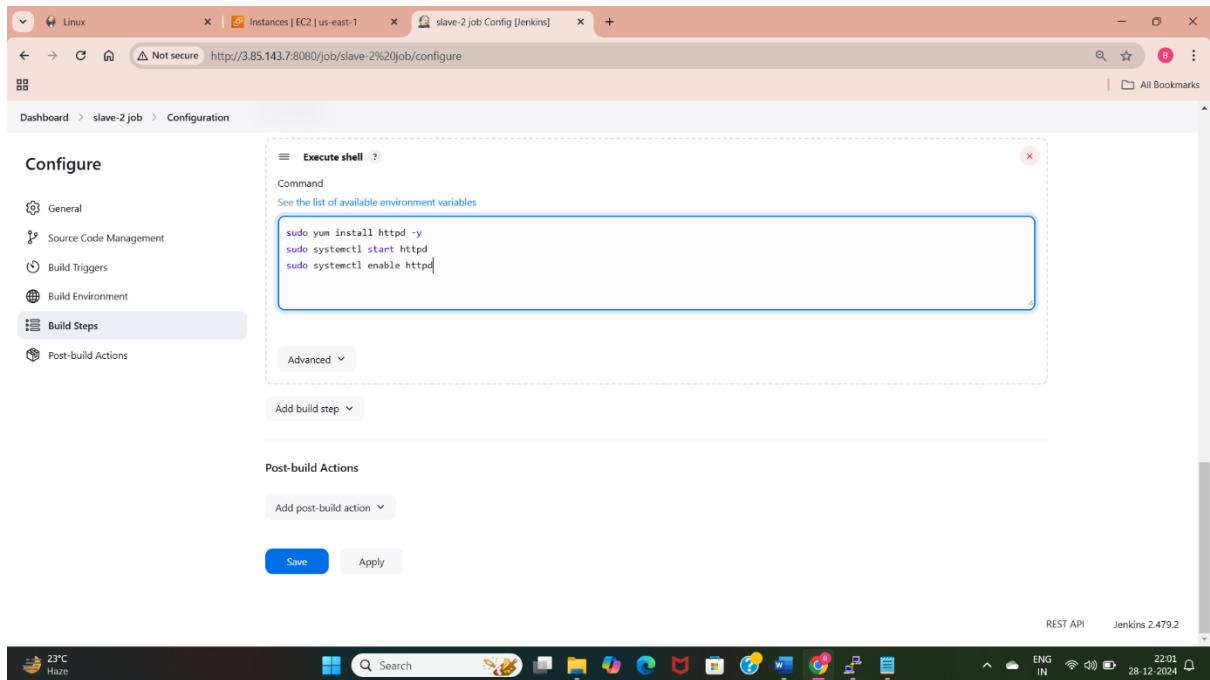
- ✓ Now am setting connection between master and slave by copying the public key of master into the authorized keys of slave2 server.



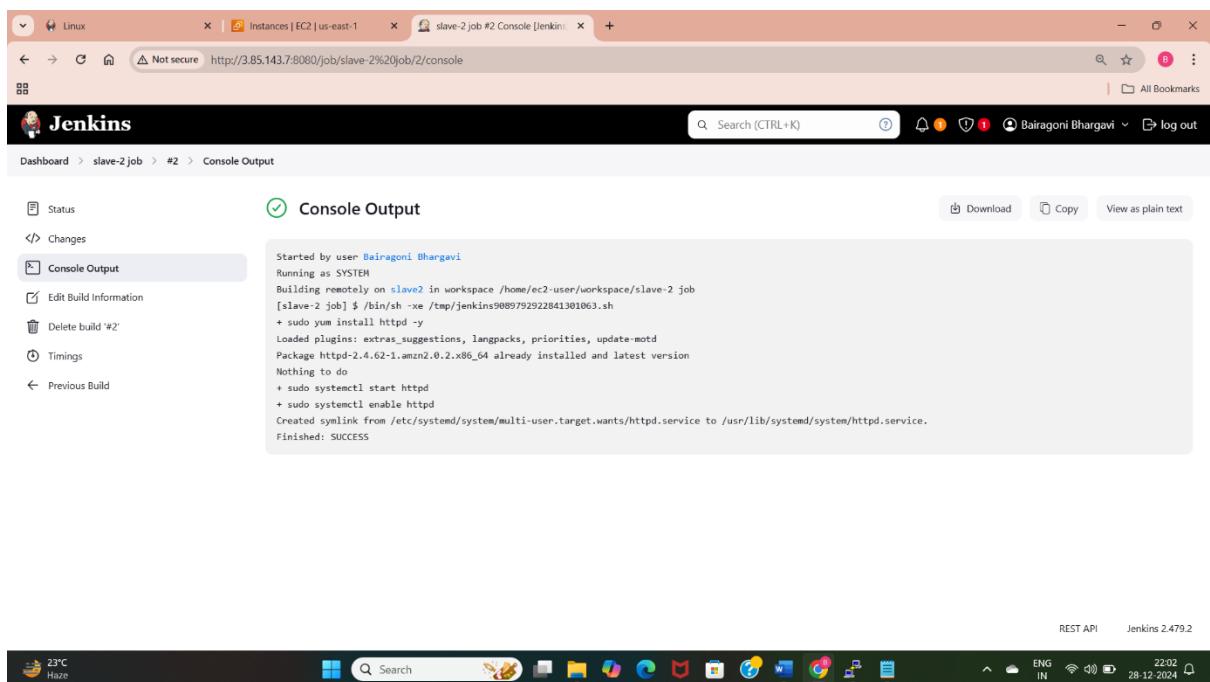
- ✓ Now am creating another job for slave2 using freestyle project and select ok.



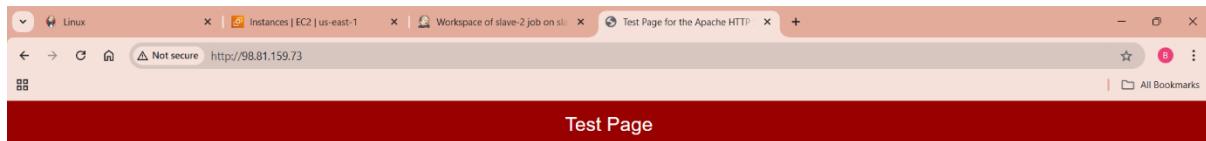
- ✓ Now select the where this job need to build. I am restricting the build by selecting slave2 label. Now whatever builds are executed are executed in this slave2.



- ✓ Now select the Build steps and select execute shell. Now give commands to install httpd using <yum install httpd -y>, start httpd using <systemctl start httpd> and save and build the job.



- ✓ Now we can see the build is success. We can also see the build is started by the user Bhargavi and executed the commands in Jenkins workspace where the job is stored.



This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

**If you are a member of the general public:**

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting [www.example.com](http://www.example.com), you should send e-mail to "webmaster@example.com".

**If you are the website administrator:**

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:



- ✓ Now am able to access the httpd webpage using the public IP address of the instance.

## Slave-3:

### Sonarqube:

**SonarQube** is an open-source platform used for continuous inspection of code quality. It helps developers and teams to track, maintain, and improve the quality of their code over time by identifying potential issues, bugs, vulnerabilities, and code smells.

Here are the key features of SonarQube:

#### Code Quality Analysis

SonarQube performs an automated review of your code to check for quality issues. It analyzes various aspects of your code, such as:

- Bugs: Defects that might cause errors or unexpected behavior.
- Vulnerabilities: Security issues that can lead to potential exploits.
- Code Smells: Areas where the code might be hard to maintain or understand.
- Duplications: Repeated code blocks that can be refactored.
- Test Coverage: Measures the percentage of your code covered by tests.
- Complexity: Evaluates the complexity of the code to help in maintaining it easily.
- SonarQube can be integrated with popular CI tools like Jenkins, GitLab CI, Azure DevOps, etc. After every build, it can analyze your code to catch issues early in the development process.

```

3.87.60.69 - PuTTY
Verifying : libX11-1.7.9-1.amzn2.0.2.x86_64
Verifying : libavu-fonts-common-2.33-6.amzn2.noarch
Verifying : libXau-1.0.8-2.1.amzn2.0.2.x86_64
Verifying : libSM-1.2.2-2.amzn2.0.2.x86_64
Verifying : libXcursor-1.4.0-1.amzn2.0.2.x86_64
Verifying : libX11-common-1.6.7-3.amzn2.0.5.noarch
Verifying : fontconfig-2.13.0-4.3.amzn2.x86_64
Verifying : dejavu-sans-fonts-2.33-6.amzn2.noarch
Verifying : libxt-1.1.5-3.amzn2.0.2.x86_64
Verifying : giflib-4.1.6-9.amzn2.0.2.x86_64
Verifying : libXinerama-1.1.3-2.1.amzn2.0.2.x86_64
Verifying : libjavu-sans-mono-fonts-2.33-6.amzn2.noarch
Verifying : log4j-cve-2021-44228-hotpatch-1.3-7.amzn2.noarch
Verifying : libavu-1.0.8-2.1.amzn-corretto-1.7.0.13+11-1.amzn2.1.x86_64
Verifying : libX11-1.7.9-1.amzn2.0.2.x86_64
Verifying : python-lxml-3.2.1-4.amzn2.0.6.x86_64
Verifying : python-javapackages-3.4.1-11.amzn2.noarch
Verifying : libICE-1.0.9-9.amzn2.0.2.x86_64
Verifying : javapackages-tools-3.4.1-11.amzn2.noarch
Installed:
java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1

Dependency Installed:
alsa-lib.x86_64 0:1.1.4.1-2.amzn2
dejavu-sans-mono-fonts.noarch 0:2.33-6.amzn2
fontpackages-fs.noarch 0:1.44-8.amzn2
javapackages-tools.noarch 0:3.4.1-11.amzn2
libX11.x86_64 0:1.6.7-3.amzn2.0.5
libXext.x86_64 0:1.3.3-3.amzn2.0.2
libXrandr.x86_64 0:1.5.1-2.amzn2.0.3
libXslt.x86_64 0:1.1.2.1-1.amzn2.0.2
log4j-cve-2021-44228-hotpatch.noarch 0:1.3-7.amzn2

Complete!
[ec2-user@ip-172-31-82-173 ~]$ cd .ssh/
[ec2-user@ip-172-31-82-173 .ssh]$ ll
total 4
-rw-r----- 1 ec2-user ec2-user 389 Dec 29 10:44 authorized_keys
[ec2-user@ip-172-31-82-173 .ssh]$ sudo vi authorized_keys
[ec2-user@ip-172-31-82-173 .ssh]$ sudo hostname slave3
[ec2-user@ip-172-31-82-173 .ssh]$ exec bash
[bash]$ exec slave3; not found
[ec2-user@ip-172-31-82-173 .ssh]$ exec bash
[ec2-user@slave3 .ssh]$
```

- Now connect the third server with terminal and install the runtime of Jenkins that is java 17. Copy the public key of master and paste it in the slave3 server authorized keys.

New node

Node name

Type

Permanent Agent

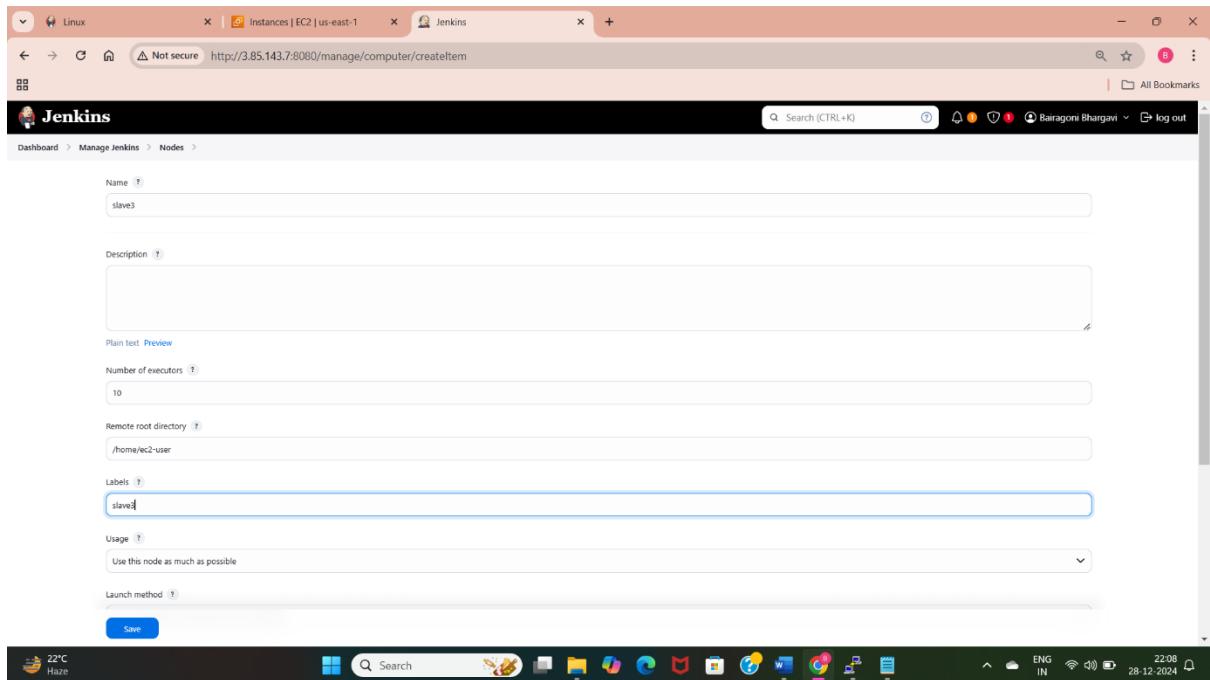
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Copy Existing Node

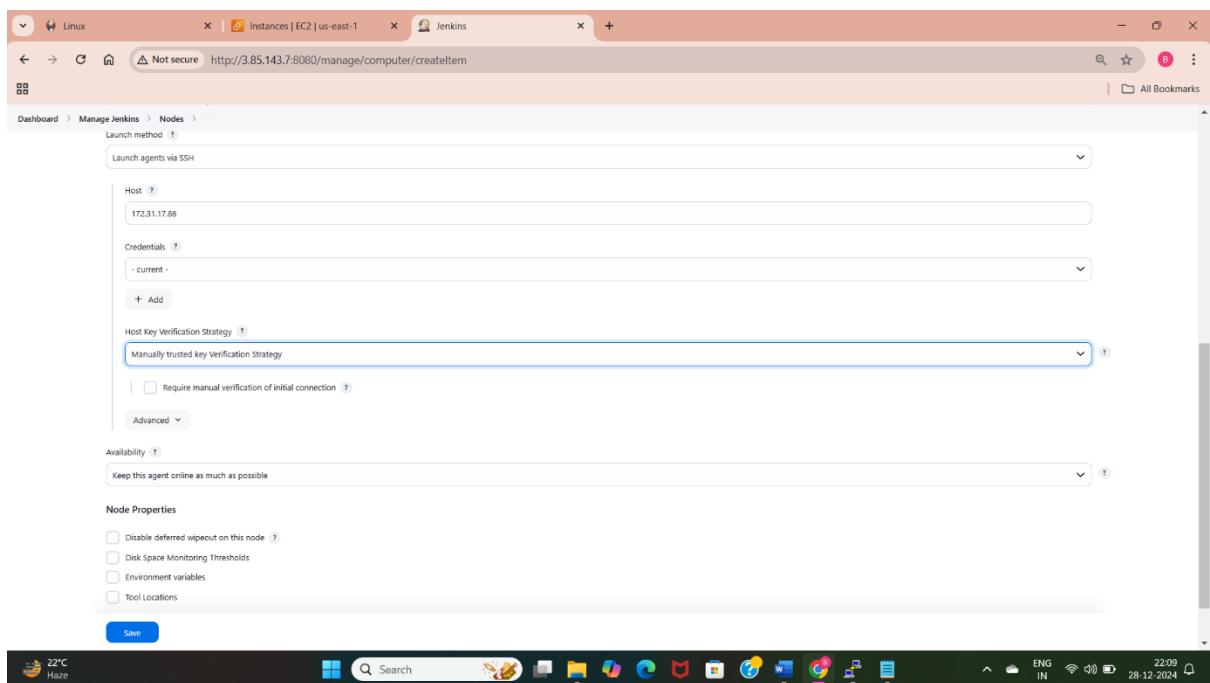
Create

REST API Jenkins 2.479.2

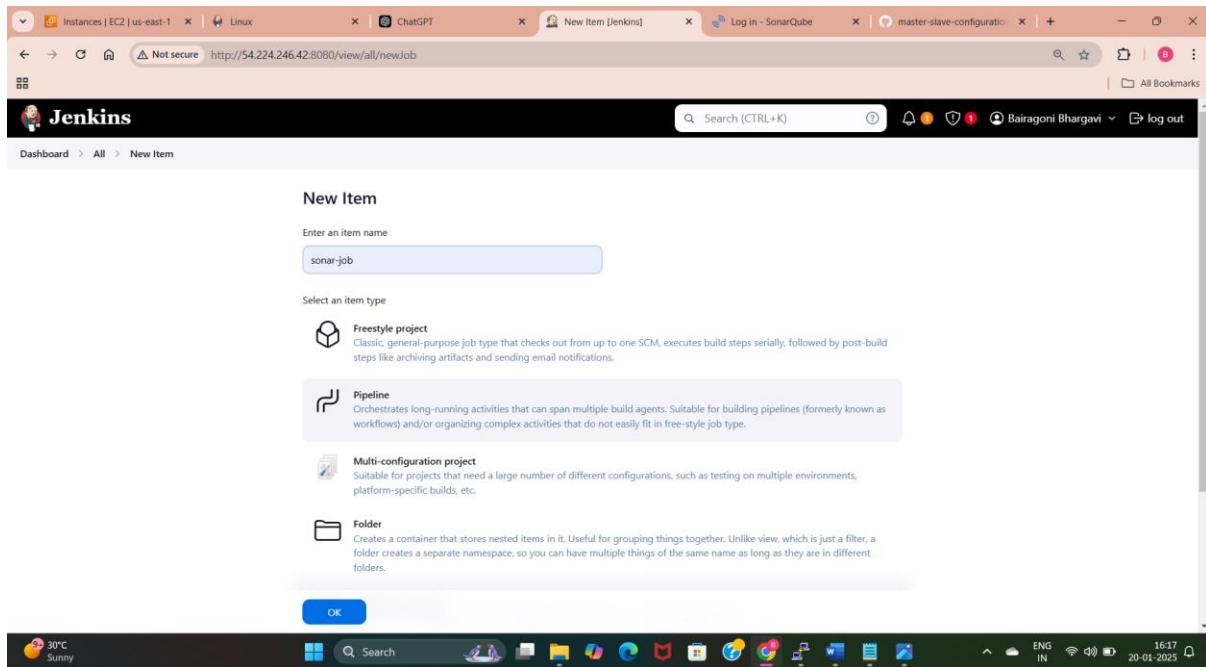
- Now create a slave3 node, by moving into Manage Jenkins> Nodes> new node name, and select permanent agent.



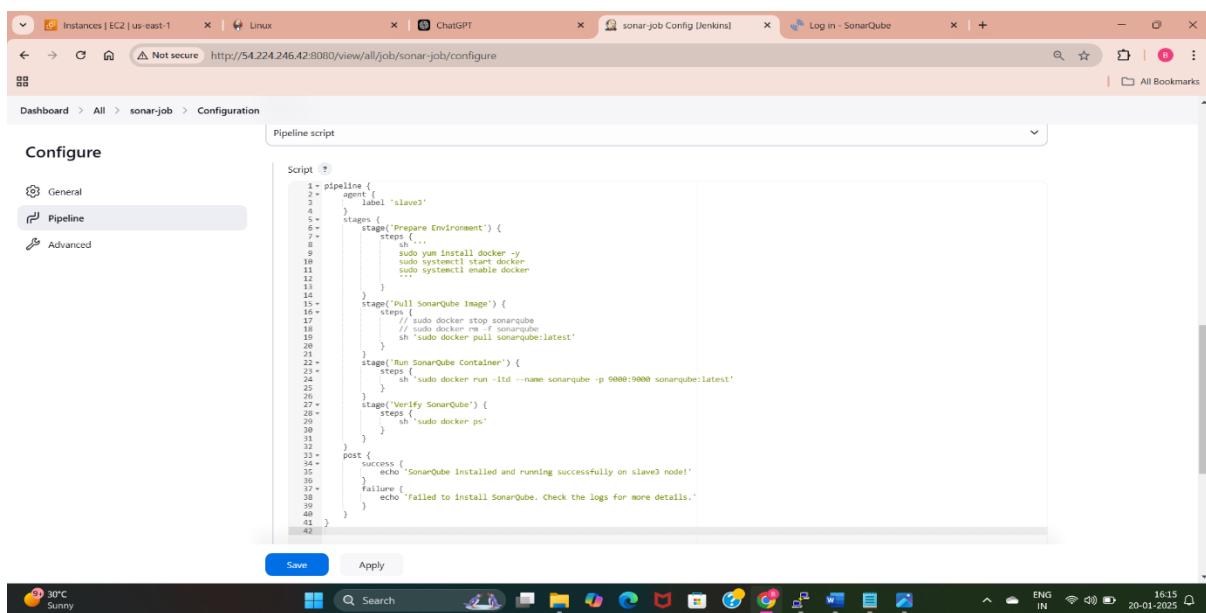
- Now give the name as slave3, select the number of executors, and provide root directory of slaver server, select use the node as much as possible as usage.



- In launch method select “launch agent via SSH” and provide private Ip address of slave4 instance and add credentials by using add option. Manually trusted key verification strategy is used as we are launching agent via SSH.



- Now create a new pipeline job for slave3 to install sonarqube and start using sonarqube webpage.

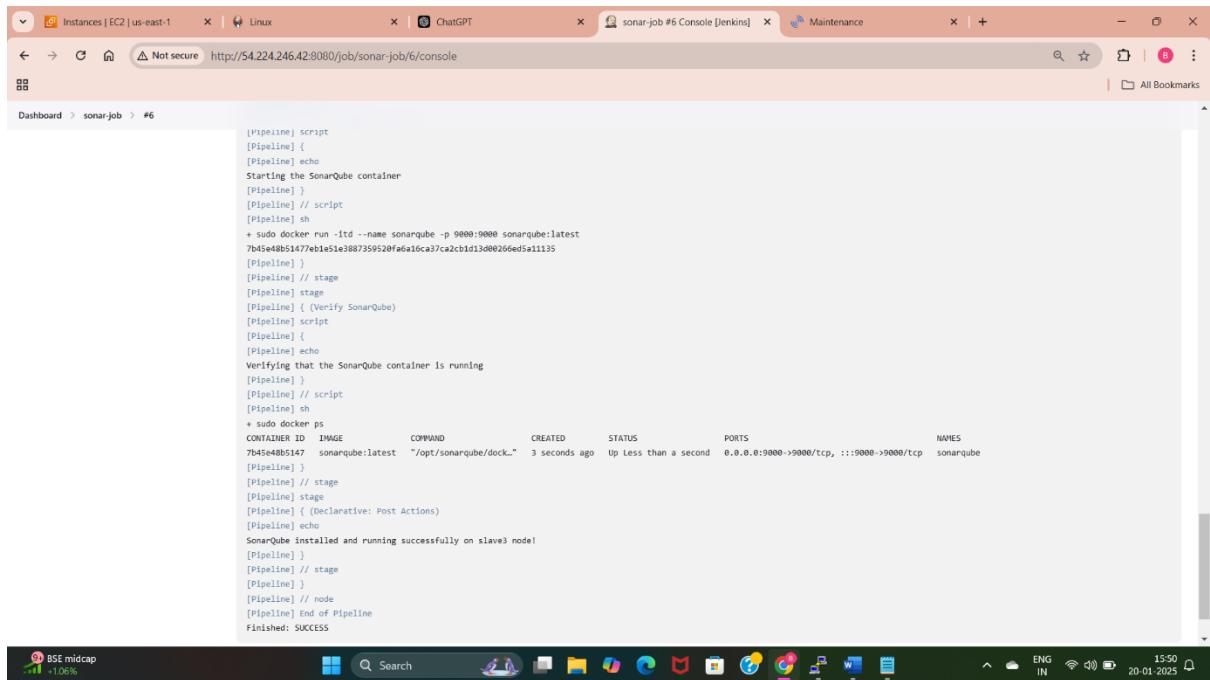


- Here am selecting the slave3 label which is created while creating slave3 node under agent in the pipeline. I have written the pipeline to install docker and pulled sonarqube image from docker.
- Now select build steps, in build steps select execute shell. In execute shell am providing commands to download the sonarqube.
- First am taking the sonarqube link from browser and downloading it using wget, downloaded as zip file.
- Now unzip the zip file of sonarqube.
- Am changing the downloaded and unzipped file name to sonar using mv command.
- Move into sonar file inside this we are having bin. Get inside the bin and there we are having linux-x86-64 file go inside this file and start sonar using “sh sonar.sh start”.

The screenshot shows the Jenkins Pipeline Stage View for the 'sonar-job' pipeline. On the left, there's a sidebar with various options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Stages, Rename, and Pipeline Syntax. Below that is a 'Builds' section showing a single build from today at 9:49 AM. The main area is titled 'Stage View' and displays four stages: 'Setup Docker Environment', 'Pull SonarQube Image', 'Run SonarQube Container', and 'Verify SonarQube'. Each stage has a bar indicating its average duration: 7s, 10s, 900ms, and 250ms respectively. Below each bar is a table showing the execution time for two specific runs on Jan 20 at 15:16. The first run failed, while the second was successful. The entire pipeline took approximately 38 seconds.

The screenshot shows the Jenkins Pipeline Console Output for build #6. The sidebar on the left includes options like Status, Changes, Console Output (which is selected), Edit Build Information, Delete build #6, Timings, Pipeline Overview, Pipeline Console, Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main content area is titled 'Console Output' and shows the command-line logs for the pipeline. It starts by running a script on a slave node, then installs Docker using sudo yum install docker -y. It lists loaded plugins and resolves dependencies. A note indicates that the package docker-x86\_64 0:25.0.0-1.amzn2.0.3 will be installed. The process continues with the installation of containerd, liblbergroup, runc, and libselinux, followed by the installation of the docker package itself. The log concludes with a note about dependency resolution.

- Now after writing commands in execute shell save and click on “build now” option. Then the build will start and executes the commands. We can see the build is success.

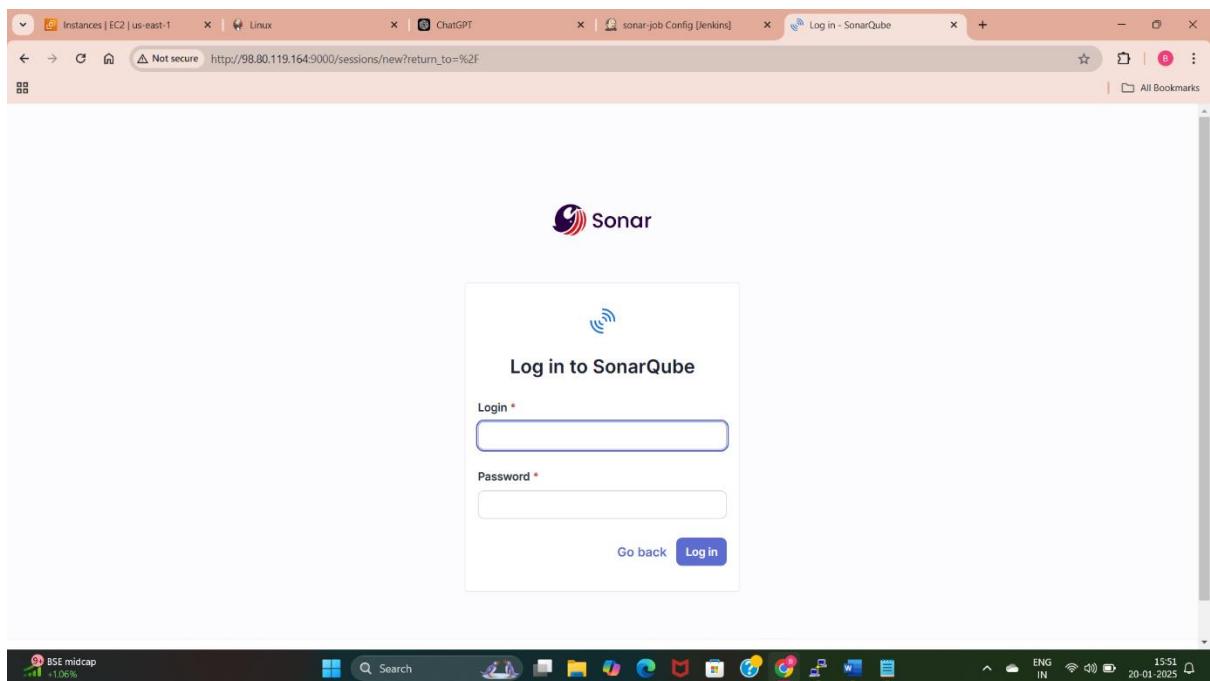


```

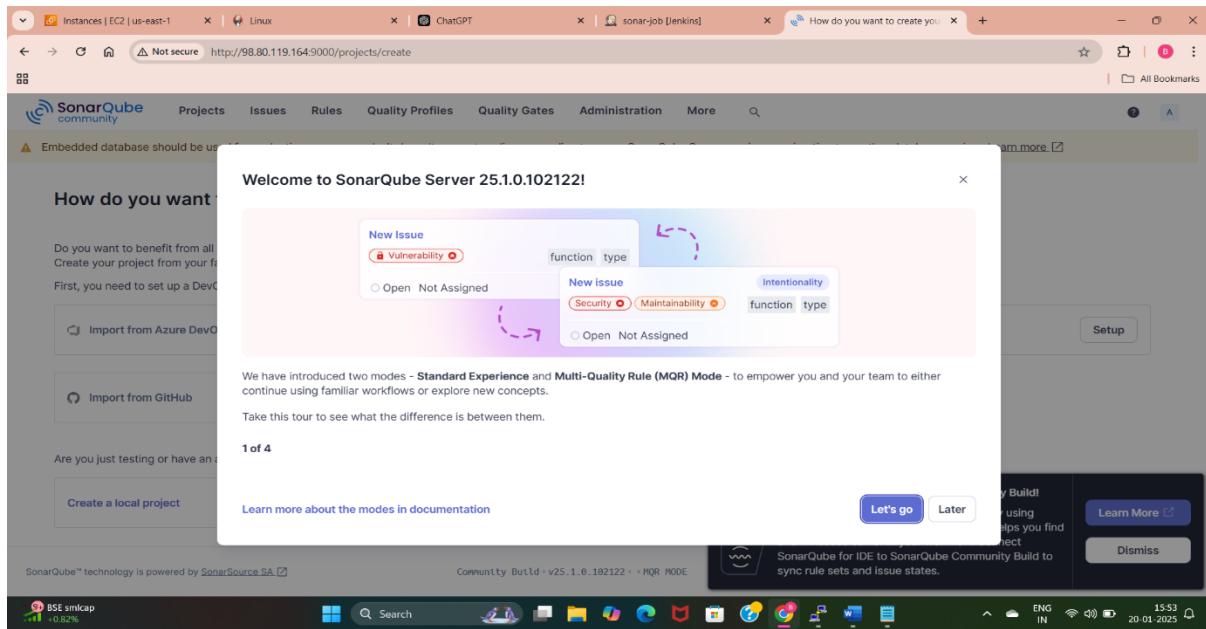
[Pipeline] script
[Pipeline] {
[Pipeline] echo
Starting the SonarQube container
[Pipeline]
[Pipeline] // script
[Pipeline] sh
+ sudo docker run -itd --name sonarqube -p 9000:9000 sonarqube:latest
7b45e48b51477eb1e51e3887359520f6a16ce37ca2cb1d13d00266ed5a11135
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Verify SonarQube)
[Pipeline] script
[Pipeline] {
[Pipeline] echo
Verifying that the SonarQube container is running
[Pipeline]
[Pipeline] // script
[Pipeline] sh
+ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
7b45e48b5147        sonarqube:latest   "/opt/sonarqube/dock..."   3 seconds ago     Up Less than a second   0.0.0.0:9000->9000/tcp, 0.0.0.0:9001->9001/tcp   sonarqube
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
SonarQube installed and running successfully on slave3 node!
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] end of Pipeline
Finished: SUCCESS

```

- Now when build is success we can observe that the sonarqube is installed and started.
- After building the job we can check whether the sonarqube files are downloaded or not by moving into the Jenkins workspace and check with the process.



- After build is success copy the public ip of instance and paste it in the browser with sonarqube port number 9000 and check whether we can able to access the sonar page or not. Here am able to access the



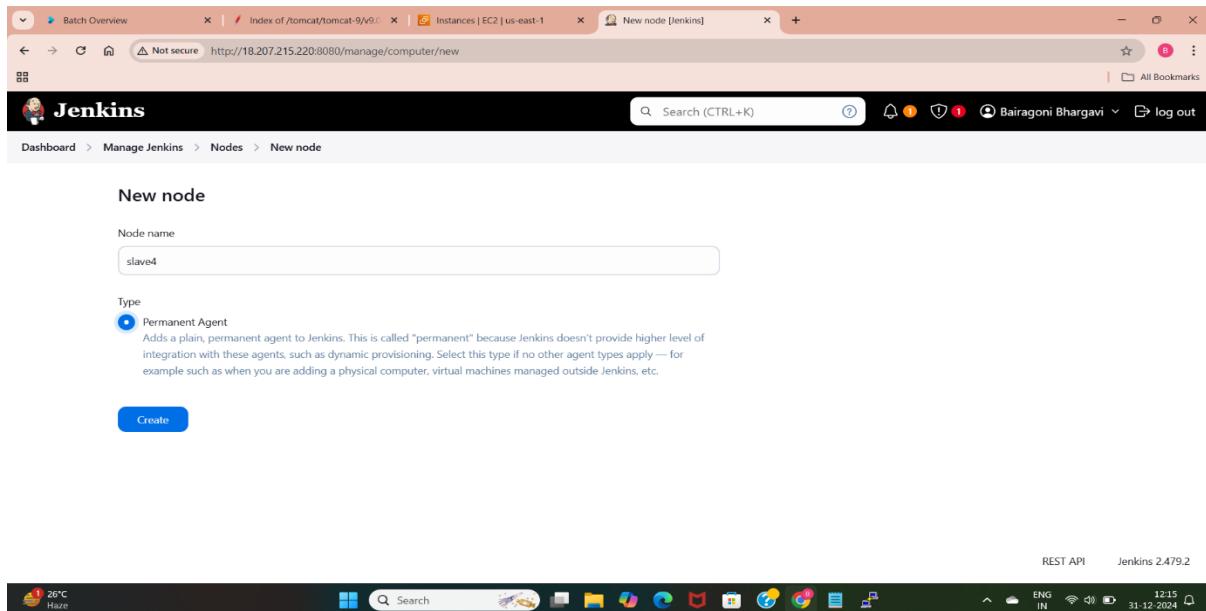
- This is the sonar page which is deployed using the Jenkins execute shell.

## Slave-4:

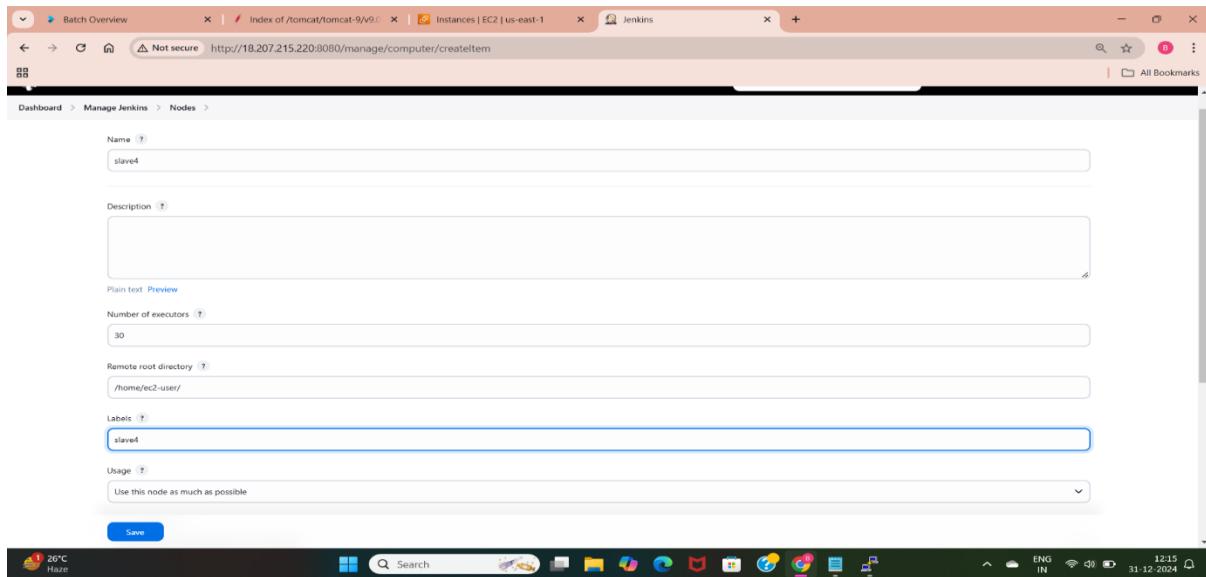
### Tomcat:

Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation. It is primarily used to serve Java-based web applications and is often associated with running Java Servlets and JavaServer Pages (JSP).

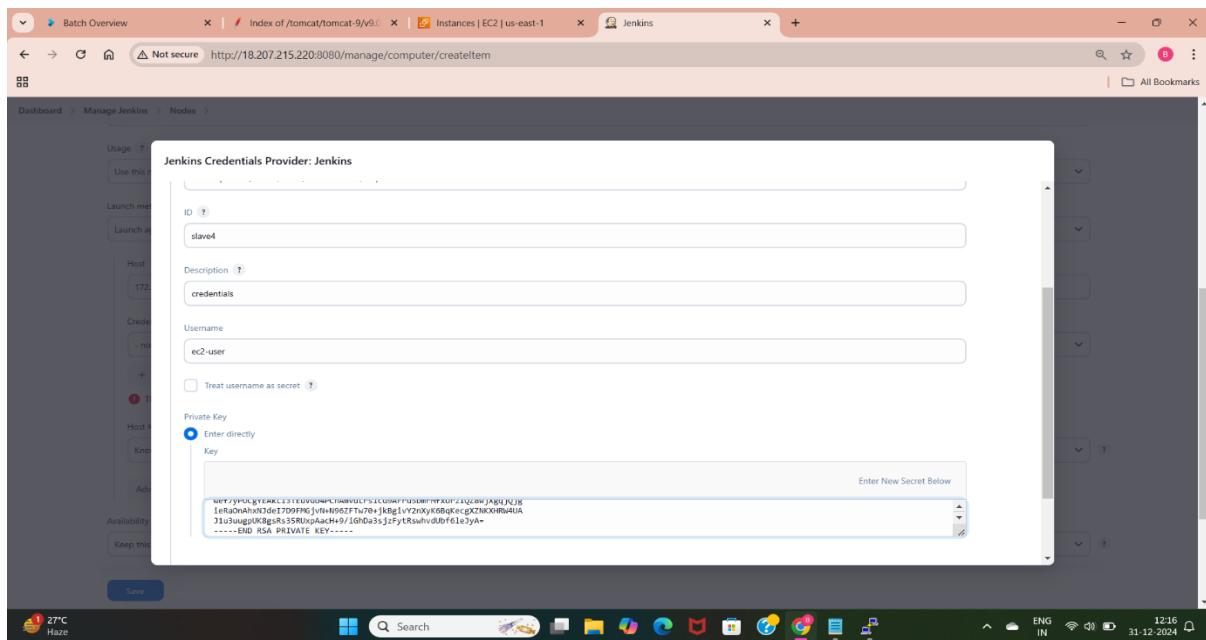
Go to Manage Jenkins > Manage Nodes and Clouds > New Node>Create Slave Node.



- ♣ Enter a name for the slave node and select Permanent Agent.
- ♣ Click OK to configure the slave node.

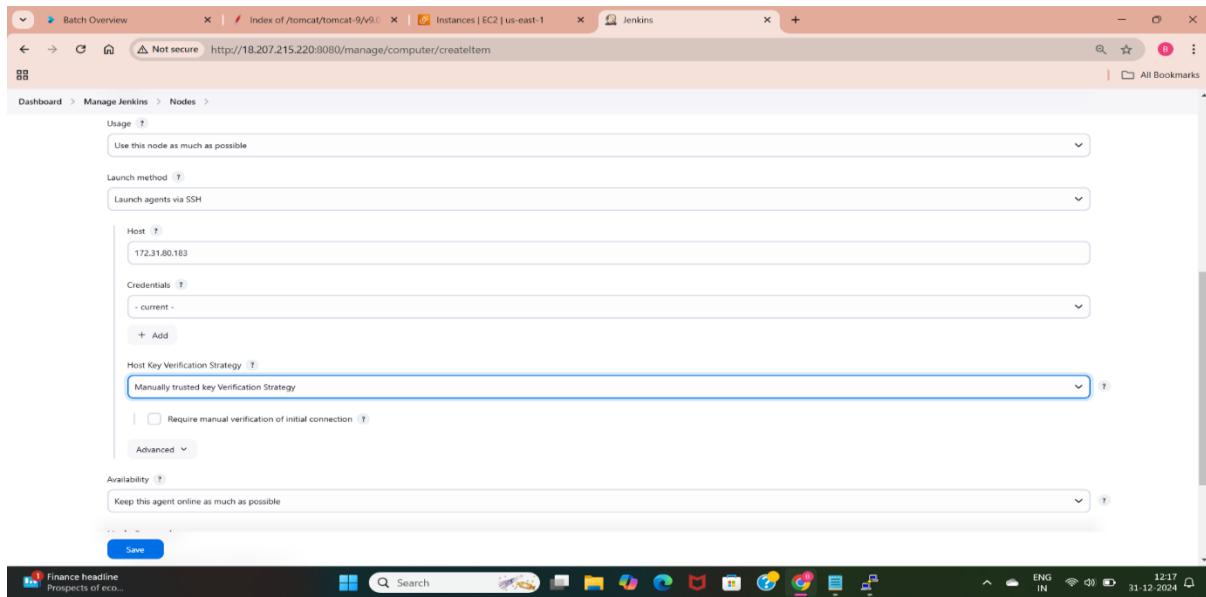


- ♣ Configure Node Settings:
- ♣ Number of Executors: Specify the number of builds the agent can execute simultaneously (e.g., 10).
- ♣ Remote Root Directory: Provide the directory where Jenkins will store job files on the slave (e.g., /home/ubuntu or /home/ec2-user).
- ♣ Labels: Assign a label for the slave (e.g., slave1, test-agent). This will help identify the node for specific jobs.
- ♣ Usage: Choose "Use this node as much as possible".



#### Set Launch Method:

- ♣ Select "Launch agents via SSH".
- ♣ Now provide the credentials to use the slave4 by using the SSH Username (ec2-user if you use amazon linux) with Private key and provide created master private key there.



- Now select the launch method, provide private Ip address of slave4 server, add credentials using above created credentials, verify credentials by selecting the manually trusted key verification strategy.

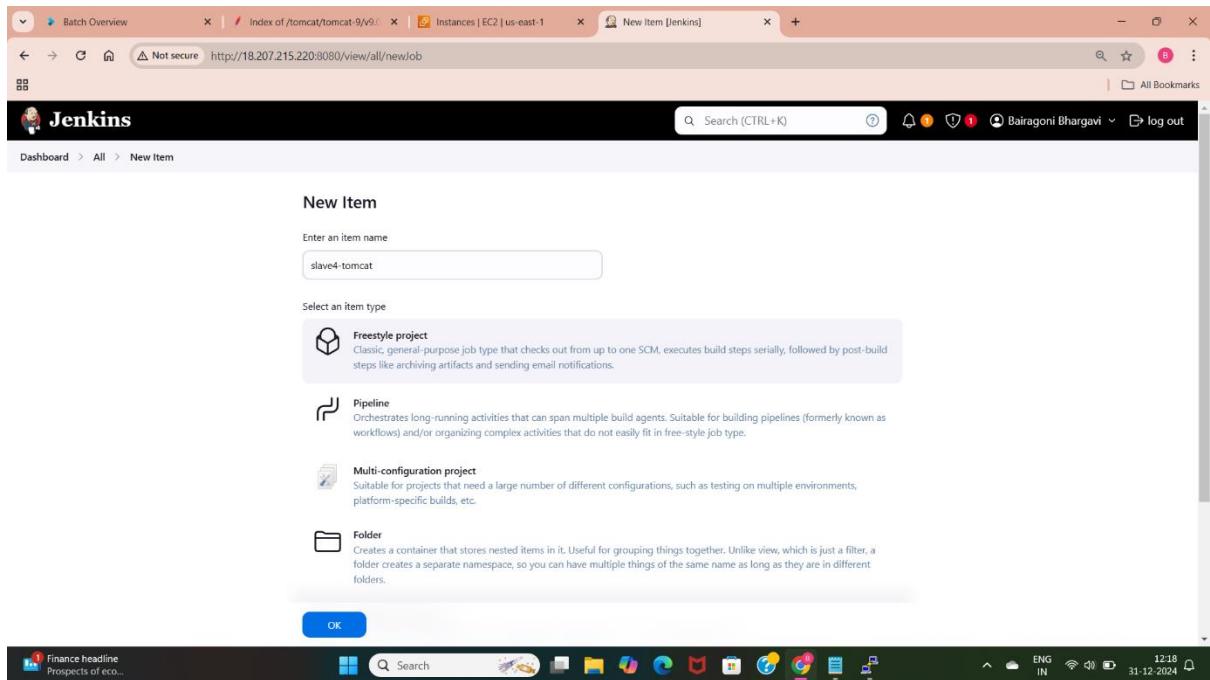
Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.70 GiB	0 B	5.70 GiB	0ms
	slave4		N/A	N/A	N/A	N/A	N/A
	Data obtained	22 ms	15 ms	7 ms	6 ms	2 ms	1 ms

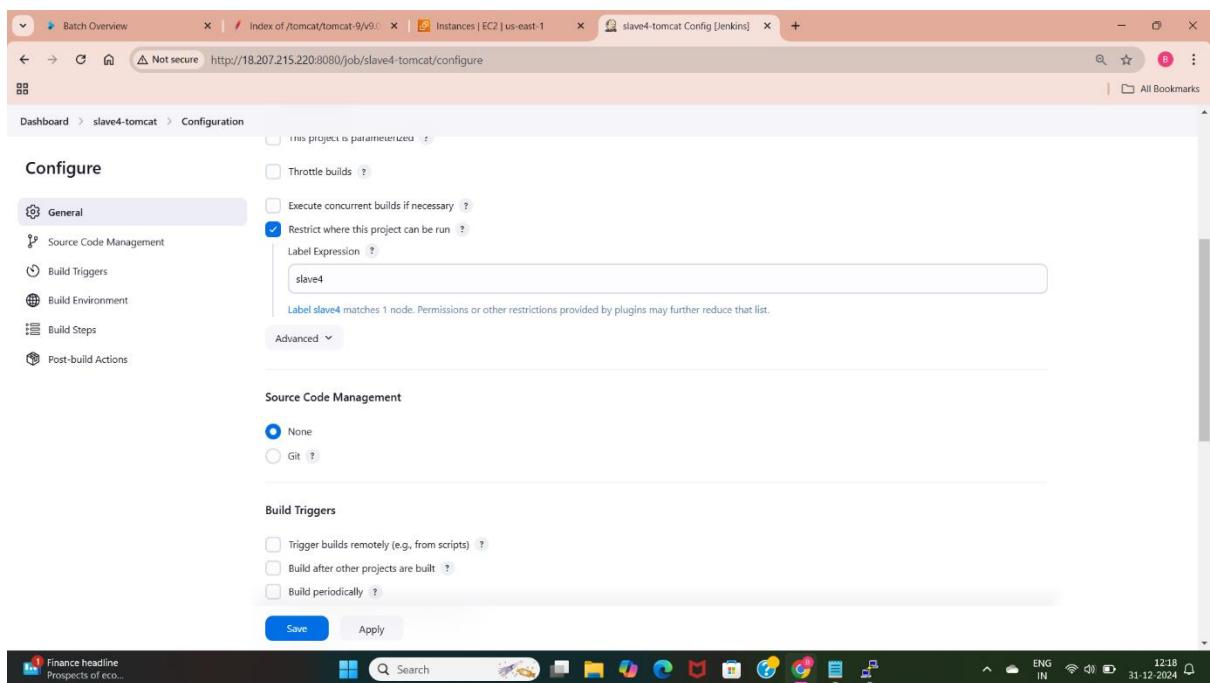
- Now we can see a new node is created with the name slave4. and Jenkins successfully connected with slave4.

### Running Jobs on Slave Nodes:

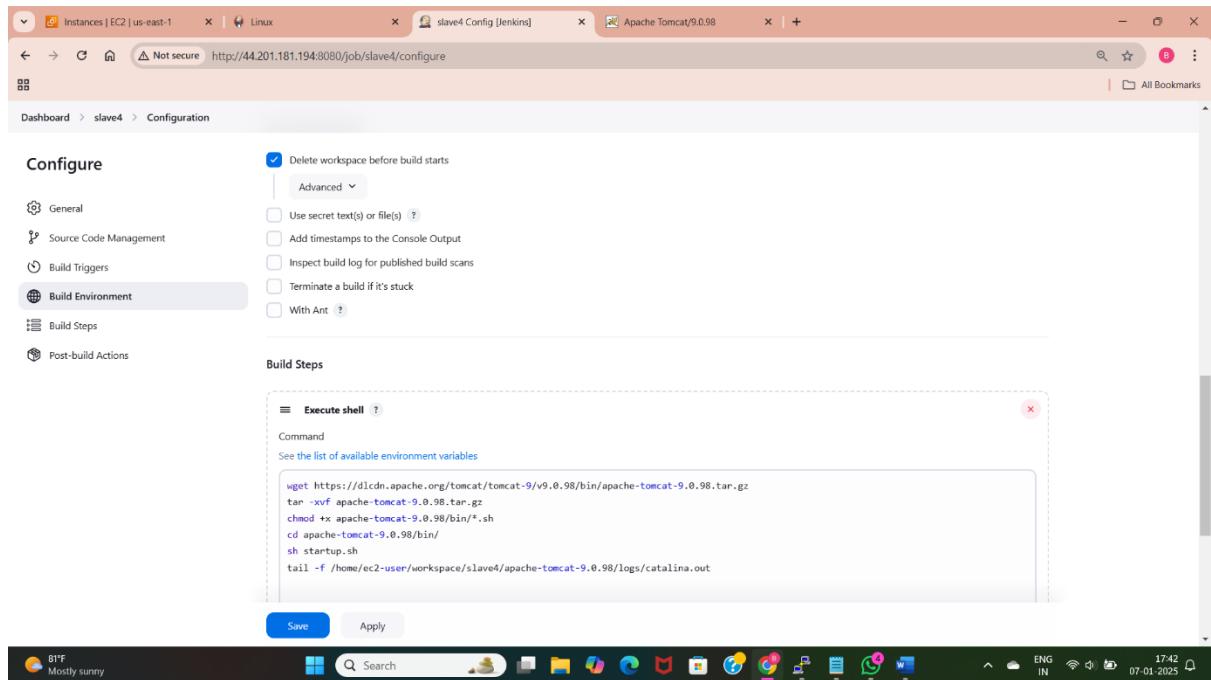
Go to Jenkins Dashboard and click New Item. Provide a name for the job and choose the desired job type (e.g., Freestyle Project).



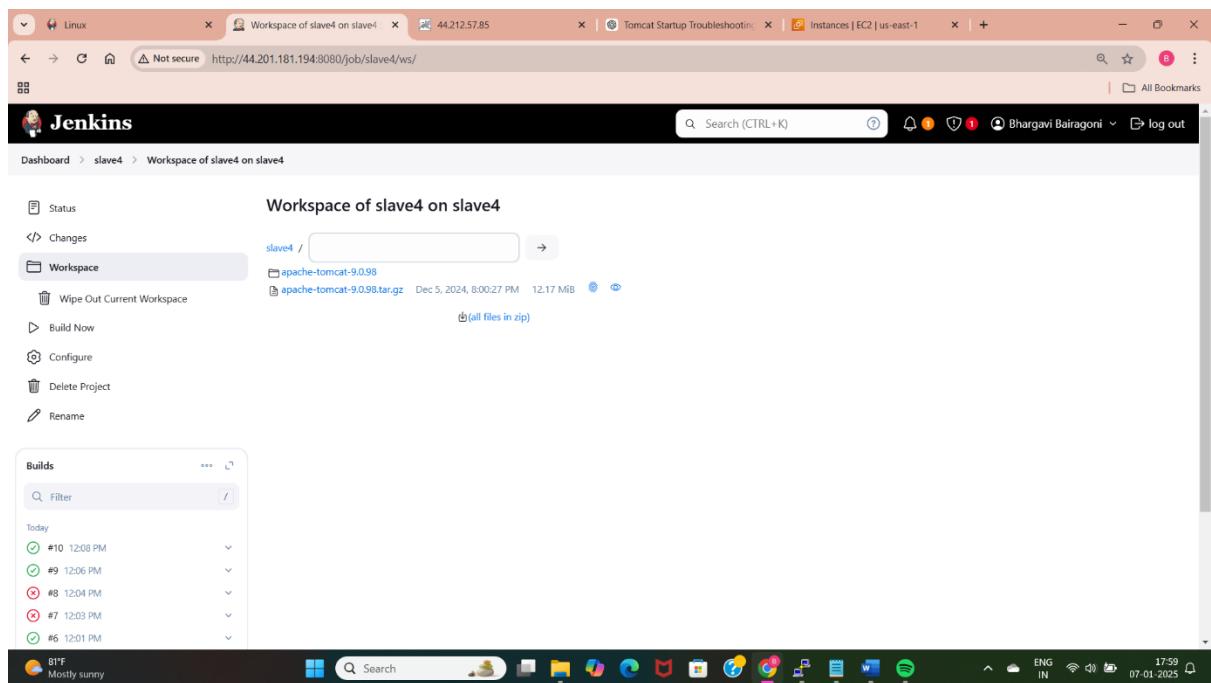
- Now am creating a new job by selecting new item at dashboard, give the name as slave4-tomcat for the new job.



- Under the General section, enable "Restrict where this project can be run".
- Enter the label of the slave node where you want this job to execute.. Here am selecting the slave4 label to execute the builds.



- Now in build steps select execute shell. Give the commands to download the tomcat using wget command. The file is downloaded as tar file, now extract the file using “tar -xvf filename”. Next move the path where tomcat is downloaded and move to bin folder inside the tomcat and start the tomcat.



- Now save and go to slave4 job and click on “build now”. Here we can see the build is success and the tomcat file is downloaded and extracted the file.

```

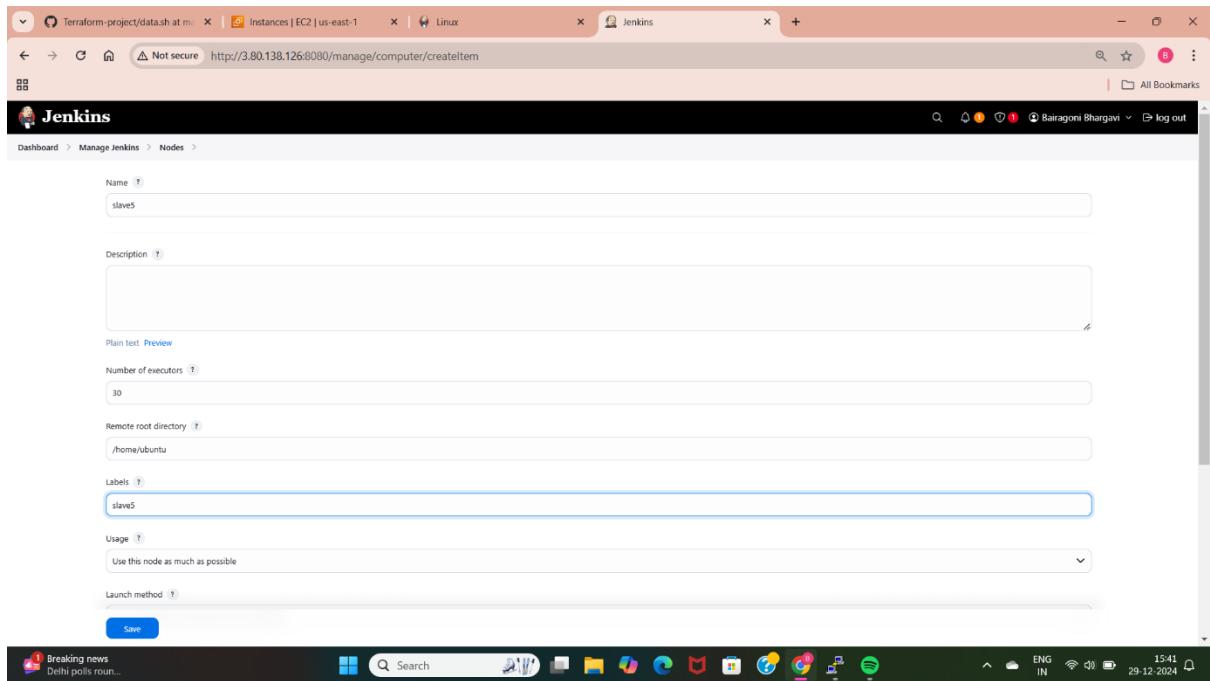
apache-tomcat-9.0.98/bin/version.sh
+ chmod +x apache-tomcat-9.0.98/bin/catalina.sh apache-tomcat-9.0.98/bin/ciphers.sh apache-tomcat-9.0.98/bin/configtest.sh apache-tomcat-9.0.98/bin/daemon.sh
apache-tomcat-9.0.98/bin/digest.sh apache-tomcat-9.0.98/bin/makebase.sh apache-tomcat-9.0.98/bin/setclasspath.sh apache-tomcat-9.0.98/bin/shutdown.sh apache-tomcat-9.0.98/bin/startup.sh apache-tomcat-9.0.98/bin/tool-wrapper.sh apache-tomcat-9.0.98/bin/version.sh
+ cd apache-tomcat-9.0.98/bin/
+ sh startup.sh
Tomcat started.
+ tail -f /home/ec2-user/workspace/slave4/apache-tomcat-9.0.98/logs/catalina.out
NOTE: Picked up JDK_JAVA_OPTIONS: --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED
-add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.rmi=sun.rmi.transport=ALL-UNNAMED
07-Jan-2025 12:30:04.773 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version name: Apache Tomcat/9.0.98
07-Jan-2025 12:30:04.781 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built: Dec 5 2024 19:50:29 UTC
07-Jan-2025 12:30:04.781 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version number: 9.0.98.0
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name: Linux
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version: 5.10.230-223.885.amzn2.x86_64
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture: amd64
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home: /usr/lib/jvm/java-17-amazon-corretto.x86_64
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version: 17.0.13+11-LTS
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor: Amazon.com Inc.
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE: /home/ec2-user/workspace/slave4/apache-tomcat-9.0.98
07-Jan-2025 12:30:04.789 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME: /home/ec2-user/workspace/slave4/apache-tomcat-9.0.98
07-Jan-2025 12:30:04.799 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: --add-opens=java.base/java.lang=ALL-UNNAMED
07-Jan-2025 12:30:04.801 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: --add-opens=java.base/java.io=ALL-UNNAMED
07-Jan-2025 12:30:04.801 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: --add-opens=java.base/java.util=ALL-UNNAMED
07-Jan-2025 12:30:04.802 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: --add-opens=java.base/java.util.concurrent=ALL-UNNAMED
07-Jan-2025 12:30:04.803 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: --add-opens=java.rmi=sun.rmi.transport=ALL-UNNAMED
07-Jan-2025 12:30:04.805 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.config.file=/home/ec2-user/workspace/slave4/apache-tomcat-9.0.98/conf/logging.properties

```

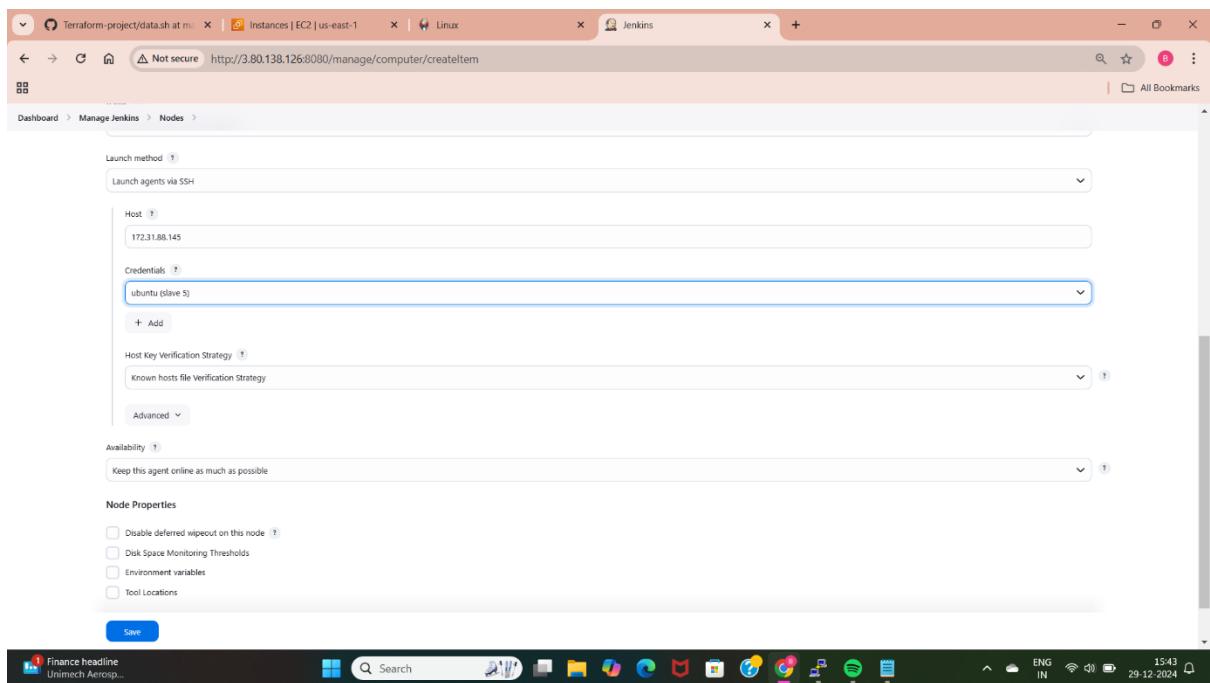
- We can see in the console output, showing that the tomcat is started and checking whether it is started or not.

- This is the tomcat page which is deployed using Jenkins execute shell we can access this using the slave public ip address and tomcat port number 8080.

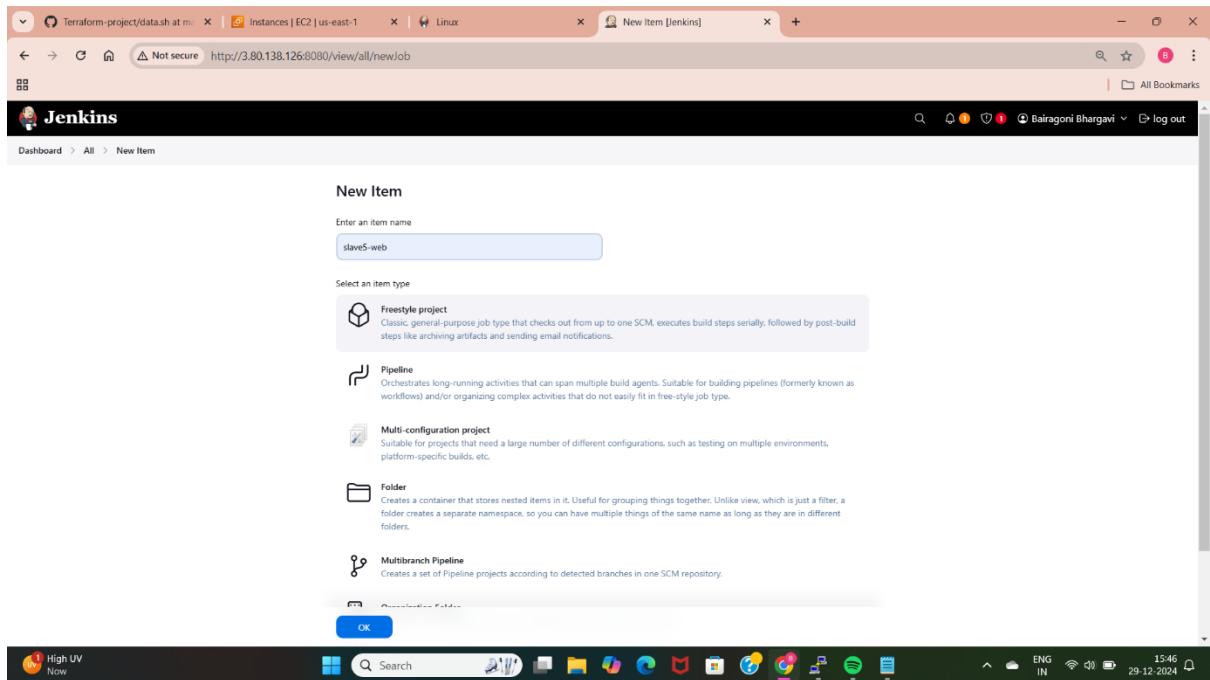
## Slave-5:



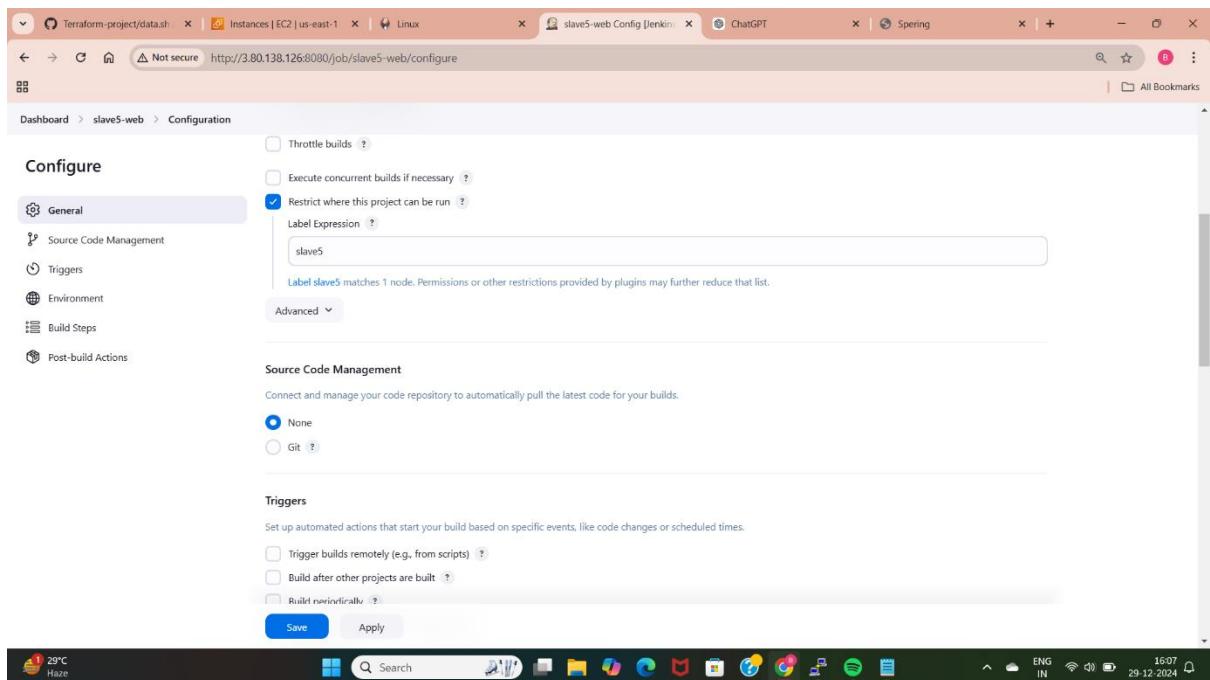
- ☛ Now create a new node with the name slave5 to deploy any static application. Now give the name as slave5, select the number of executors, and provide root directory of slaver server, select use the node as much as possible as usage and give any name for labels.



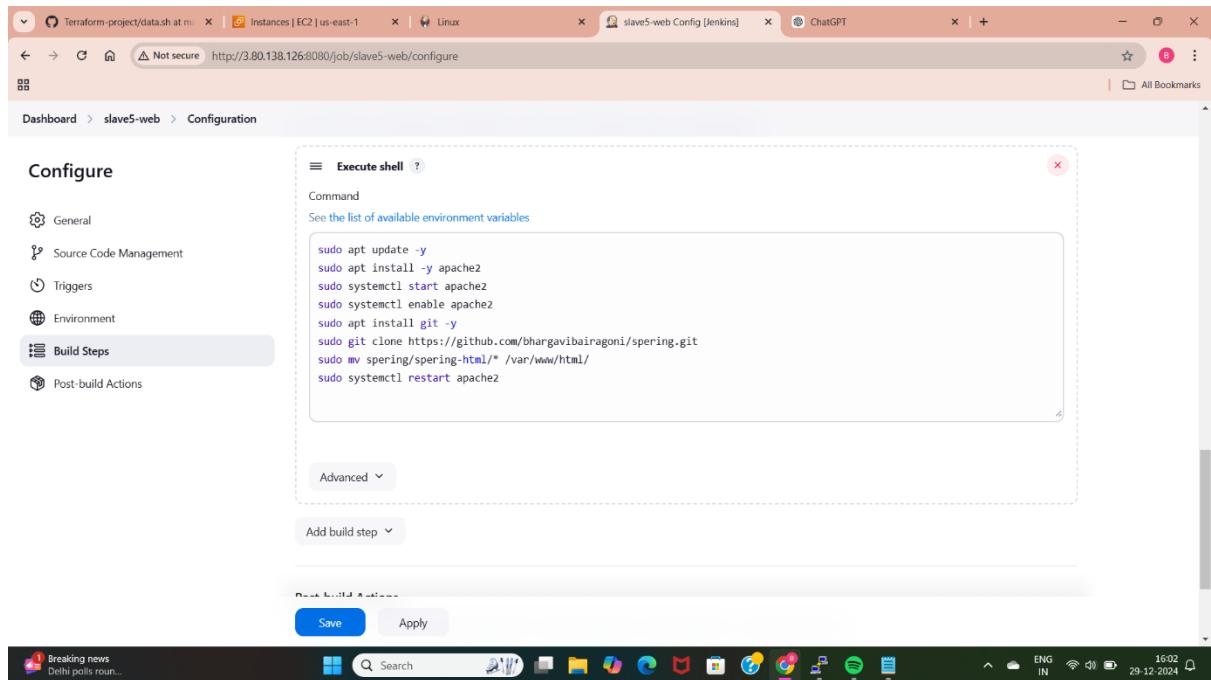
- ☛ Now select the launch method, provide private Ip address of slave5 server, add credentials using above created credentials, verify credentials by selecting the manually trusted key verification strategy.



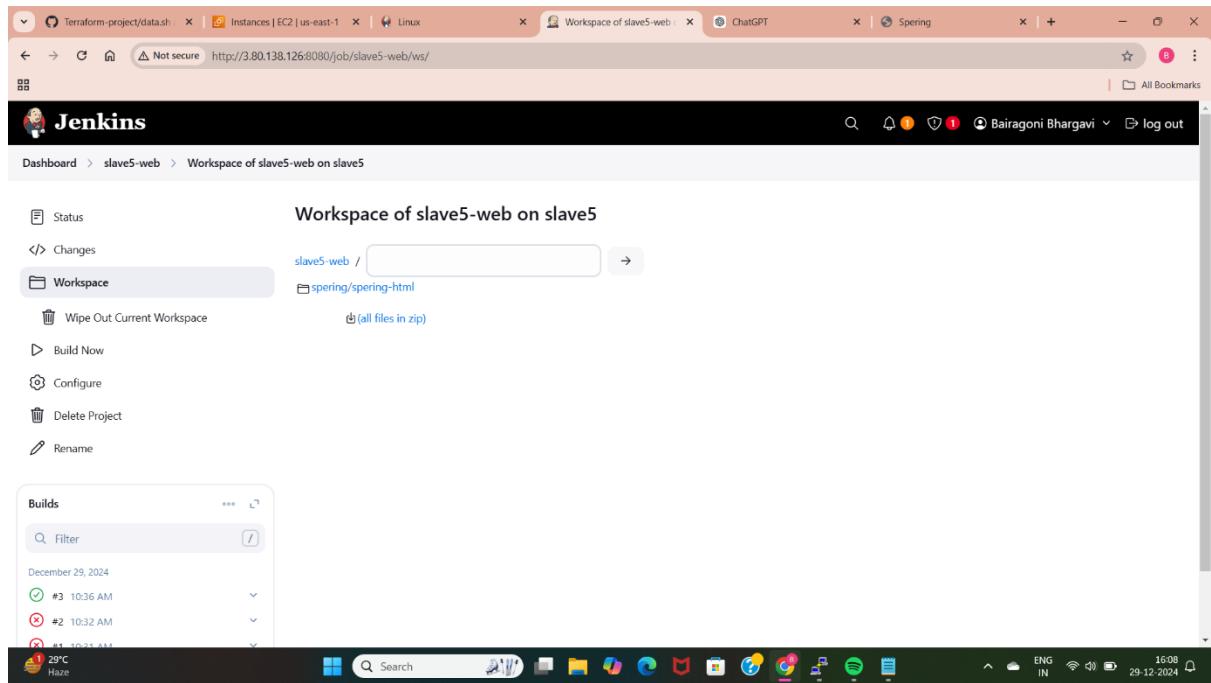
- ☛ Now create a new job to deploy static application. Here am creating a new job with the name “slave-web”.



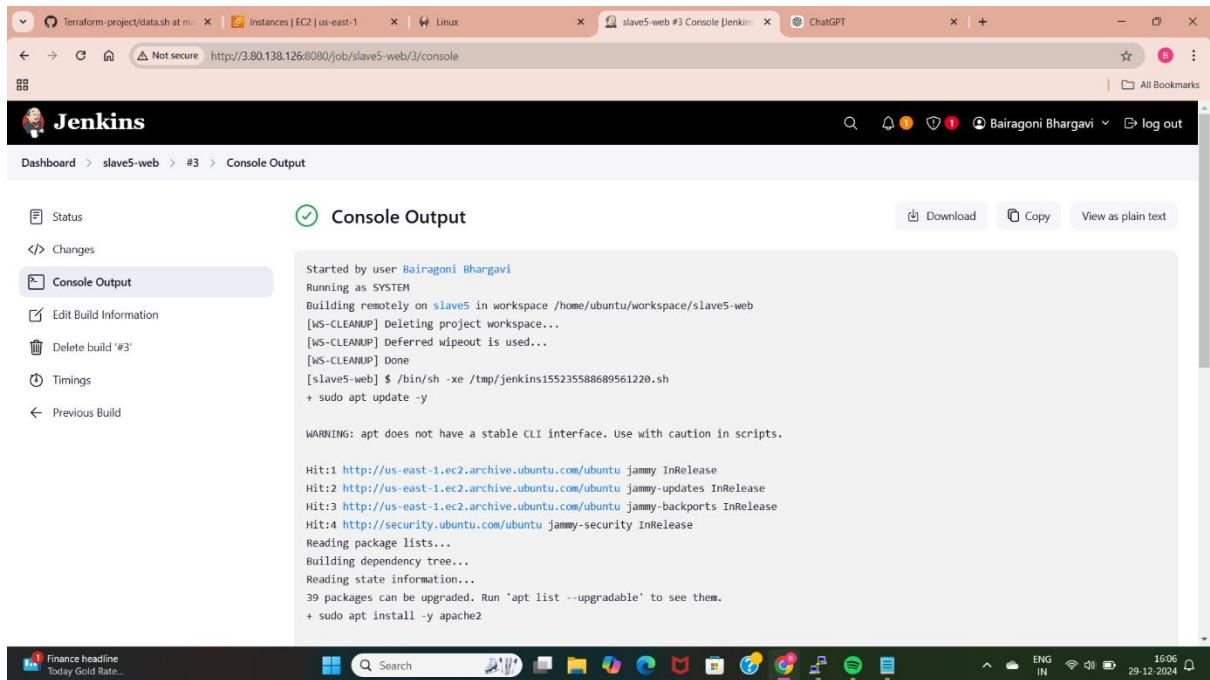
- ☛ Now go to configurations of slave5-web job and select the above created slave5 node label name to restrict where the project can run.



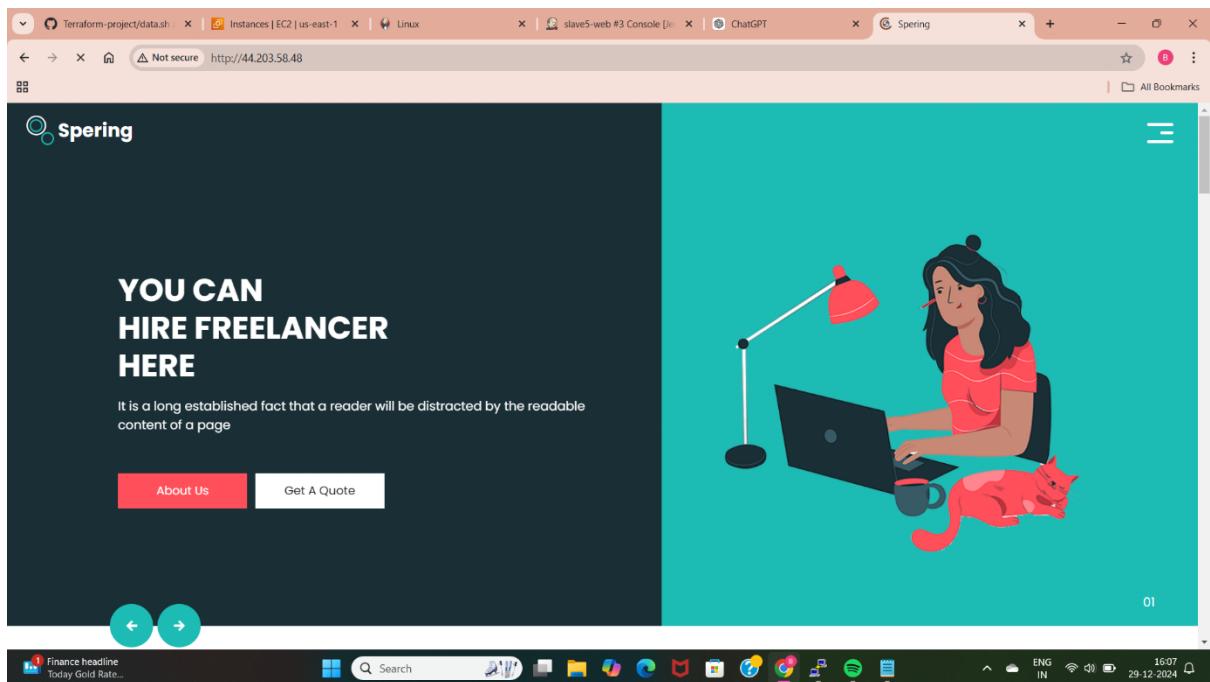
- ⦿ Go to build steps select execute shell. Inside execute shell install httpd, git and start httpd. Take the code from github. Move the files from the repository to the httpd root directory, and restart httpd.



- ⦿ After writing commands execute the commands by clicking on build now option. We can see that the build is success. Here the files are stored in the workspace.



- ➲ We can also check with the console output by clicking on green colored right mark. We can see the build is started by the user Bhargavi and building on workspace, executed the commands to deploy the web application.



- ➲ We can also access the application by using the public IP address of the instance and here we are seeing the spering web application.

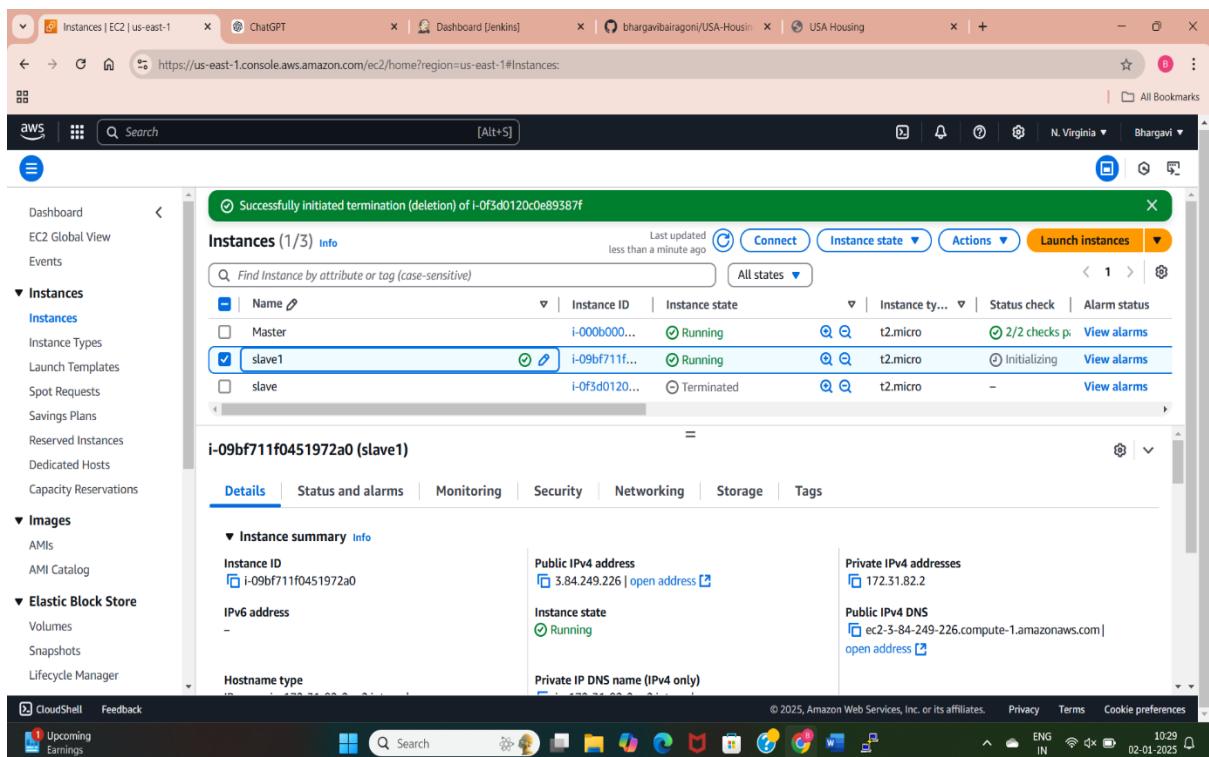
## METHOD-2:

**Create One Master And Five Slave Nodes And Deploy Python Web Applications Through Manually (Slave-1), Bash Script (Slave-2), Terraform (Slave-3) And Deploy Java-Based Application Through Manually (Slave-4) And Bash Script (Execute Shell) (Slave-5).**

In method2 we are creating master slave configurations to deploy python web application, java based application through manually and also using bash script.

Here first we need to create a master server in aws management console. After creating master need to install Jenkins dependency i.e java 17 using `<sudo yum install java-17 -y>` After that we need to install Jenkins repository, token which is having password, install Jenkins and start Jenkins.

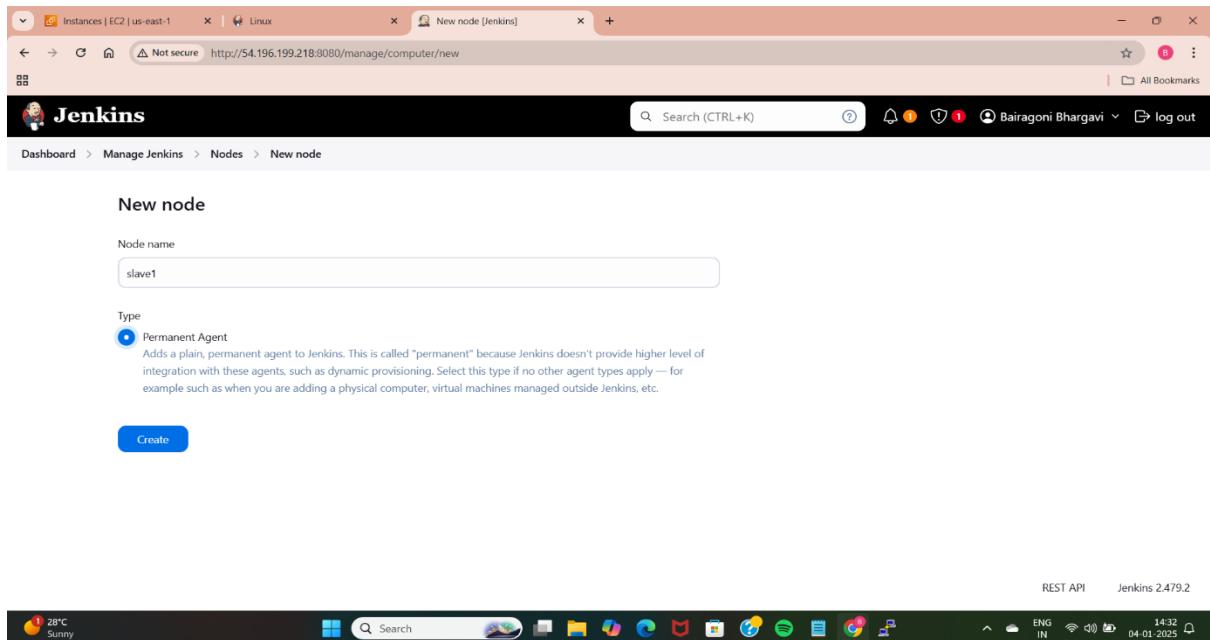
Here in master server we need to generate key pairs so we can use it in slave servers. Take the public key of master using `<sudo vi id_rsa.pub>` and copy the key without any empty spaces.



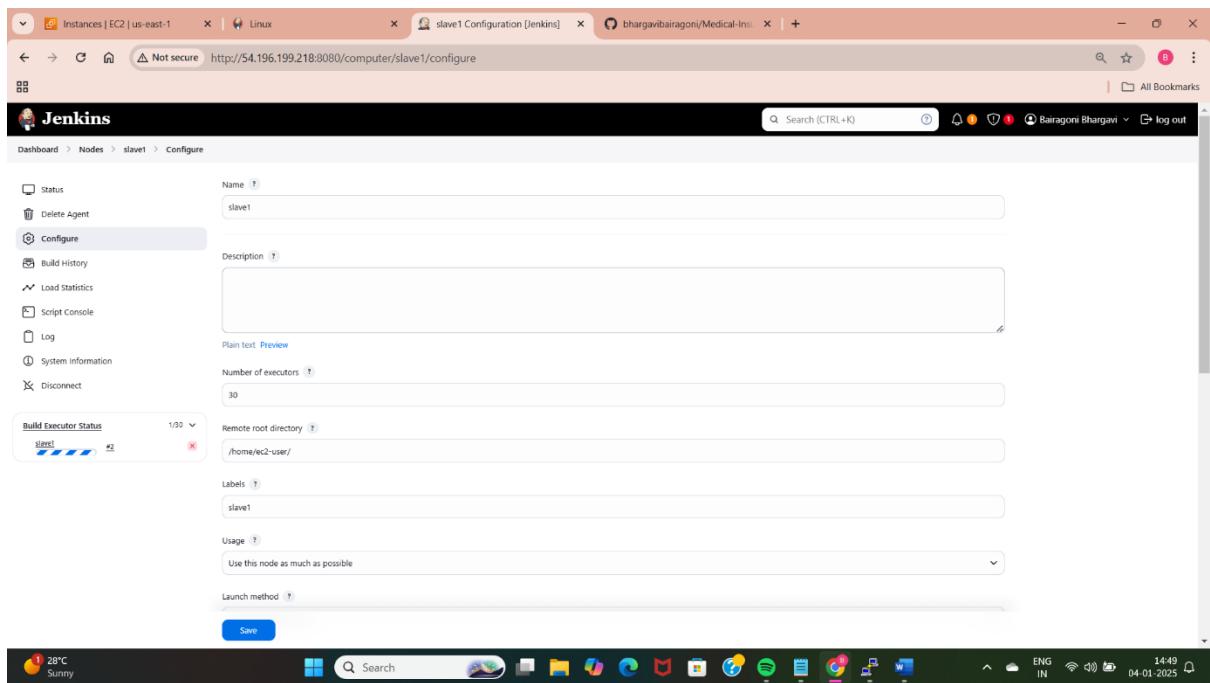
Here am launching one master and slave1 server. After launching slave1 server we need to connect it with master by pasting the masters public key in the slaves authorized keys using `<sudo vi authorized_keys>` don't select insert mode instead select shift + a, enter to next line and paste the key in slave server and save.

Next install git in slave1 server and take the code from the github by using `<git clone>` or by adding the github repository to the server using `<git remote add origin url>`, pull the files from the github.

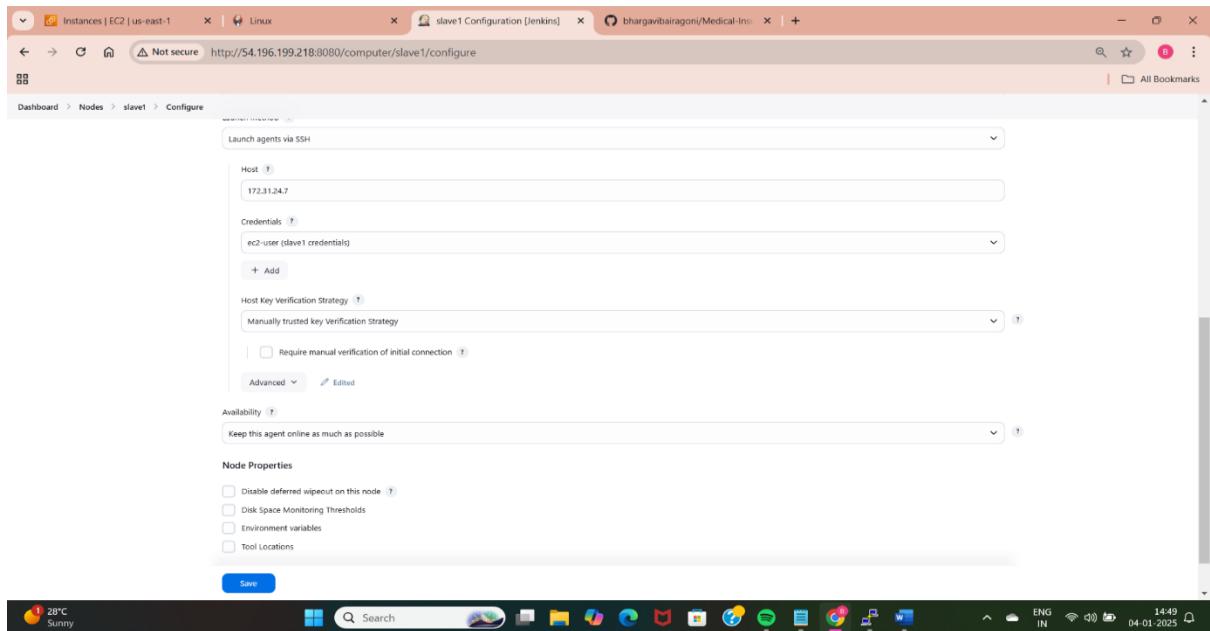
## Slave1 - Deploy Python Web Application:



- ❖ Go to Manage Jenkins > Manage Nodes and Clouds > New Node, give name for the node and select permanent agent and click "ok".



- ❖ Next we need to create a node in Jenkins using the above slave1 server.
- ❖ Here first we need to go to manage Jenkins>Nodes.
- ❖ Now I have given the name as “slave1”, select the number of executors(no. of builds) according to requirement.
- ❖ Give the remote root directory of the slave1 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”.
- ❖ Give the name for label. We will use this label while running builds.
- ❖ Select usage as “use this node as much as possible” to execute builds.

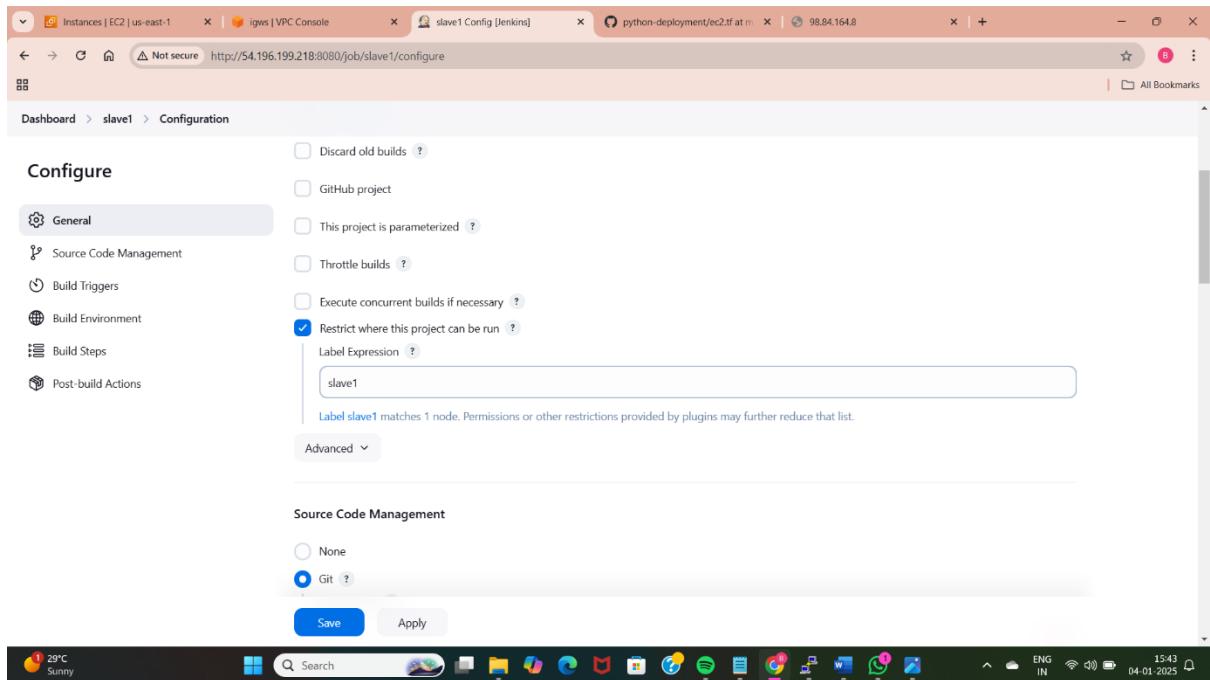


- ❖ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.

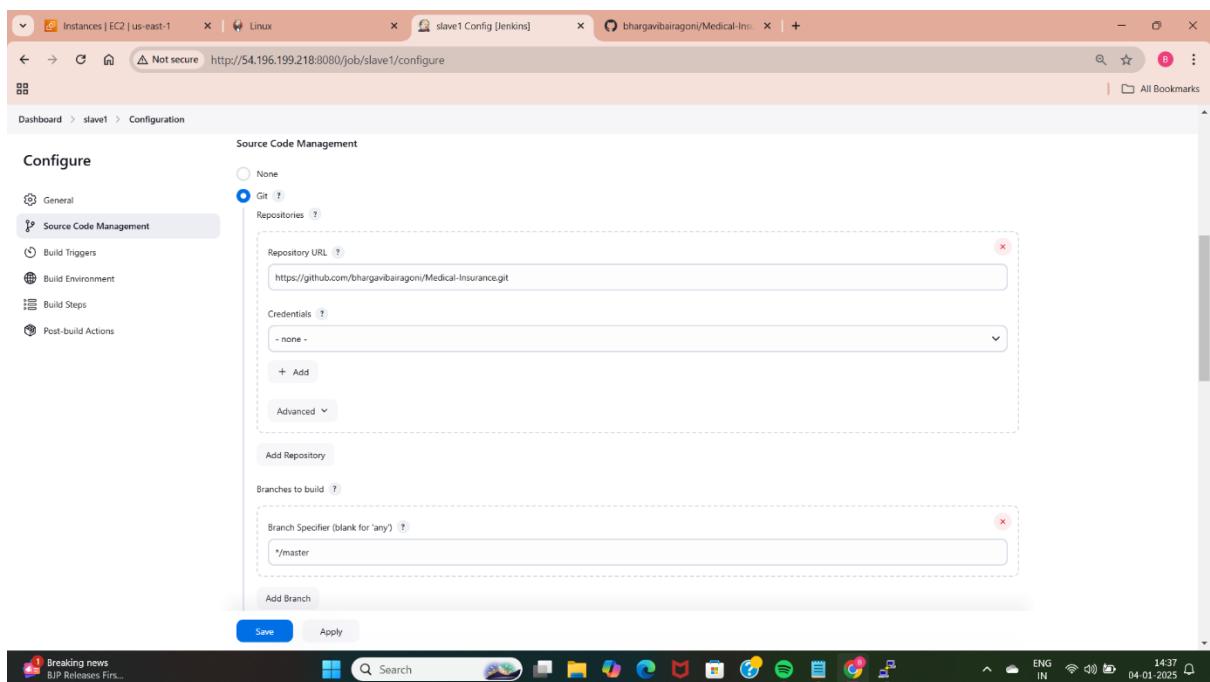
The screenshot shows the Jenkins 'Nodes' page. It lists two nodes: 'Built-In Node' and 'slave1'. The 'slave1' node is highlighted. The table provides details for each node, including Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The 'slave1' node has a status of 'Data obtained' and a response time of '12 min'. A legend at the bottom right indicates icons for S, M, and L.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	15.50 GiB	0 B	15.50 GiB	0ms
	slave1	Linux (amd64)	In sync	5.90 GiB	0 B	5.90 GiB	25ms

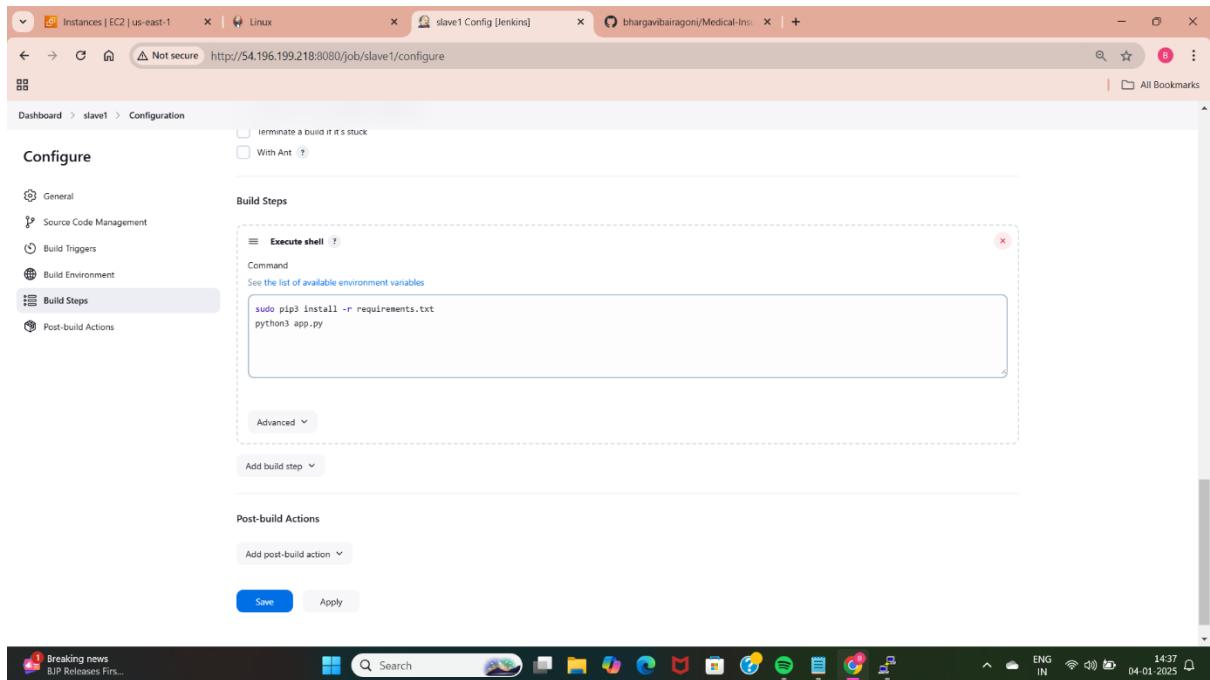
- ❖ Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.



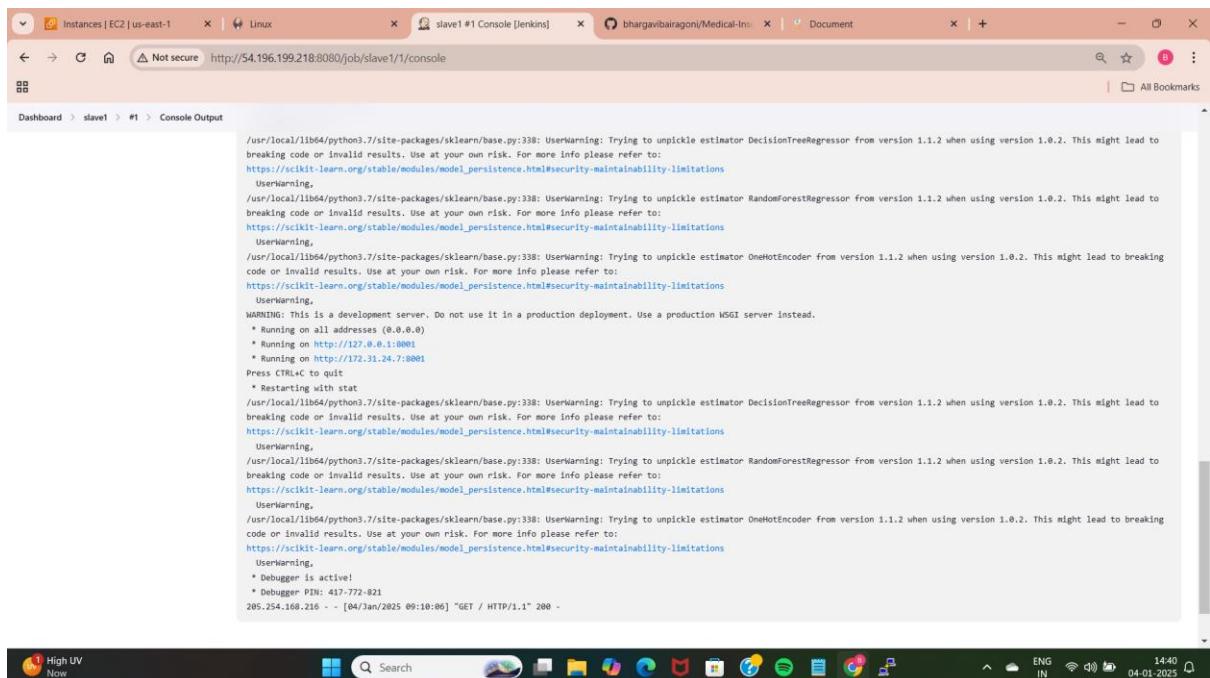
- ❖ Select the above created slave1 by using “Restrict where this project can be run” and select the label name which is created above.



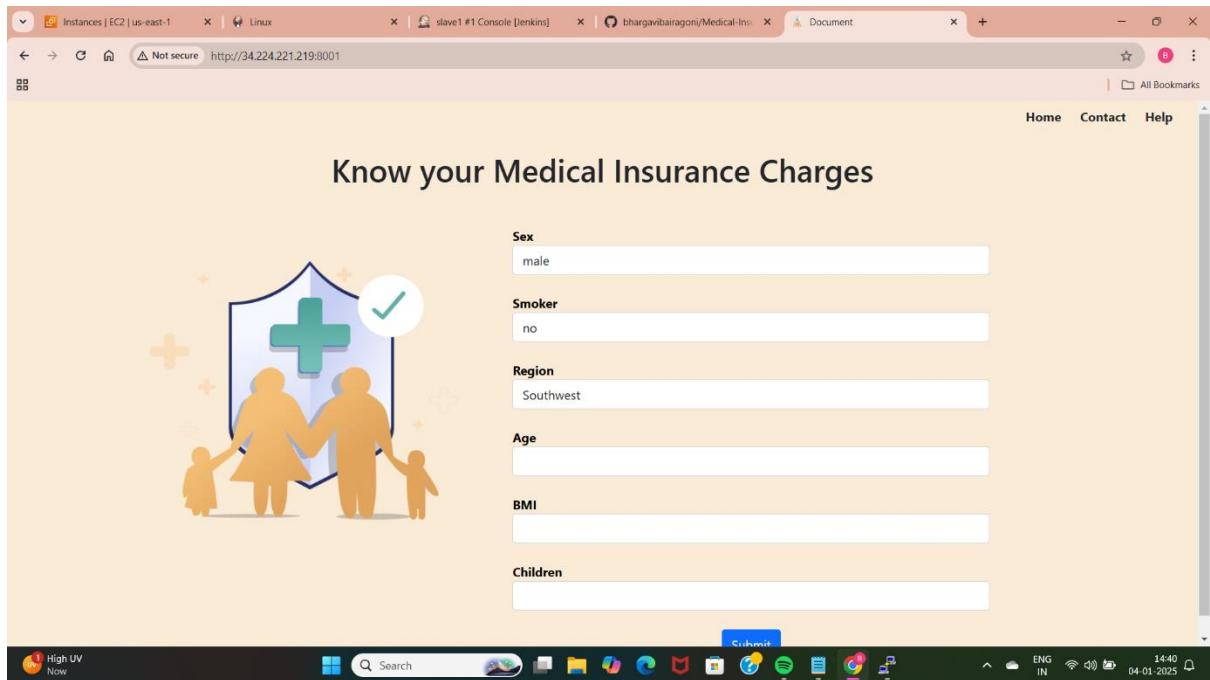
- ❖ Go to source code management in Jenkins and select git, give the github repository url here and select branch where the files are stored.



- ❖ Here I have pulled the files from the github and installing the requirements related to this python application. Save the job configuration and click Build Now.

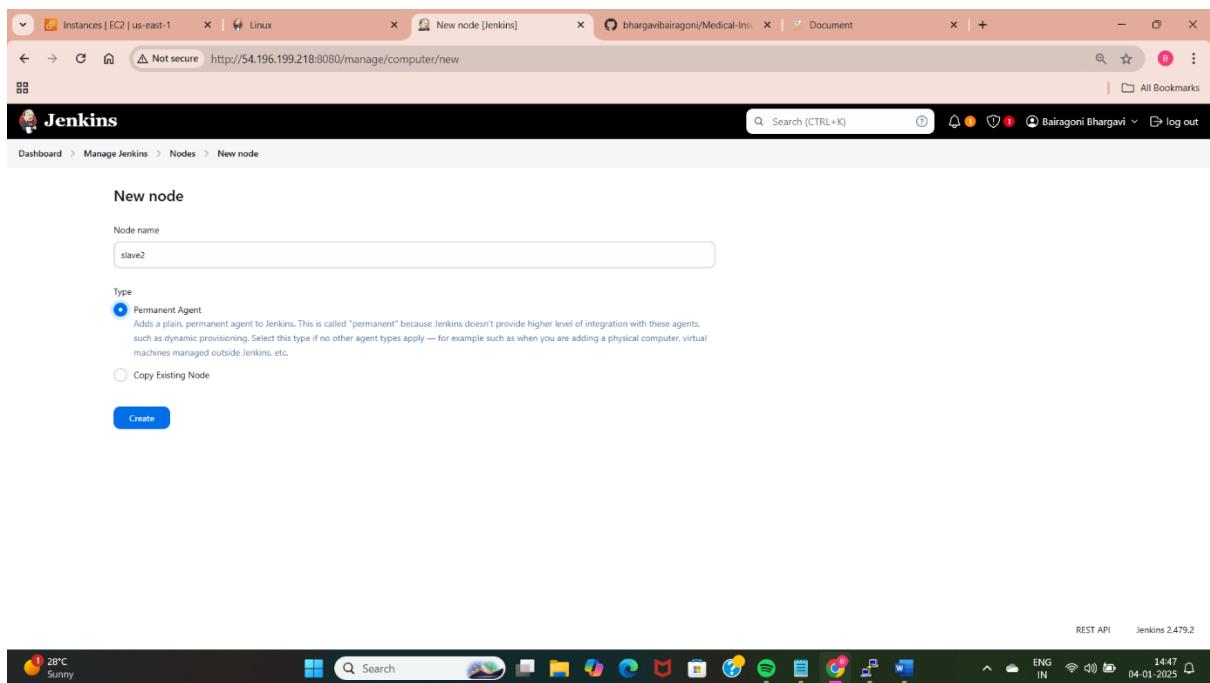


- ❖ Monitor the job's output to ensure the application is deployed on Slave-1. After installing requirements in the slave server we need to deploy the python application manually by using the command <python3 app.py>. Here we can see the url that means our application is deployed successfully without any error.

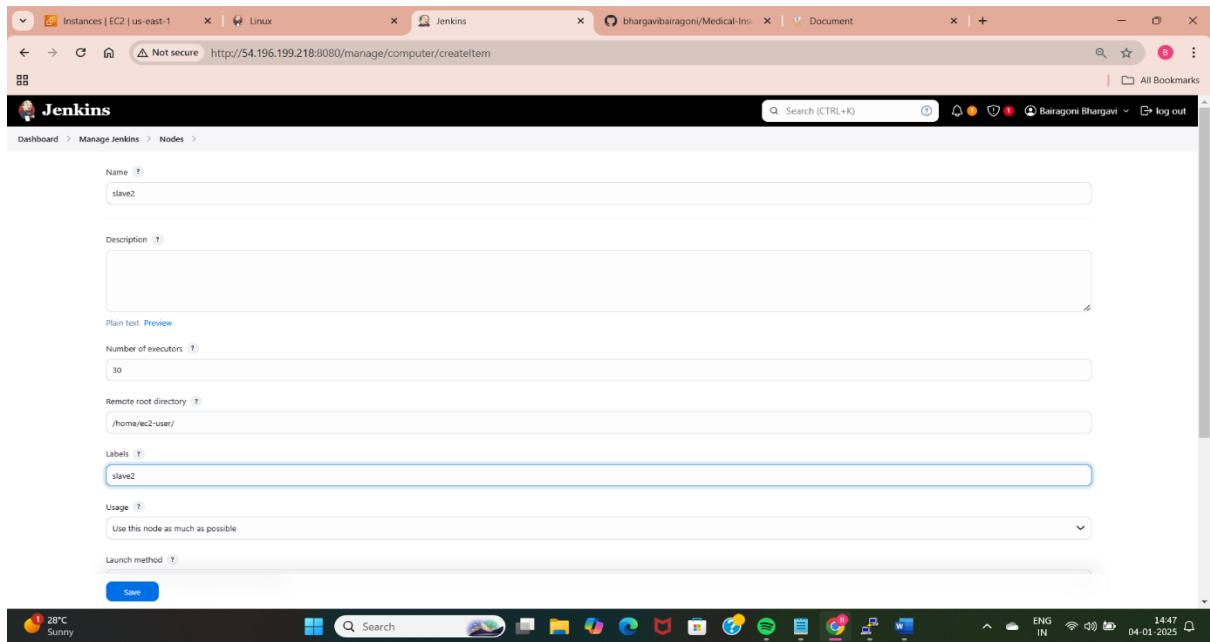


- ❖ After deploying the application we need to access the application to check whether it is working or not by using the public IP address of the slave1 server with application port number in the browser.

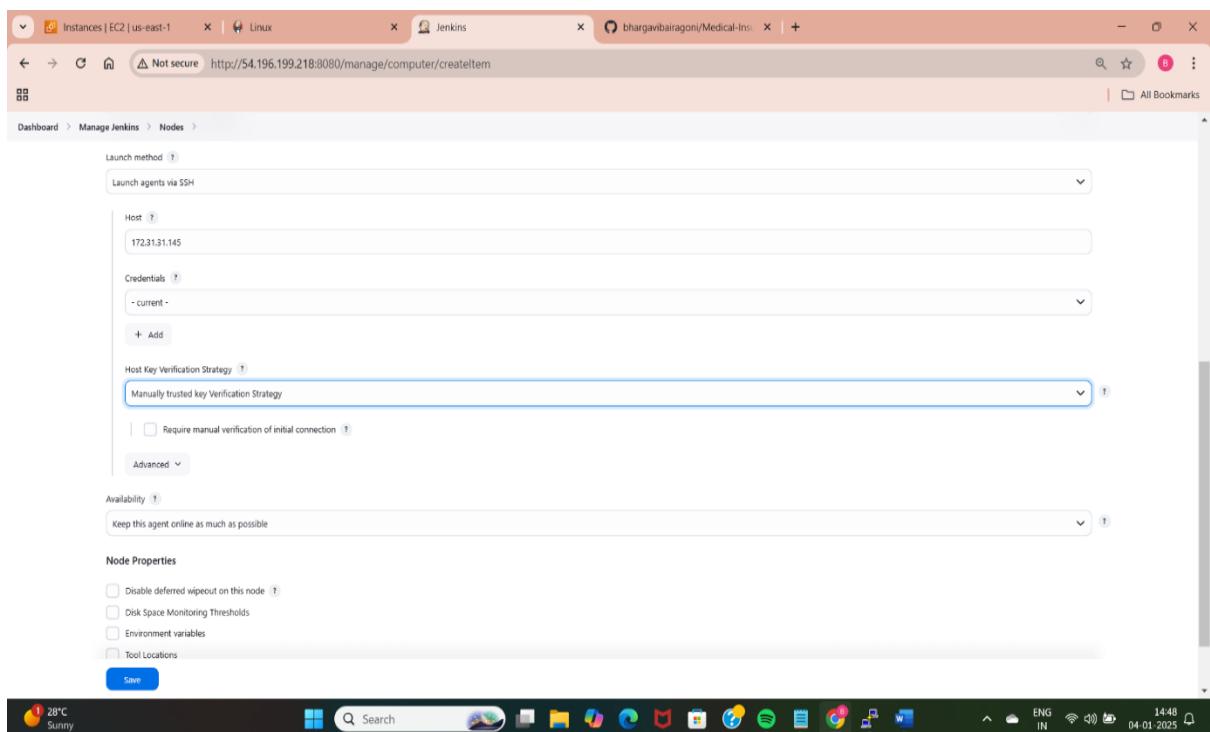
## Slave-2: Deployment Using Bash Script



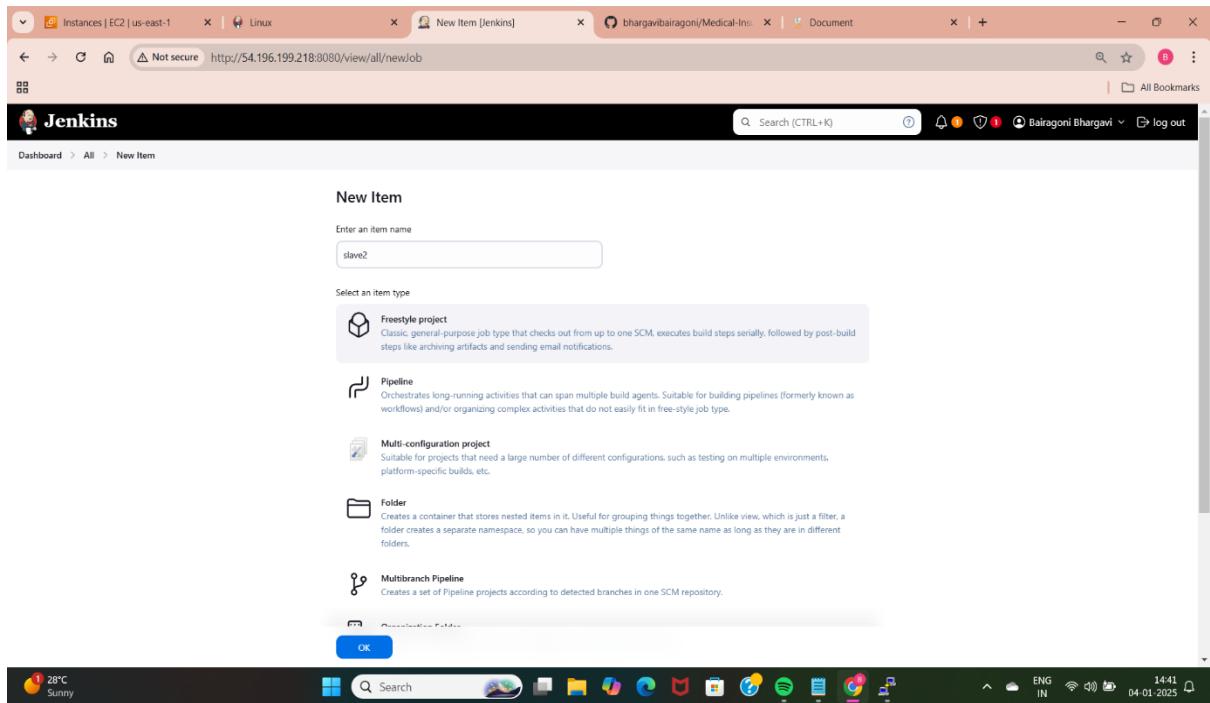
- ➔ Go to Manage Jenkins > Manage Nodes > New Node, give name for the node and select permanent agent and click “ok”.



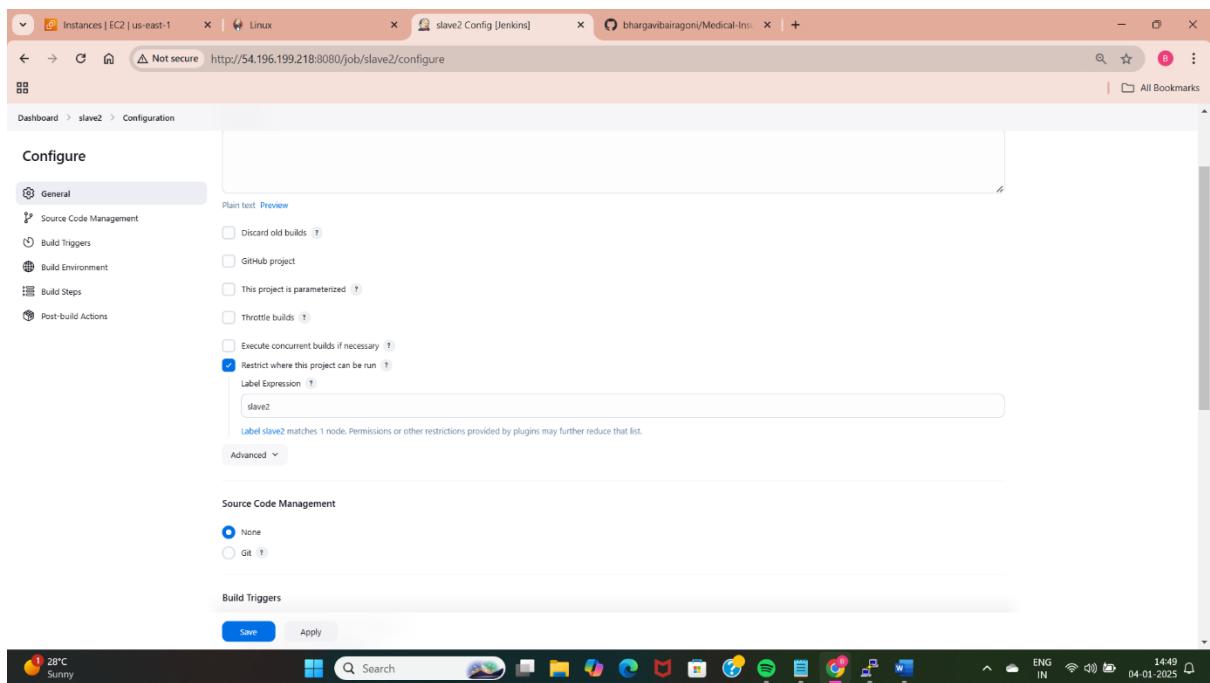
- Next we need to create a node in Jenkins using the above slave2 server.
- Here first we need to go to manage Jenkins>Nodes.
- Now I have given the name as “slave2”, select the number of executors(no. of builds) according to requirement.
- Give the remote root directory of the slave2 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”.
- Give the name for label. We will use this label while running builds.
- Select usage as “use this node as much as possible” to execute builds.



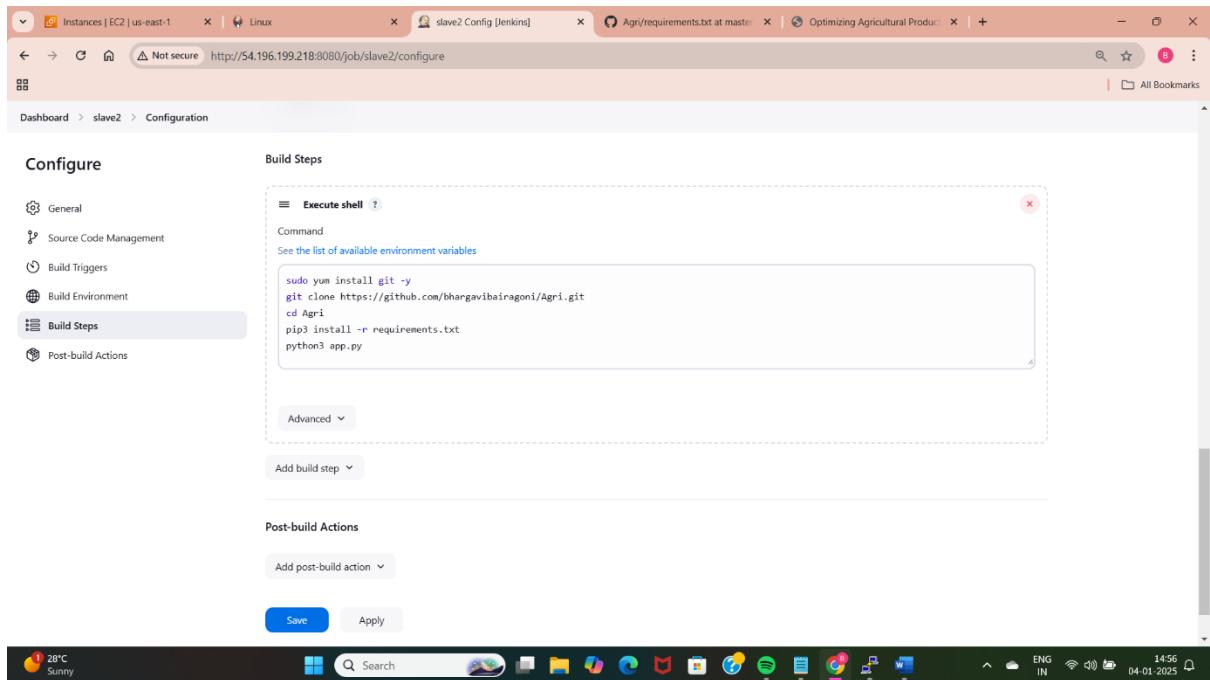
- I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



- Now am creating a new job by selecting new item at dashboard, give the name as slave4 for the new job.

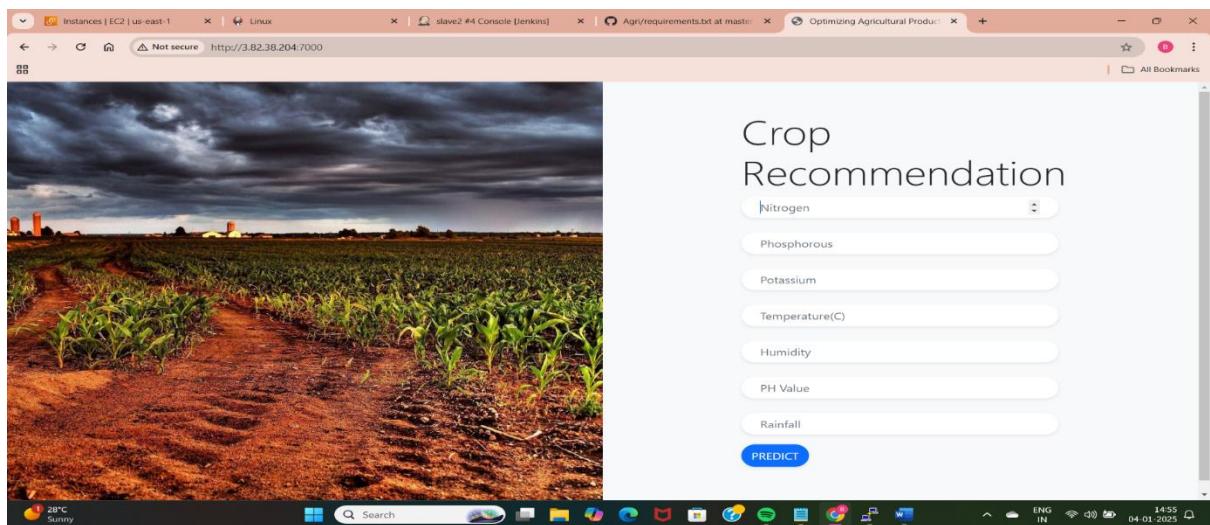


- Select the above created slave2 by using “Restrict where this project can be run” and select the label name which is created above.



After launching the slave2 server we need to deploy the python application using the bash script. Here I have written the script.

- First we need to install the git in the server using <sudo yum install git -y>.
- We need to take the code/python files from github using git clone.
- Go inside the repository where we are having files related to application.
- Install the requirements in the root directory of the server.
- Deploy the application using “python3 app.py”.



- After executing the bash script we need to access the application by using the public IP address of the instance with the application port number.

## Slave-3: Deployment Using Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool created by HashiCorp. It allows you to define, provision, and manage infrastructure using simple configuration files written in HashiCorp Configuration Language (HCL) or JSON. With Terraform, you can automate the deployment and management of resources across multiple cloud providers or on-premises systems.

### Key Features of Terraform

#### 1. Declarative Syntax:

You declare the desired state of your infrastructure (e.g., the number of servers, networking setup, etc.) in configuration files, and Terraform ensures it matches that state.

#### 2. Provider Support:

Terraform supports multiple cloud providers like **AWS, Azure, Google Cloud**, and on-premises solutions like VMware, OpenStack, and Kubernetes.

#### 3. State Management:

Terraform maintains a **state file** to track resources it manages, enabling it to compare the current state of your infrastructure with the desired state and make updates accordingly.

#### 4. Plan and Apply Workflow:

**Plan:** Terraform generates an execution plan, showing what changes it will make to achieve the desired state.

**Apply:** Executes the plan to provision or update the infrastructure.

#### 5. Modular Infrastructure:

Terraform supports reusable code modules, making it easier to manage complex configurations.

#### 6. Immutable Infrastructure:

Instead of modifying existing resources, Terraform often creates new resources and replaces the old ones, ensuring consistency and reducing errors.

## Core Terraform Concepts

#### 1. Providers:

Plugins to interact with specific APIs, like AWS, Azure, Google Cloud, Kubernetes, etc.

#### 2. Resources:

The components you create and manage, such as EC2 instances, S3 buckets, or databases.

#### 3. Variables:

Allow parameterization of configurations for reusability and flexibility.

#### 4. Modules:

Logical groupings of resources to simplify code and promote reuse.

#### 5. State:

A file (`terraform.tfstate`) that keeps track of resources and their relationships.

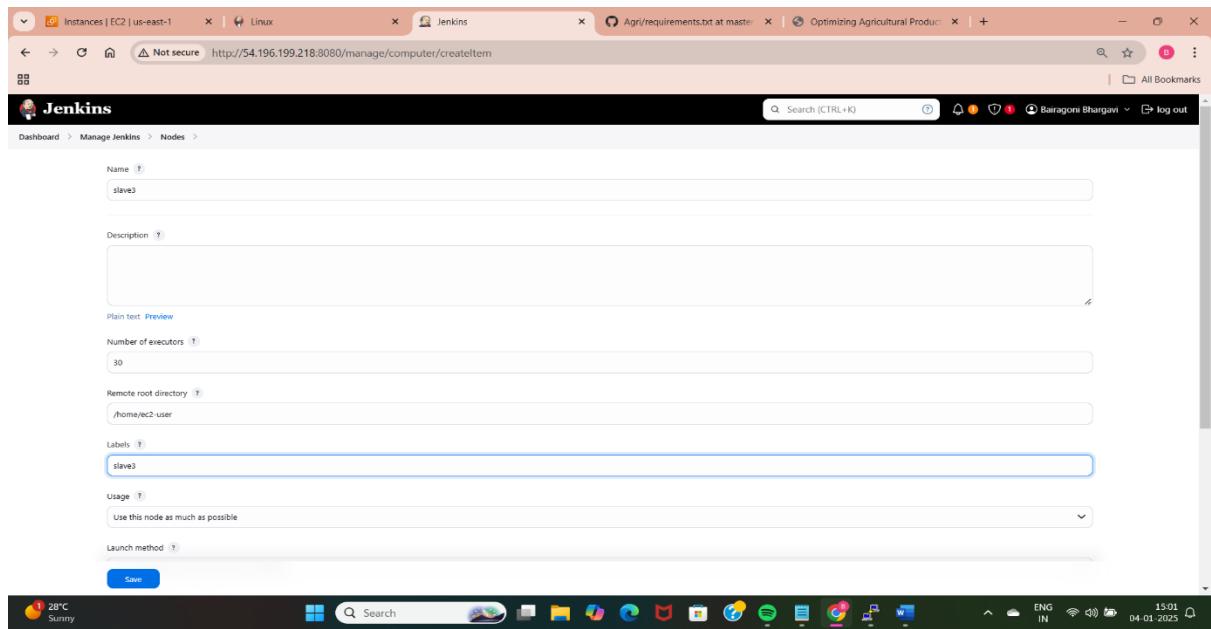
```

ec2-user@slave3:~-
[ec2-user@ip-172-31-31-161 ~]$ sudo hostname slave3
[ec2-user@ip-172-31-31-161 ~]$ exec bash
[ec2-user@slave3 ~]$ sudo vi .ssh/
[ec2-user@slave3 ~]$ cd .ssh/
[ec2-user@slave3 .ssh]$ ll
total 4
-rw-r--r-- 1 ec2-user ec2-user 389 Jan 2 12:39 authorized_keys
[ec2-user@slave3 .ssh]$ sudo vi authorized_keys
[ec2-user@slave3 .ssh]$ sudo yum install -y yum-utils shadow-utils
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package yum-utils-1.1.31-46.amzn2.0.1.noarch already installed and latest version
Package shadow-utils-4.1.5.1-24.amzn2.0.3.x86_64 already installed and latest version
Nothing to do
[ec2-user@slave3 .ssh]$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
adding repo from https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
grabbing file https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo to /etc/yum.repos.d/hashicorp.repo
repo saved to /etc/yum.repos.d/hashicorp.repo
[ec2-user@slave3 .ssh]$ sudo yum -y install terraform
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
hashicorp
hashicorp/x86_64/primary
hashicorp
Resolving Dependencies
--> Running transaction check
--> Package terraform.x86_64 0:1.10.3-1 will be installed
--> Processing Dependency: git for package: terraform-1.10.3-1.x86_64
--> Running transaction check
--> Package git.x86_64 0:2.40.1-1.amzn2.0.3 will be installed
--> Processing Dependency: git-core = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl-Git = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Git) for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Running transaction check
--> Package git-core.x86_64 0:2.40.1-1.amzn2.0.3 will be installed

```

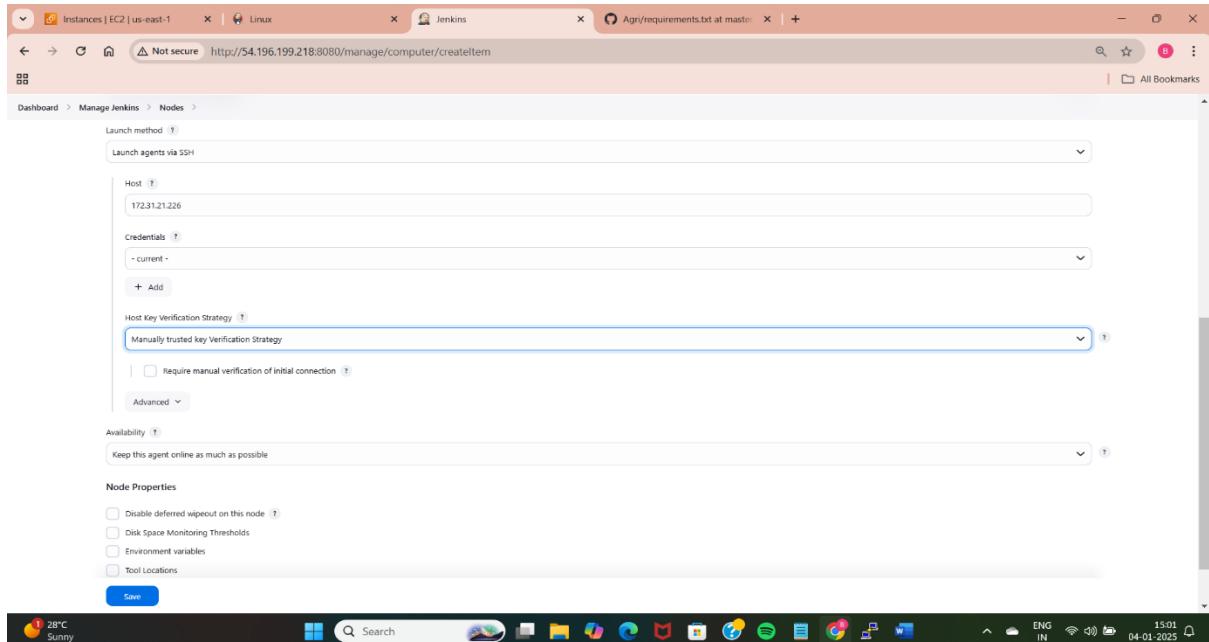
Now launch slave3 server and connect it with the terminal.

- ♣ Here am changing the hostname to slave3 using <sudo hostname slave3>, applied the name to the terminal using <exec bash>.
- ♣ Now connect the slave3 with the master by using the public key of master. For that we need to move into <sudo vi .ssh/>. There we will find the authorized\_keys files. Copy the masters public ip in the slaves authorized keys.

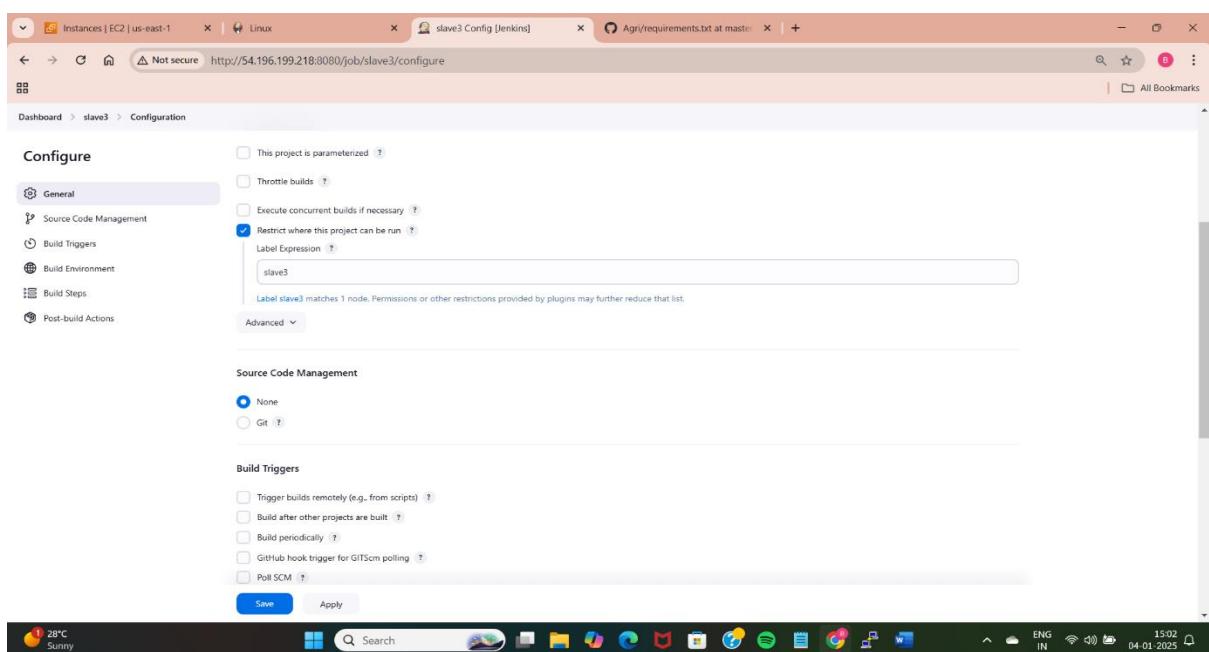


- ✓ Next we need to create a node in Jenkins using the above slave3 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave3”, select the number of executors(no. of builds) according to requirement.

- ✓ Give the remote root directory of the slave3 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.



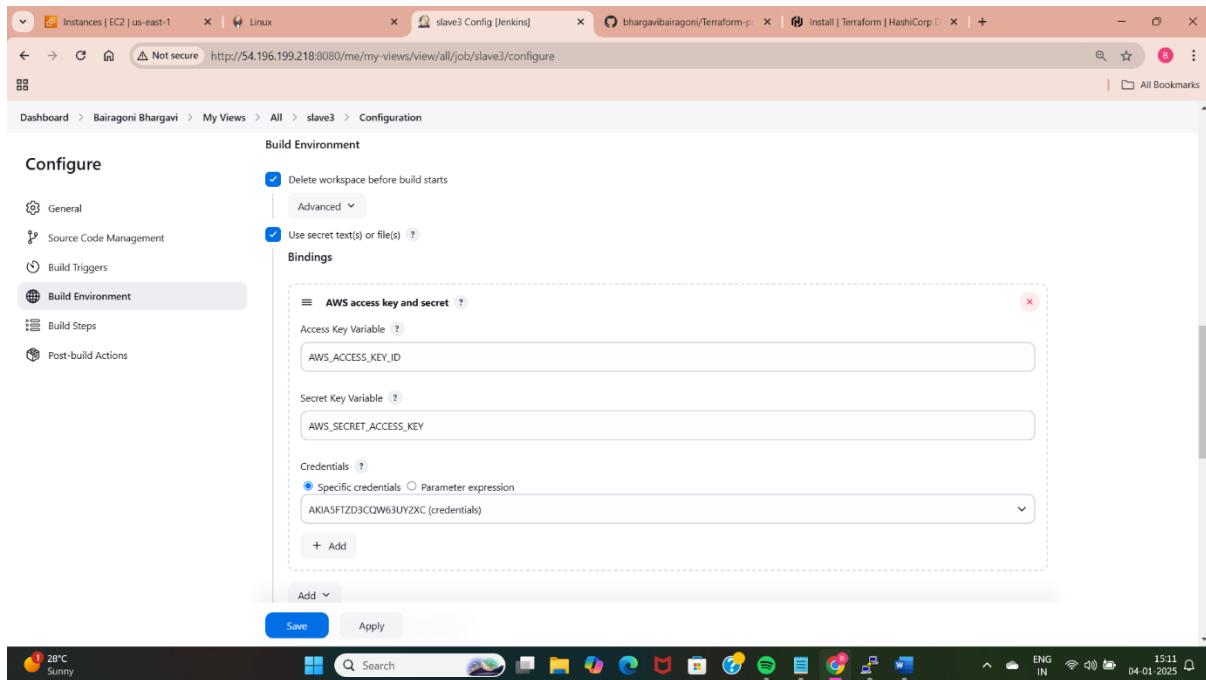
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



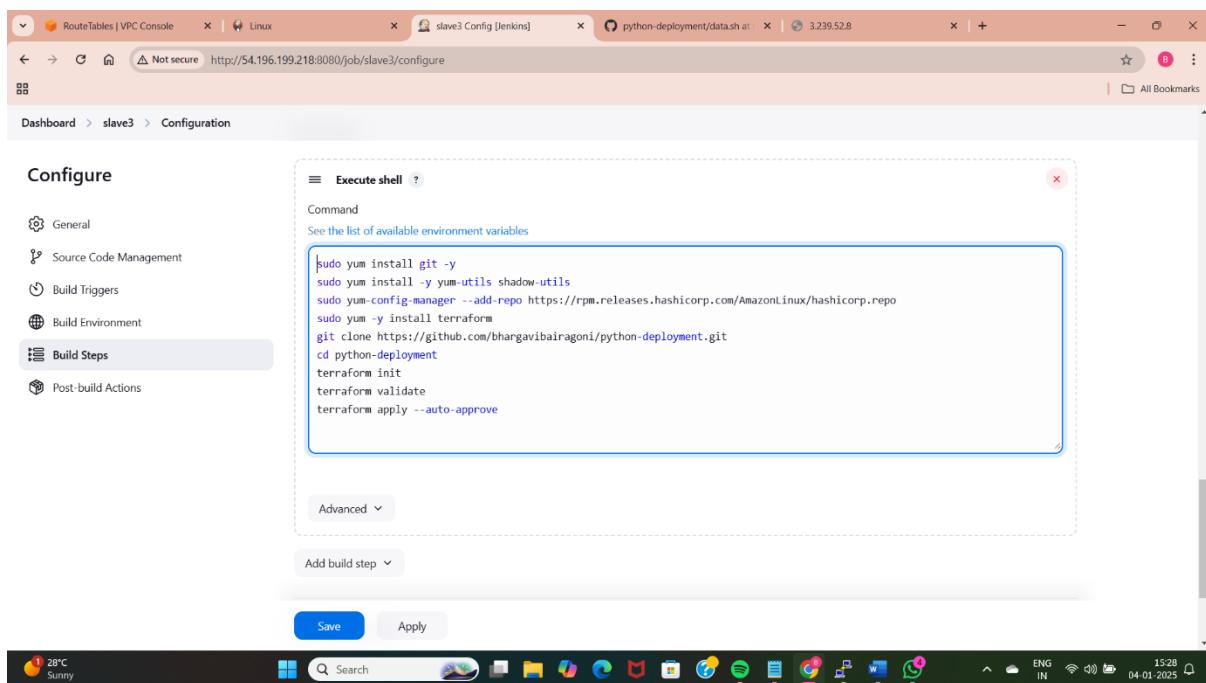
- ✓ Select the above created slave3 by using “Restrict where this project can be run” and select the label name which is created above.

- ♣ Go to manage Jenkins select plugins>available plugins install aws credentials plugin to provide aws configurations.

- ♣ Go to manage select credentials and add aws configurations using aws credentials and give any id and description. Next give created aws access keys and secret access keys there.



- Provide aws configurations using the access keys and secret access keys using the aws credentials plugin.



- Install git to take the code from github which is having files to create aws resources using the terraform.
- Install terraform in the terminal using the bash script next clone the github repository which is having files related to creation of ec2 instance, vpc.
- After that we need to initialize the terraform in created files.
- Check the validation of the code/files using the <terraform validate>.
- Apply the changes using <terraform apply --auto-approve>.

```

Started by user Bairagoni Bhargavi
Running as SYSTEM
Building remotely on slave3 in workspace /home/ec2-user/workspace/slave3
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[slave3] $ /bin/sh -xe /tmp/jenkins4425140142379112736.sh
+ sudo yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package git-2.40.1-1.amzn2.0.3.x86_64 already installed and latest version
Nothing to do
+ sudo yum install -y yum-utils shadow-utils
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package yum-utils-1.1.31-46.amzn2.0.1.noarch already installed and latest version
Package z:shadow-utils-4.1.5.1-24.amzn2.0.3.x86_64 already installed and latest version
Nothing to do
+ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
adding repo from https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
grabbing file https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo to /etc/yum.repos.d/hashicorp.repo
repo saved to /etc/yum.repos.d/hashicorp.repo

```

- ✓ Monitor the job's output to ensure the application is deployed on Slave-3. Here the build is started by the user Bairagoni Bhargavi, build is started by slave3 node.

```

{
    "aws_vpc": {
        "name": "DEMO_VPC"
    }
}

@{32m} @{0m} @{0m} tags_all = {
    "@{32m} @{0m} @{0m} "Name" = "DEMO_VPC"
}
}

@{1m} plan:@{0m} 7 to add, 0 to change, 0 to destroy.
@{0m} @{0m} @{1m}aws_vpc: Creating...@{0m}
@{0m} @{1m}aws_vpc: Creation complete after 1s [id=vpc-0bad401bfbaf6d9fc3]@{0m}
@{0m} @{1m}aws_security_group: Creating...@{0m}
@{0m} @{1m}aws_subnet:public_subnet-1: Creating...@{0m}
@{0m} @{1m}aws_internet_gateway: Creating...@{0m}
@{0m} @{1m}aws_route_table:route: Creating...@{0m}
@{0m} @{1m}aws_route_table:route: Creation complete after 1s [id=rtb-0653a4b6b9a0441a2]@{0m}
@{0m} @{1m}aws_security_group: Creation complete after 3s [id=sg-0e9ae5b19ac00fdfe]@{0m}
@{0m} @{1m}aws_subnet:public_subnet-1: Still creating... [10s elapsed]@{0m}
@{0m} @{1m}aws_subnet:public_subnet-1: Creation complete after 11s [id=subnet-06a7b017366dcc254]@{0m}
@{0m} @{1m}aws_instance:public_subnet-1@{0}: Creating...@{0m}
@{0m} @{1m}aws_route_table_association:rti: Creating...@{0m}
@{0m} @{1m}aws_route_table_association:rti: Creation complete after 1s [id=rtbassoc-0f05ae8485c053dda]@{0m}
@{0m} @{1m}aws_instance:public_subnet-1@{0}: Still creating... [10s elapsed]@{0m}
@{0m} @{1m}aws_instance:public_subnet-1@{0}: Creation complete after 12s [id=i-008280e5702713eac]@{0m}
@{0m} @{1m} finished: SUCCESS

```

- ♣ Here we can see all the configurations i.e vpc, security group, instance, internet gateway, route table subnets are created.

```

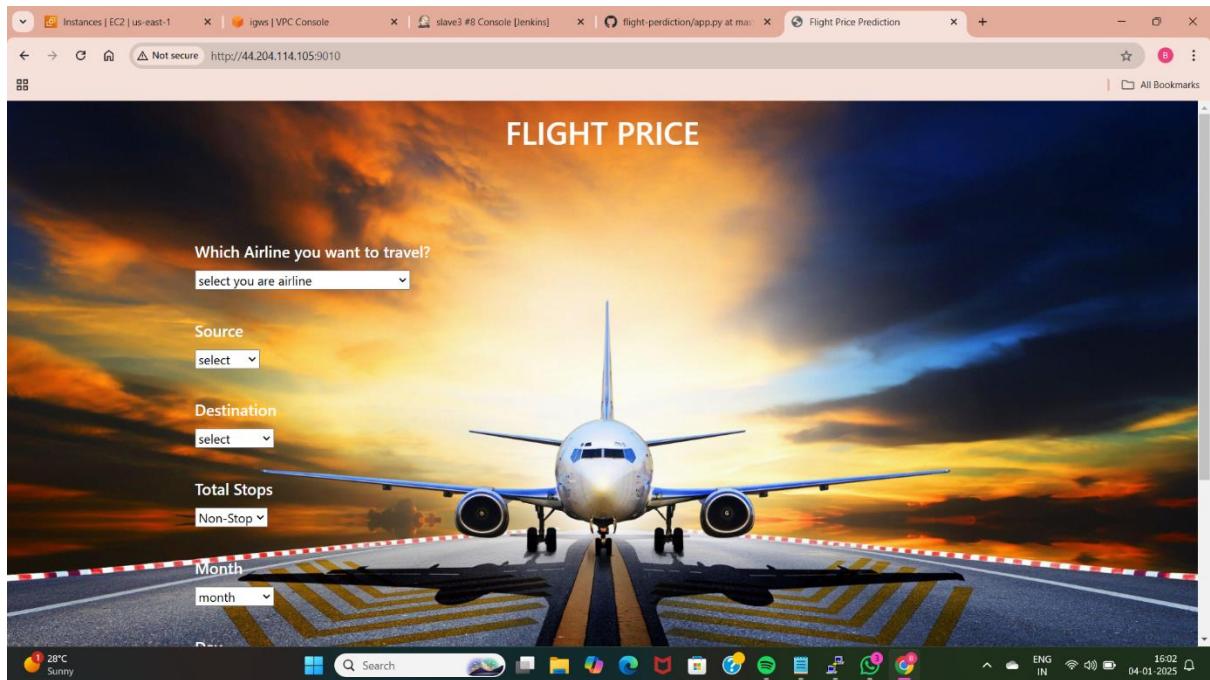
[ec2-user@slave3:~/workspace/slave3/python-deployment]
$ ls
total 1360
drwxr-x 4 ec2-user ec2-user 34 Jan 4 09:31 remoting
-rw-r--r-- 1 ec2-user ec2-user 1393083 Jan 4 09:31 remoting.jar
-rw-r--r-- 1 root root 181 Jan 4 09:55 terraform.tfstate
drwxr-x 3 ec2-user ec2-user 20 Jan 4 10:19 workspace
[ec2-user@slave3 workspace]$ ls
total 0
drwxr-x 3 ec2-user ec2-user 31 Jan 4 10:19 slave3
[ec2-user@slave3 workspace]$ cd slave3/
[ec2-user@slave3 slave3]$ ls
total 0
drwxr-x 4 ec2-user ec2-user 296 Jan 4 10:20 python-deployment
[ec2-user@slave3 slave3]$ cd python-deployment/
[ec2-user@slave3 python-deployment]$ ls
total 9916
-rw-r--r-- 1 ec2-user ec2-user 275 Jan 4 10:19 data.sh
-rw-r--r-- 1 ec2-user ec2-user 4389098 Jan 4 10:19 DEPLOYING_PYTHON_APPLICATION.pptx
-rw-r--r-- 1 ec2-user ec2-user 363 Jan 4 10:19 deployment.yml
-rw-r--r-- 1 ec2-user ec2-user 799 Jan 4 10:19 provider.tfc
-rw-r--r-- 1 ec2-user ec2-user 114 Jan 4 10:19 igw.tf
-rw-r--r-- 1 ec2-user ec2-user 44 Jan 4 10:19 provider.tf
-rw-r--r-- 1 ec2-user ec2-user 5711453 Jan 4 10:19 python-project deployment1.pdf
-rw-r--r-- 1 ec2-user ec2-user 328 Jan 4 10:19 route.tf
-rw-r--r-- 1 ec2-user ec2-user 181 Jan 4 10:19 svc.yml
-rw-r--r-- 1 ec2-user ec2-user 15116 Jan 4 10:20 terraform.tfstate
-rw-r--r-- 1 ec2-user ec2-user 300 Jan 4 10:19 vpc.tf
[ec2-user@slave3 python-deployment]$

```

- We can also check in the terminal when build is success whether the files are saved in Jenkins path or not. For that move to terminal and check files and find workspace. If workspace is found move inside the workspace and see the created job. Move inside the job and check whether files are deployed or not.
- Here we can see all the files related to terraform.

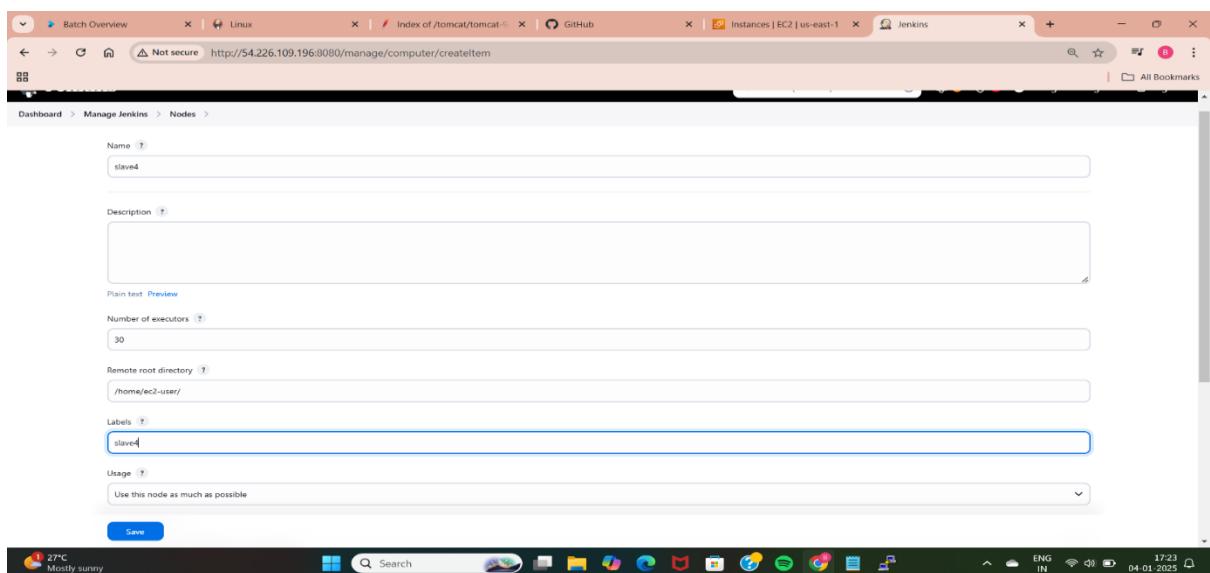
Name	Instance ID	Instance state	Type	Status check	Alarm status	Availability zone	Public IP	Elastic IP	IPv6
master	i-04e193c77...	Running	t2.micro	2/2 checks p	View alarms +	us-east-1c	ec2-54-19...	54.196.199.218	-
slave2	i-007a8672...	Running	t2.micro	2/2 checks p	View alarms +	us-east-1c	ec2-3-82-5...	3.82.38.204	-
slave1	i-005a801...	Running	t2.micro	2/2 checks p	View alarms +	us-east-1c	ec2-34-22...	34.224.221.219	-
slave3	i-058a09dc...	Running	t2.micro	2/2 checks p	View alarms +	us-east-1c	ec2-3-89-4...	3.89.49.242	-
Terraform	i-008280e5...	Running	t2.micro	Initializing	View alarms +	us-east-1a	-	44.204.114.105	-

- After applying the changes it is going to create the aws instance with t2.micro, associating the public ip address to the created instance. Similarly it is creating all the configurations which are mentioned in the terraform files.

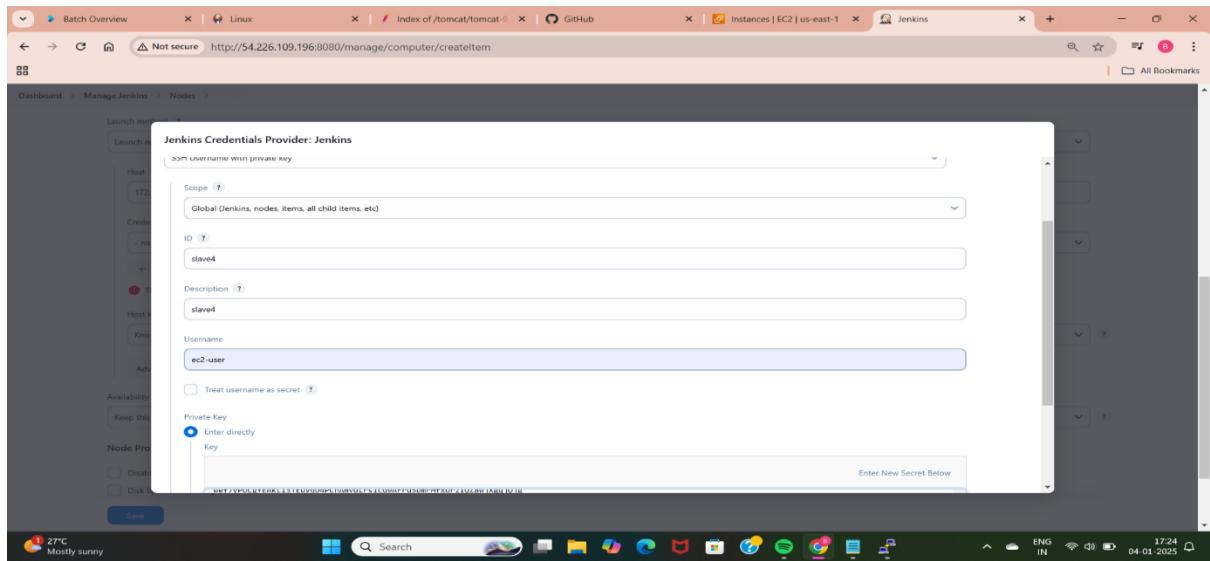


- Now to get the application we need to copy the public ip address of the slave server and paste in browser to access the application.

## Slave-4: Deploy Java-Based Applications (Manual Deployment)

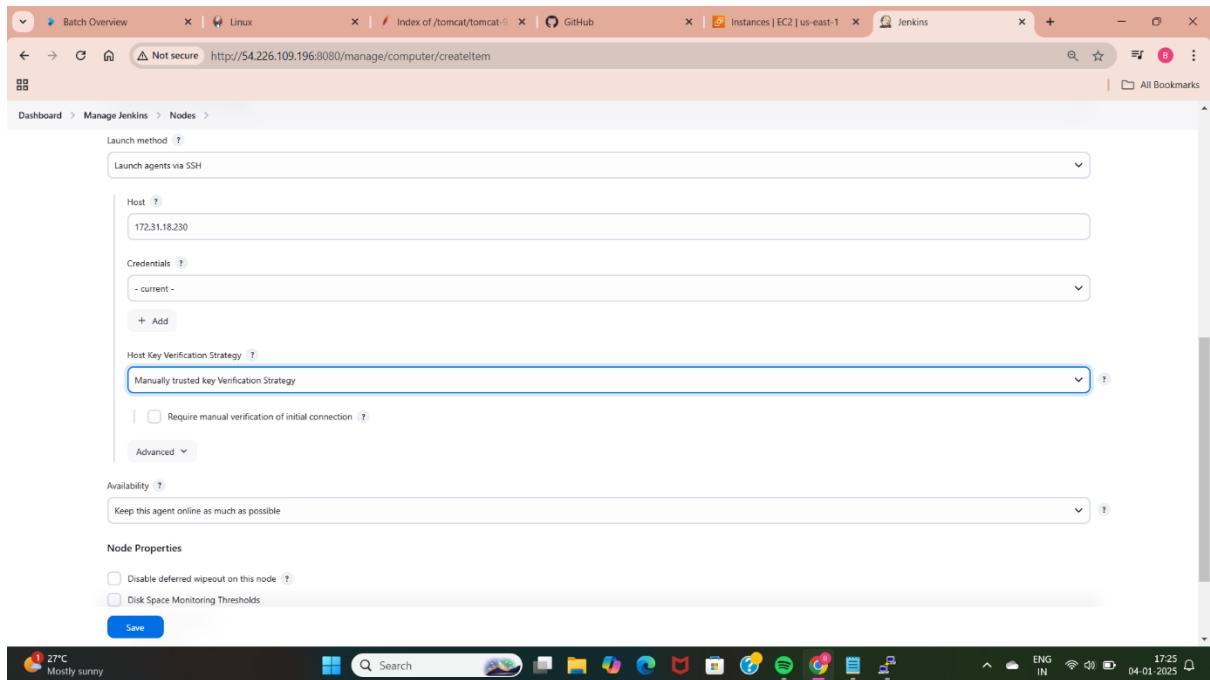


- Next we need to create a node in Jenkins using the above slave4 server.
- Here first we need to go to manage Jenkins>Nodes.
- Now I have given the name as “slave4”, select the number of executors(no. of builds) according to requirement.
- Give the remote root directory of the slave4 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- Give the name for label. We will use this label while running builds.
- Select usage as “use this node as much as possible” to execute builds.



Add SSH credentials:

- ID: Assign a name for the credential.
- Username: Enter the slave's instance username (ubuntu, ec2-user, etc.).
- Private Key: Paste the private key generated on the master server.



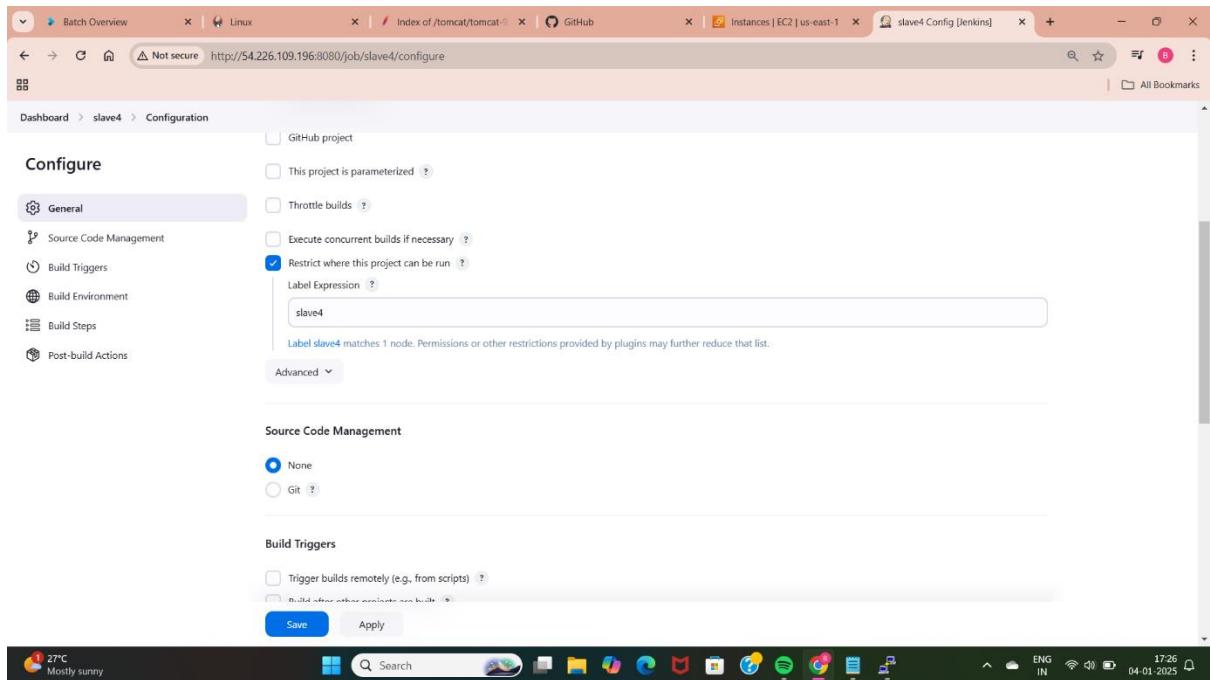
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.

The screenshot shows the Jenkins Nodes page. At the top, there are several browser tabs: 'Batch Overview', 'Linux', 'Index of /tomcat/tomcat...', 'GitHub', 'Instances | EC2 | us-east-1', and 'Nodes [Jenkins]'. The main content area is titled 'Nodes' and displays a table of nodes. The table has columns: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. It shows two entries: 'Built-In Node' (Architecture: Linux (amd64), State: In sync, Free Disk Space: 5.70 GiB, Free Swap Space: 0 B, Free Temp Space: 5.70 GiB, Response Time: 0ms) and 'slave4' (Architecture: N/A, State: N/A, Free Disk Space: N/A, Free Swap Space: N/A, Free Temp Space: N/A, Response Time: N/A). Below the table, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (Built-In Node 0/2, slave4 (launching...)). A legend at the bottom right indicates icons for S (Slave), M (Master), and L (Label). The status bar at the bottom shows '27°C Mostly sunny' and system icons.

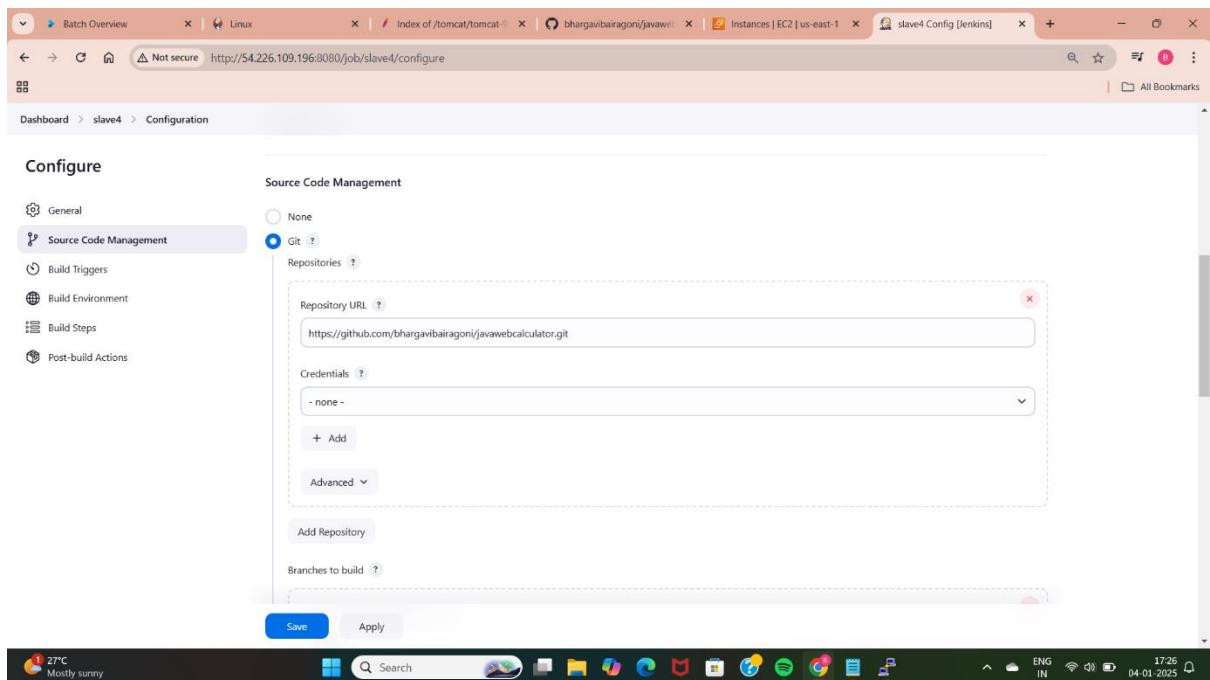
- Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.

The screenshot shows the Jenkins 'New Item' creation page. The URL in the address bar is 'http://54.226.109.196:8080/view/all/new/job'. The main title is 'New Item'. A search bar contains the name 'slave4'. Below it, a section titled 'Select an item type' lists four options: 'Freestyle project' (selected), 'Pipeline', 'Multi-configuration project', and 'Folder'. Each option has a brief description. At the bottom is an 'OK' button. The status bar at the bottom shows '27°C Mostly sunny' and system icons.

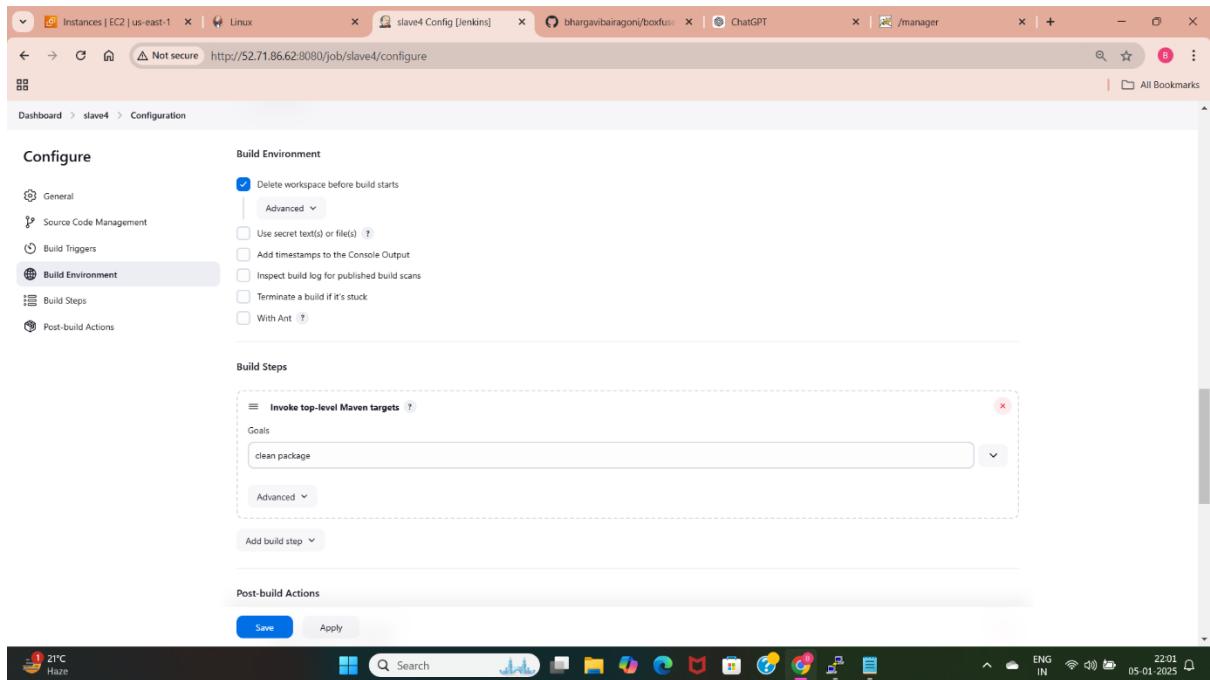
- Now am creating a new job by selecting new item at dashboard, give the name as slave4 for the new job.



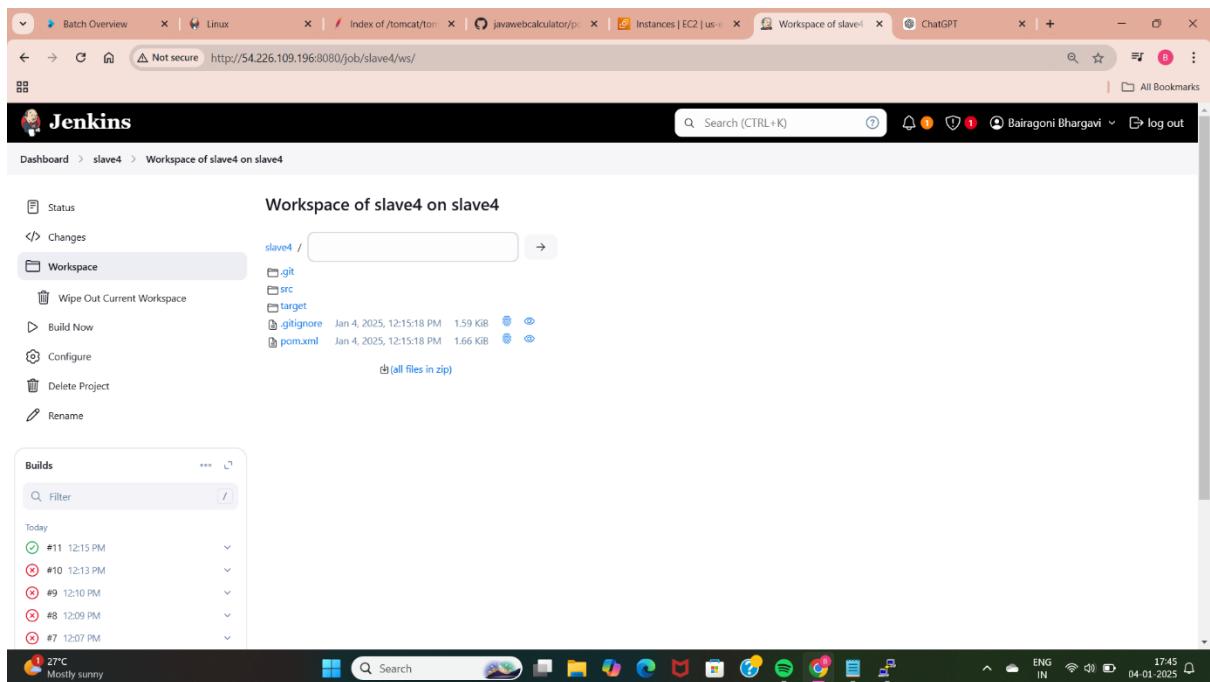
- Select the above created slave 4 by using “Restrict where this project can be run” and select the label name which is created above.



- Go to source code management and select git and give git url which is having the code related to java application and select the branch where we are having the code.



- ✓ Go to build steps and select “Invoke top level maven targets” and give goal as “clean package” to generate the target file. Save the job configuration and click Build Now. Monitor the job's output to ensure the application is deployed on Slave-1.



- ✓ Now we can build the job by clicking on “build now” option. Here build is successful so the files are deployed in Jenkins workspace.

```

ec2-user@ip-172-31-18-230:~/apache-tomcat-9.0.98/webapps/manager
plexus-classworlds.noarch 0:2.4.2-8.amzn2
plexus-containers-component-annotations.noarch 0:1.5.5-14.amzn2
plexus-interactivity.noarch 0:1.0-0.14.alphab.amzn2
plexus-sec-dispatcher.noarch 0:1.4-13.amzn2
qdox.noarch 0:1.12.1-10.amzn2
slf4j-api.noarch 0:2.3.0-11.amzn2
slf4j-log4j12.noarch 0:1.7.4-1.amzn2
xalan-j2.noarch 0:2.7.1-23.1.amzn2
xerces-j2.noarch 0:2.11.0-17.amzn2.0.2
xml-commons-resolve.noarch 0:1.2-15.amzn2

Complete!
[ec2-user@slave4 ~]$ mvn -v
Apache Maven 3.0.5 (Red Hat 3.0.5-17)
Maven home: /usr/share/maven
Java version: 17.0.1, vendor: Amazon.com Inc.
Java home: /usr/lib/jvm/java-17-amazon-corretto.x86_64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.10.230-223.885.amzn2.x86_64", arch: "amd64", family: "unix"
[ec2-user@slave4 ~]$ wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.98/bin/apache-tomcat-9.0.98.tar.gz
--2025-01-04 12:16:17-- https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.98/bin/apache-tomcat-9.0.98.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12760610 [B] (application/x-gzip)
Saving to: 'apache-tomcat-9.0.98.tar.gz'

100%[=====] 12,760,610 --.-K/s in 0.1s

2025-01-04 12:16:17 (101 MB/s) - 'apache-tomcat-9.0.98.tar.gz' saved [12760610/12760610]

[ec2-user@slave4 ~]$ ls
total 13828
-rw-r--r-- 1 ec2-user ec2-user 12760610 Dec 5 20:00 apache-tomcat-9.0.98.tar.gz
drwxr-xr-x 4 ec2-user ec2-user 34 Jan 4 11:55 bin
-rw-r--r-- 1 ec2-user ec2-user 1393083 Jan 4 11:55 remoting.jar
drwxrwxr-x 3 ec2-user ec2-user 20 Jan 4 12:15 workspace
[ec2-user@slave4 ~]$ tar -xvf apache-tomcat-9.0.98.tar.gz
apache-tomcat-9.0.98/conf/
apache-tomcat-9.0.98/conf/catalina.policy
apache-tomcat-9.0.98/conf/catalina.properties
apache-tomcat-9.0.98/conf/context.xml
apache-tomcat-9.0.98/conf/aspic-providers.xml
apache-tomcat-9.0.98/conf/aspic-providers.xsd
apache-tomcat-9.0.98/conf/aspic.xsd
apache-tomcat-9.0.98/conf/beans.properties
apache-tomcat-9.0.98/conf/server.xml
apache-tomcat-9.0.98/conf/tomcat-users.xml
apache-tomcat-9.0.98/conf/tomcat-users.xsd
apache-tomcat-9.0.98/conf/web.xml

```

- Install tomcat on slave4 server. For that go to [dlcdn.apache.org.in](https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.98/bin/apache-tomcat-9.0.98.tar.gz) select tomcat and copy the tomcat url and download it using <wget tomcat-url>.
- Extract the file using <tar -xvf apache-tomcat-9.0.98.tar.gz>.

```

ec2-user@ip-172-31-18-230:~/apache-tomcat-9.0.98/webapps/manager/META-INF
Total 132
drwxr-x--- 2 ec2-user ec2-user 4096 Jan 4 12:16 bin
-rw-r----- 1 ec2-user ec2-user 20913 Dec 5 19:50 BUILDING.txt
drwxr-x--- 3 ec2-user ec2-user 6144 Jan 4 12:16 conf
-rw-r----- 1 ec2-user ec2-user 6144 Jan 4 12:16 COPYING
drwxr-x--- 2 ec2-user ec2-user 4096 Jan 4 12:16 lib
-rw-r----- 1 ec2-user ec2-user 57092 Dec 5 19:50 LICENSE
drwxr-x--- 2 ec2-user ec2-user 197 Jan 4 12:16 logs
-rw-r----- 1 ec2-user ec2-user 2333 Dec 5 19:50 NOTICE
-rw-r----- 1 ec2-user ec2-user 3283 Dec 5 19:50 README.md
-rw-r----- 1 ec2-user ec2-user 6901 Dec 5 19:50 RELEASE-NOTES
-rw-r----- 1 ec2-user ec2-user 16538 Dec 5 19:50 RUNNING.txt
drwxr-x--- 2 ec2-user ec2-user 30 Jan 4 12:16 temp
drwxr-x--- 2 ec2-user ec2-user 81 Dec 5 19:50 webapps
drwxr-x--- 3 ec2-user ec2-user 22 Jan 4 12:16 work
[ec2-user@slave4 apache-tomcat-9.0.98]$ cd webapps/
webapps/ work/
[ec2-user@slave4 apache-tomcat-9.0.98]$ cd webapps/
[ec2-user@slave4 webapps]$ ls
total 4
drwxr-x--- 16 ec2-user ec2-user 4096 Jan 4 12:16 docs
drwxr-x--- 7 ec2-user ec2-user 99 Jan 4 12:16 examples
drwxr-x--- 6 ec2-user ec2-user 79 Jan 4 12:16 host-manager
drwxr-x--- 4 ec2-user ec2-user 114 Jan 4 12:16 manager
drwxr-x--- 3 ec2-user ec2-user 223 Jan 4 12:16 ROOT
[ec2-user@slave4 webapps]$ cd manager/
[ec2-user@slave4 manager]$ ll
bash: ll: command not found
[ec2-user@slave4 manager]$ LL
bash: LL: command not found
[ec2-user@slave4 manager]$ 
[ec2-user@slave4 manager]$ 
[ec2-user@slave4 manager]$ ll
total 20
drwxr-x--- 2 ec2-user ec2-user 25 Jan 4 12:16 css
drwxr-x--- 2 ec2-user ec2-user 44 Jan 4 12:16 images
-rw-r----- 1 ec2-user ec2-user 913 Dec 5 19:50 index.jsp
drwxr-x--- 2 ec2-user ec2-user 25 Jan 4 12:16 META-INF
-rw-r----- 1 ec2-user ec2-user 4374 Dec 5 19:50 status.xsd
drwxr-x--- 3 ec2-user ec2-user 32 Jan 4 12:16 WEB-INF
-rw-r----- 1 ec2-user ec2-user 4709 Dec 5 19:50 xform.xsl
[ec2-user@slave4 manager]$ cd META-INF/
[ec2-user@slave4 META-INF]$ ll
total 4
-rw-r----- 1 ec2-user ec2-user 1352 Dec 5 19:50 context.xml
[ec2-user@slave4 META-INF]$ 

```

- Go to apache-tomcat.9.0.98/webapps/manager/META-INF/context.xml and delete two lines. The two lines typically removed are related to restrictions on access to the Manager App.

```

ec2-user@ip-172-31-18-230:~/apache-tomcat-9.0.98/webapps/manager
apache-tomcat-9.0.98/bin/version.sh
[ec2-user@slave4 ~]$ ll
total 13828
drwxr-xr-x 9 ec2-user ec2-user 220 Jan 4 12:16 apache-tomcat-9.0.98
-rw-r--r-- 1 ec2-user ec2-user 1276061 Dec 5 20:00 apache-tomcat-9.0.98.tar.gz
drwxr-xr-x 1 ec2-user ec2-user 24 Jan 4 11:55 remoting
drwxr-xr-x 1 ec2-user ec2-user 1393083 Jan 4 11:55 remoting.jar
drwxr-xr-x 3 ec2-user ec2-user 20 Jan 4 12:16 workspace
[ec2-user@slave4 ~]$ cd apache-tomcat-9.0.98/
[ec2-user@slave4 apache-tomcat-9.0.98]$ ll
total 132
drwxr-x--- 2 ec2-user ec2-user 4096 Jan 4 12:16 bin
-rw-r----- 1 ec2-user ec2-user 20913 Dec 5 19:50 BUILDING.txt
drwxr-x--- 2 ec2-user ec2-user 238 Dec 5 19:50 conf
-rw-r----- 1 ec2-user ec2-user 6166 Dec 5 19:50 CONTRIBUTING.md
drwxr-x--- 2 ec2-user ec2-user 4096 Dec 5 19:50 lib
drwxr-x--- 1 ec2-user ec2-user 57092 Dec 5 19:50 LICENSE
drwxr-x--- 2 ec2-user ec2-user 6 Dec 5 19:50 logs
-rw-r----- 1 ec2-user ec2-user 2333 Dec 5 19:50 NOTICE
-rw-r----- 1 ec2-user ec2-user 3283 Dec 5 19:50 README.md
-rw-r----- 1 ec2-user ec2-user 6901 Dec 5 19:50 RELEASE-NOTES
-rw-r----- 1 ec2-user ec2-user 16538 Dec 5 19:50 RUNNING.txt
drwxr-x--- 2 ec2-user ec2-user 30 Jan 4 12:16 temp
drwxr-x--- 7 ec2-user ec2-user 81 Dec 5 19:50 webapps
drwxr-x--- 2 ec2-user ec2-user 6 Dec 5 19:50 work
[ec2-user@slave4 apache-tomcat-9.0.98]$ cd bin/
[ec2-user@slave4 bin]$ ll
total 816
-rw-r----- 1 ec2-user ec2-user 35459 Dec 5 19:50 bootstrap.jar
-rw-r----- 1 ec2-user ec2-user 16856 Dec 5 19:50 catalina.bat
-rw-r----- 1 ec2-user ec2-user 25323 Dec 5 19:50 catalina.sh
-rw-r----- 1 ec2-user ec2-user 1664 Dec 5 19:50 catalina-tasks.xml
-rw-r----- 1 ec2-user ec2-user 2123 Dec 5 19:50 ciphers.bat
-rw-r----- 1 ec2-user ec2-user 2123 Dec 5 19:50 ciphers.sh
-rw-r----- 1 ec2-user ec2-user 25834 Dec 5 19:50 commons-daemon.jar
-rw-r----- 1 ec2-user ec2-user 214459 Dec 5 19:50 commons-daemon-native.tar.gz
-rw-r----- 1 ec2-user ec2-user 2040 Dec 5 19:50 configtest.bat
-rw-r----- 1 ec2-user ec2-user 1922 Dec 5 19:50 configtest.sh
-rw-r----- 1 ec2-user ec2-user 9100 Dec 5 19:50 daemon.sh
-rw-r----- 1 ec2-user ec2-user 2091 Dec 5 19:50 digest.bat
-rw-r----- 1 ec2-user ec2-user 1965 Dec 5 19:50 digest.sh
-rw-r----- 1 ec2-user ec2-user 360 Dec 5 19:50 makebase.bat
-rw-r----- 1 ec2-user ec2-user 360 Dec 5 19:50 makebase.sh
-rw-r----- 1 ec2-user ec2-user 3014 Dec 5 19:50 setclasspath.bat
-rw-r----- 1 ec2-user ec2-user 4317 Dec 5 19:50 setclasspath.sh
-rw-r----- 1 ec2-user ec2-user 2020 Dec 5 19:50 shutdown.bat
-rw-r----- 1 ec2-user ec2-user 1902 Dec 5 19:50 shutdown.sh
-rw-r----- 1 ec2-user ec2-user 2022 Dec 5 19:50 startup.bat
-rw-r----- 1 ec2-user ec2-user 1904 Dec 5 19:50 startup.sh

```

- ✓ After making changes, restart Tomcat by moving into apache-tomcat.9.0.98/bin to apply changes.
- ✓ First shutdown the tomcat using sh shutdown.sh and then start using sh startup.sh.
- ✓ Now copy the public ip address where we installed tomcat paste it in browser as <slavepublicip:8080>.
- ✓ Select the manager and access it. To access it we need to create the user and add roles.

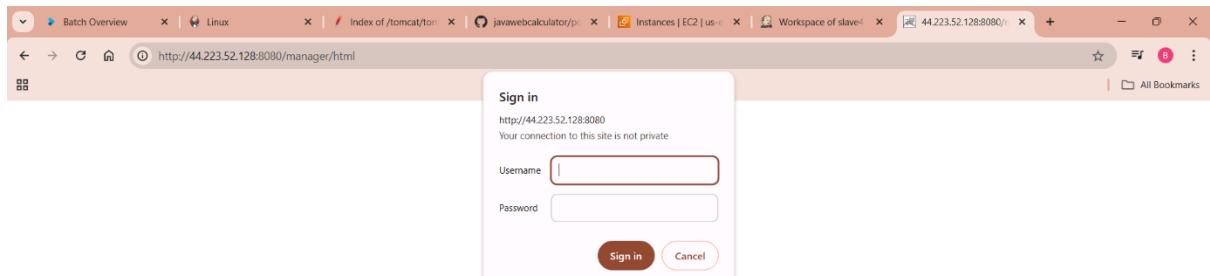
```

ec2-user@ip-172-31-18-230:~/apache-tomcat-9.0.98/conf

<tomcat-users>
    <!-- Default, no user is included in the "manager-gui" role required
        to operate the "manager-gui" web application. If you wish to use this app,
        you must define such a user - add username and password here arbitrary. -->
    <!-- Built-in default manager roles:
        * manager-script - allows access to the HTML GUI and the status pages
        * manager-jmx - allows access to the JMX API and the status pages
        * manager-status - allows access to the JMX proxy and the status pages
        * manager-gui - allows access to the status pages only
    -->
    <!-- The users below are wrapped in a comment and are therefore ignored. If you
        wish to configure one or more of these users for use with the manager web
        application, do not forget to remove the <!-- ... --> that surrounds them. You
        will also need to set the password to something appropriate. -->
    <!--
        <user username="admin" password="admin-is-changed" roles="manager-gui"/>
        <user username="tomcat" password="tomcat-is-changed" roles="manager-script"/>
    -->
    <!--
        The sample user and role entries below are intended for use with the
        examples web application. They are wrapped in a comment and thus are ignored
        when reading this file. If you wish to configure these users for use with the
        examples web application, do not forget to remove the <!-- ... --> that surrounds
        them. You will also need to set the password to something appropriate.
    -->
    <!--
        <role rolename="tomcat"/>
        <role rolename="status"/>
        <user username="tomcat" password="tomcat-is-changed" roles="tomcat,status"/>
        <user username="admin" password="admin-is-changed" roles="manager-gui,tomcat,status"/>
        <user username="root" password="root-is-changed" roles="root,tomcat,status"/>
    -->
    <!--
        <role rolename="manager-gui"/>
        <role rolename="manager-script"/>
        <user username="tomcat" password="admin123" roles="manager-gui,manager-script"/>
    -->
</tomcat-users>
-- INSERT --

```

- ✓ Now go to apache-tomcat.9.0.98/conf and add a user with manager-gui and/or manager-script roles to enable access.



- ✓ Sign into tomcat manager using the above created username and password.

Install	Name	Released
<input type="checkbox"/>	Deploy to container 1.16 Artifact Uploaders	4 yr 2 mo ago
<input type="checkbox"/>	Docker Pipeline 580.vcd3c340686b_54 pipeline DevOps Deployment docker	7 mo 17 days ago
<input type="checkbox"/>	Artifactory 4.0.8 pipeline	5 mo 27 days ago
<input type="checkbox"/>	Ansible 403.v0dca_dcb_b_502 pipeline External Site/Tool Integrations DevOps Build Tools Deployment	6 mo 20 days ago



- ✓ To deploy java application into tomcat we need a plugin i.e “deploy to container” install it by moving to manage Jenkins>plugins>available plugins.

The screenshot shows the Tomcat Web Application Manager interface. At the top, there is a header bar with tabs for "Batch Overview", "Linux", "Index of /tomcat/", "javawebcalculator/p...", "Instances | EC2 | us...", "slave4 [Jenkins]", and "/manager". Below the header, the URL is http://44.223.52.128:8080/manager/html. The main content area is titled "Tomcat Web Application Manager". It displays a table of deployed applications:

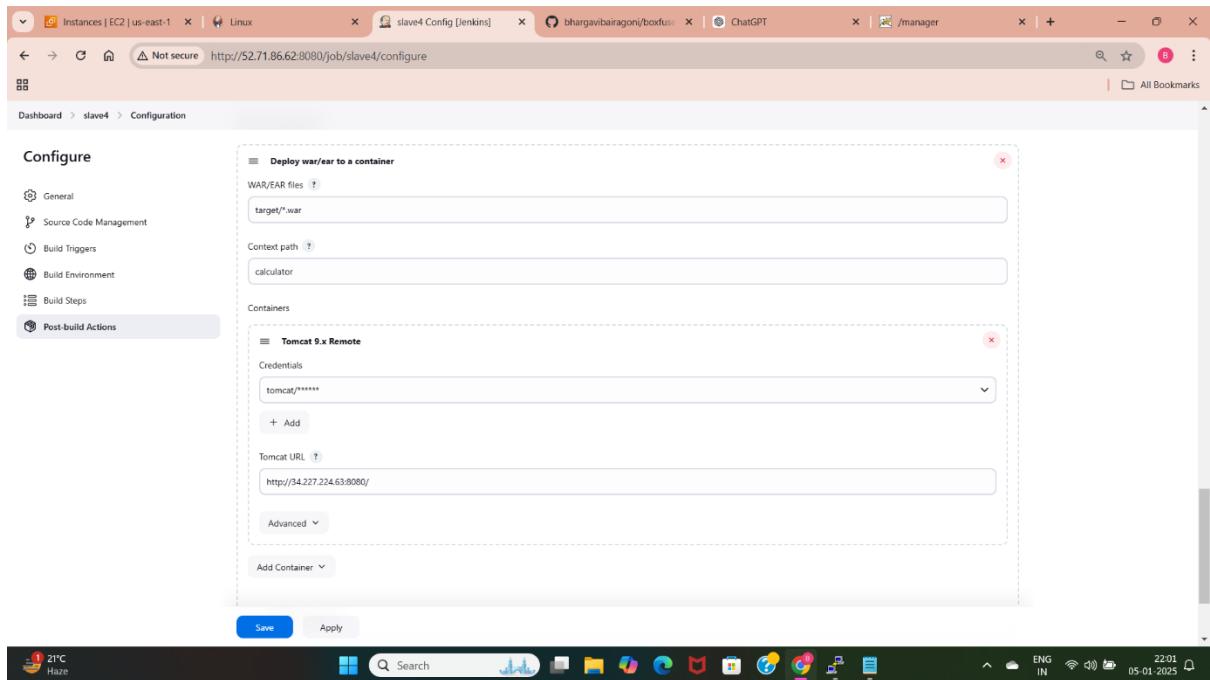
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions with idle ≥ 30 minutes</a>
/docs	None specified	Tomcat Documentation	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions with idle ≥ 30 minutes</a>
/examples	None specified	Servlet and JSP Examples	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions with idle ≥ 30 minutes</a>
/host-manager	None specified	Tomcat Host Manager Application	true	0	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions with idle ≥ 30 minutes</a>
/manager	None specified	Tomcat Manager Application	true	1	<a href="#">Start</a> <a href="#">Stop</a> <a href="#">Reload</a> <a href="#">Undeploy</a> <a href="#">Expire sessions with idle ≥ 30 minutes</a>

Below the table, there is a "Deploy" section with a message: "So three, why to copy three lines? Is pissed. You can change. Manager. Hyphen, GUI". The system tray at the bottom shows the date as 04-01-2025.

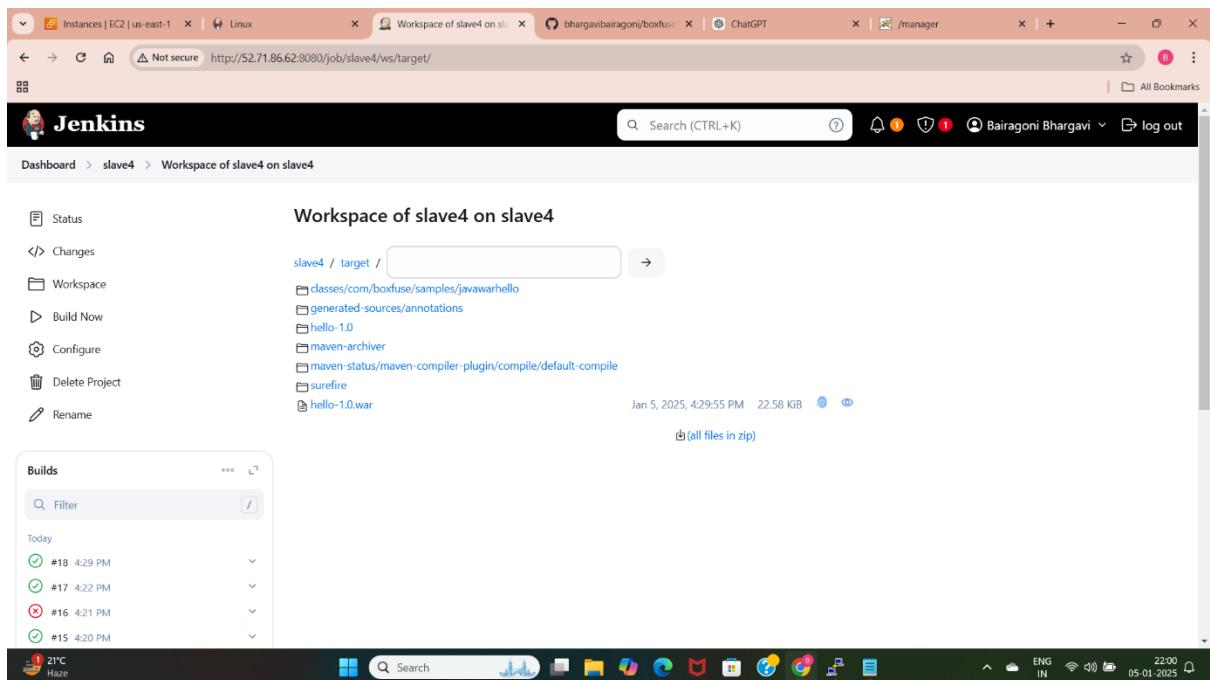
- ✓ This is the manager page we have logged into it using username and password.

The screenshot shows the Jenkins Credentials Provider configuration page. The URL is https://54.226.109.196:8080/job/slave4/configure. The left sidebar shows "Configure" with options: General, Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main panel is titled "Jenkins Credentials Provider: Jenkins". It contains fields for "Username with password", "Scope" (set to "Global (Jenkins, nodes, items, all child items, etc.)"), "Username" (set to "tomcat"), "Password" (set to "\*\*\*\*\*"), "ID" (set to "Tomcat"), and "Description". At the bottom are "Save" and "Apply" buttons. The system tray at the bottom shows the date as 04-01-2025.

- ✓ We need to create credentials for tomcat. Go to credentials in manage Jenkins and select “username with password” and give username as tomcat which is created above and give above created password, id and save.



- ✓ Now go to “Post-build actions” and select deploy war/ear to a container and give the war file which is generated using maven. War file is under target folder so give name as target/\*.war.
- ✓ Give any context path which is displayed on tomcat page to access the java application. I have given the context path as “calculator”.
- ✓ Select add containers “tomcat 9x remote” give created credentials and provide the tomcat url so we can deploy the java application there.



- After a successful build, Jenkins places the output files (e.g., compiled code, artifacts, logs) into the workspace directory of the job.

**Tomcat Web Application Manager**

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/calculator	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

**Deploy**  
Deploy directory or WAR file located on server

- In tomcat page we can find the calculator application deployed using Jenkins.

**Congratulations!**

**boxfuse**

You have successfully launched your Instance!

This instance is running on **null** and has the id **null**.  
It is based on the Image **null** generated from **null**.

**Now it's your turn!**

Check out [this app](#) from GitHub, modify it, and give it version 2.  
You are now ready to **fire up** and **deploy it with zero downtime** using the commands you already know.

Alternatively you can go back to the [Boxfuse Console](#),  
and simply create your own.

If you need any help the [documentation](#) is there for you,  
or simply shoot us an email at [support@boxfuse.com](mailto:support@boxfuse.com)

Say goodbye to snowflake servers.

Enjoy Boxfuse!

- This is the java application which is deployed using Jenkins and tomcat.

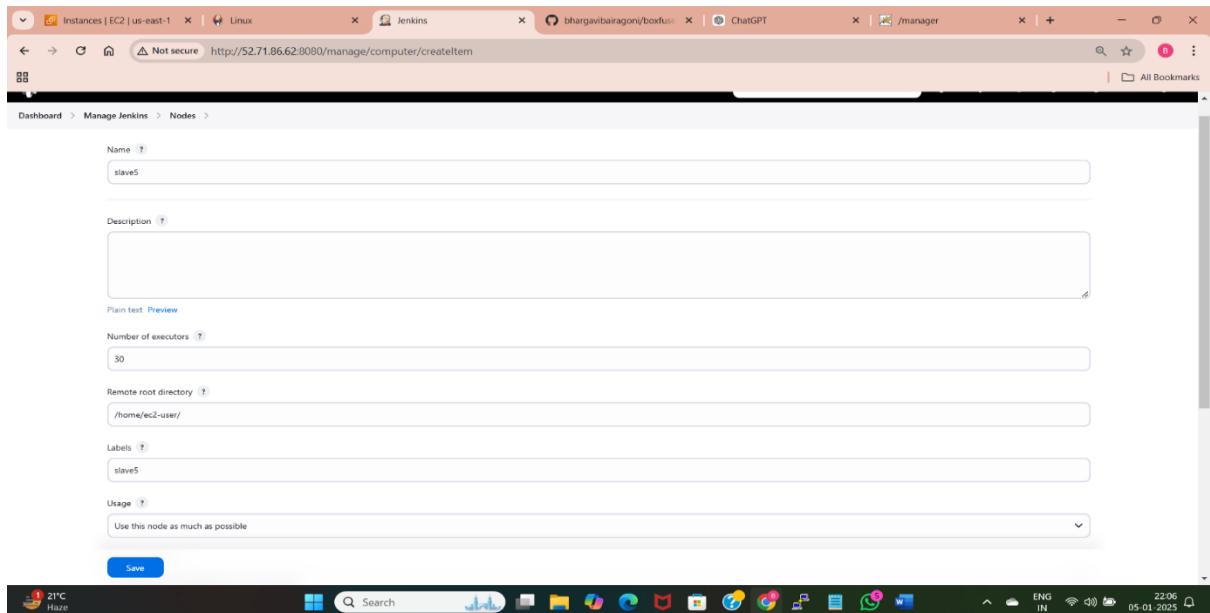
## Slave-5: Deployment Using Bash Script

The screenshot shows the AWS EC2 Instances page. A green banner at the top indicates "Successfully initiated termination (deletion) of i-00c203cf77adf3064". Below this, a table lists four instances: slave3, slave4, master, and slave5. Slave5 is selected and has a checkmark next to its name. The table includes columns for Name, Instance ID, Instance state, Status check, Alarm status, Availability zone, Public IPv4 address, Elastic IP, and IPv6. Slave5's status is "Running". The table header shows "Instances (1/4) Info" and "Actions". On the left sidebar, there are links for Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, and Elastic IPs.

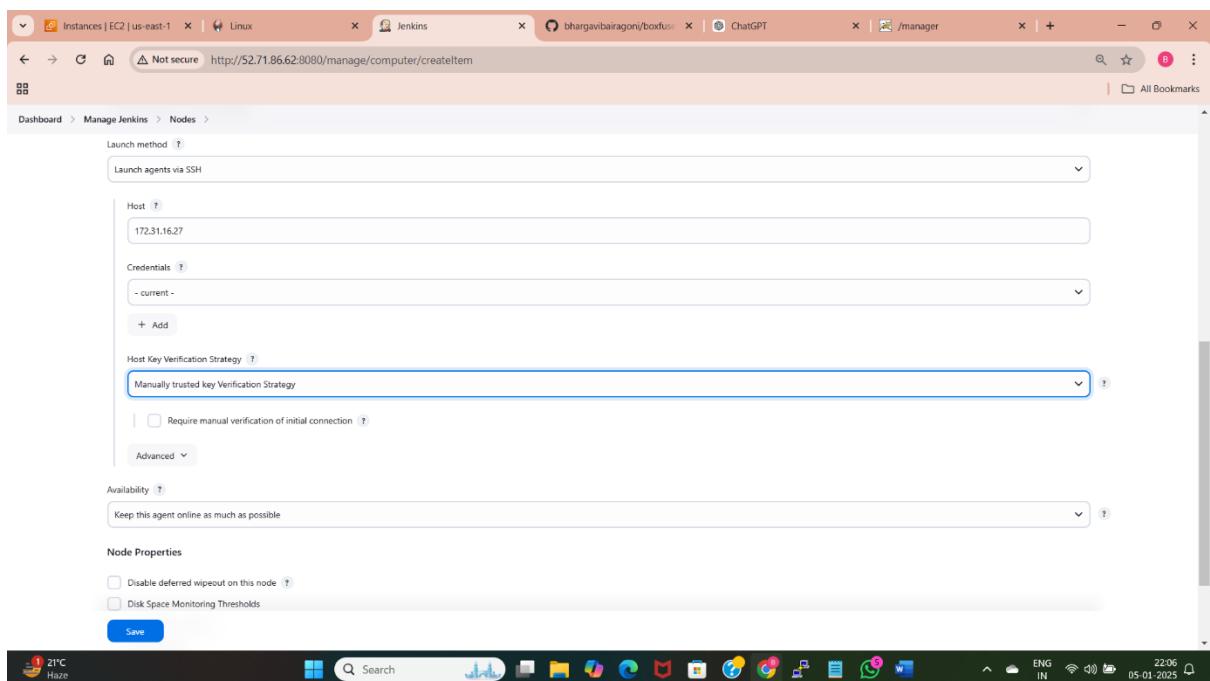
- ✓ I have launched a slave5 server to deploy java application using bash script.

The screenshot shows the Jenkins "New node" configuration page. The "Node name" field is set to "slave5". Under the "Type" section, the "Permanent Agent" option is selected. A tooltip explains that this adds a plain, permanent agent to Jenkins. Below this, there is a "Copy Existing Node" link. At the bottom, a blue "Create" button is visible. The page also includes a "Search (CTRL+K)" bar and navigation links for "Dashboard", "Manage Jenkins", "Nodes", and "New node". The bottom of the screen shows a Windows taskbar with various icons and system status information.

- ✓ Go to Manage Jenkins > Manage Nodes > New Node, give name for the node and select permanent agent and click "ok".



- ✓ Next we need to create a node in Jenkins using the above slave5 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave5”, select the number of executors(no. of builds) according to requirement.
- ✓ Give the remote root directory of the slave5 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.



- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.

The screenshot shows the Jenkins 'Nodes' page. At the top, there are tabs for 'Instances | EC2 | us-east-1', 'Linux', 'Nodes [Jenkins]', 'bhargavibaibagon/boxfus...', 'ChatGPT', and '/manager'. The URL in the address bar is <http://52.71.86.62:8080/manage/computer/>. The main content area is titled 'Nodes' with a sub-section 'Nodes'. It lists three nodes:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.43 GiB	0 B	5.43 GiB	0ms
	slave4	Linux (amd64)	In sync	5.73 GiB	0 B	5.73 GiB	41ms
	slave5		N/A	N/A	N/A	N/A	N/A

Below the table, it says 'Data obtained' and shows times: 6 min 59 sec for all columns. There are icons for S, M, and L at the bottom left. A legend is at the bottom right.

- ✓ Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful and here slave5 is connected to master successfully.

The screenshot shows the Jenkins 'New Item' creation page. The URL in the address bar is <http://52.71.86.62:8080/view/all/newJob>. The main content area is titled 'New Item'.

Enter an item name: slave5

Select an item type:

- Freestyle project**: Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

- ✓ Now am creating a new job by selecting new item at dashboard, give the name as slave5 for the new job.

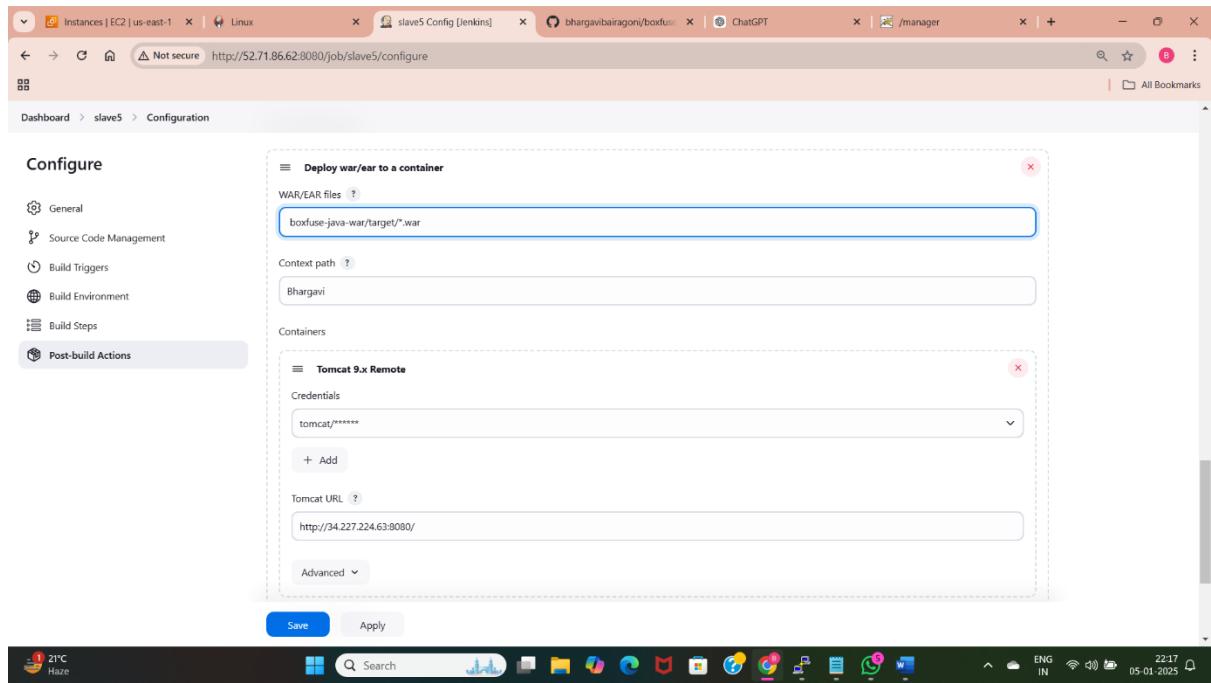
The screenshot shows the Jenkins configuration page for the 'slave5' job. In the 'General' section, the 'Restrict where this project can be run' checkbox is checked, and the 'Label Expression' field contains 'slave5'. Other options like 'Discard old builds', 'GitHub project', and 'Throttle builds' are unchecked.

- ✓ Select the above created slave5 by using “Restrict where this project can be run” and select the label name which is created above.

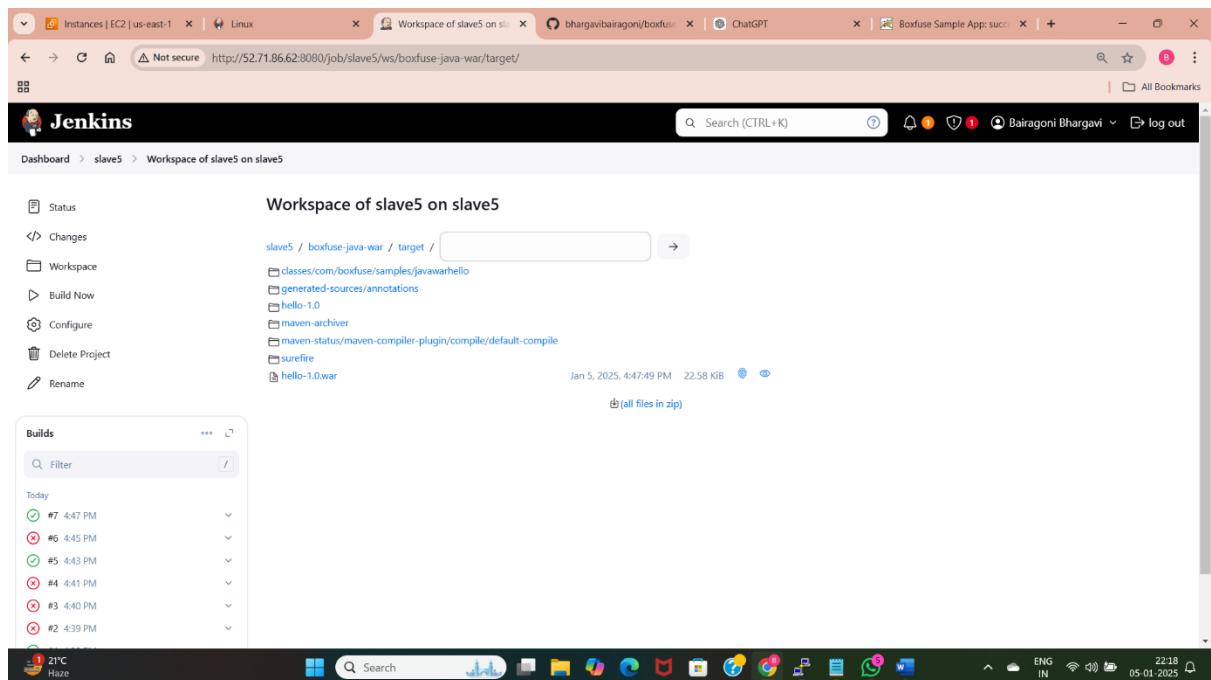
The screenshot shows the Jenkins configuration page for the 'slave5' job. In the 'Build Steps' section, there is one step named 'Execute shell' with the following command:

```
sudo yum install git -y  
git clone https://github.com/bhargavibairagi/boxfuse-java-war.git  
sudo yum install maven -y  
cd boxfuse-java-war  
mvn clean package
```

- ✓ Now go to build steps and select execute shell write commands to install git, and clone the github url which is having java application.
- ✓ Next install maven in that directory and move to the repository where we have java files.
- ✓ Now generate war file from the application using <mvn clean package>.



- ✓ Now go to “Post-build actions” and select deploy war/ear to a container and give the war file which is generated using maven. War file is under target folder so give name as target/\*.war.
- ✓ Give any context path which is displayed on tomcat page to access the java application. I have given the context path as “Bhargavi”.
- ✓ Select add containers “tomcat 9x remote” give created credentials and provide the tomcat url so we can deploy the java application there.



- ✓ Now click on build now option so that we can deploy the application into tomcat and it can also performs the tasks whatever mentioned in the execute shell.

**Tomcat Web Application Manager**

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
Bhargavi	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
calculator	None specified		true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

**Deploy**  
Deploy directory or WAR file located on server

- ✓ In tomcat web page we can see the application path “Bhargavi” which is having the java application.

**Congratulations!**

**boxfuse**

You have successfully launched your Instance!

This Instance is running on **null** and has the id **null**.  
It is based on the Image **null** generated from **null**.

**Now it's your turn!**

Check out [this app](#) from GitHub, modify it, and give it version 2.  
You are now ready to fuse and deploy it with zero downtime using the commands you already know.

Alternatively you can go back to the [Boxfuse Console](#),  
and simply create your own.

If you need any help the [documentation](#) is there for you,  
or simply shoot us an email at [support@boxfuse.com](mailto:support@boxfuse.com).

Say goodbye to snowflake servers.

- ✓ We can access the application by clicking on the path in tomcat and this is the application I have deployed.