

TEAM PROJECT

Master and Slave Configuration

The master-slave configuration is a design pattern often used in distributed systems, databases, and replication setups. In this architecture:

Master:

- Acts as the central authority or main node.
- Handles the primary operations such as data writing, updates, or control over the connected slaves.
- Serves as the single source of truth in the system.

Slave:

- Receives data, commands, or replication from the master.
- Acts as a replica or secondary node that synchronizes with the master.
- Typically handles read-only operations or specific delegated tasks.

This configuration ensures scalability, load balancing, and fault tolerance. For example:

- In databases, the master is responsible for writes, and slaves handle reads.
- In Jenkins, the master manages jobs, and slaves execute them on distributed systems.

Different Account Creation

- Here, the master and slave nodes are created in separate accounts.
- Use Case: This is common in multi-account setups, often used for added security, resource segregation, or cross-organization collaboration.
- Advantages:
 - Improved security, as access is isolated between accounts.
 - Better control for resource segregation across teams or departments.
- Disadvantages:
 - Requires configuring cross-account access (e.g., AWS IAM roles for account communication).
 - Slightly more complex to manage due to additional permissions and policies.

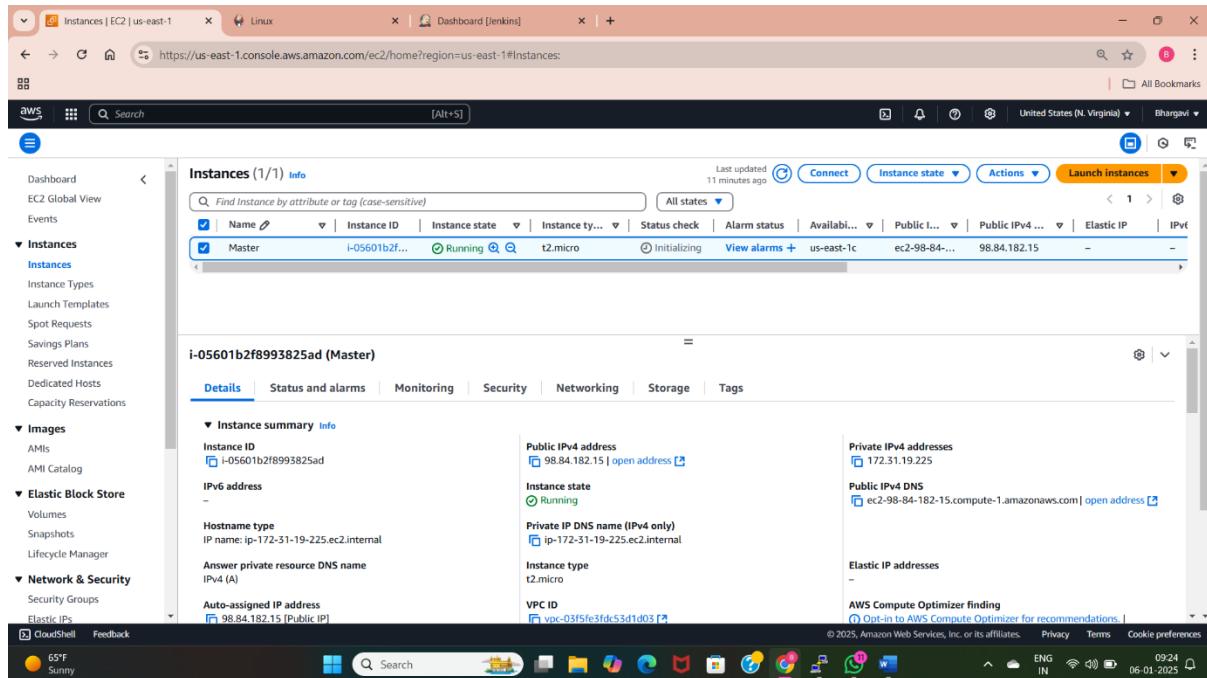
METHOD-3:

This project involves deploying WordPress across multiple slave nodes (AWS EC2 instances) using different deployment techniques. The deployments include:

- Master, Slave-5: Bash Script (Bhargavi)
- Slave-1: Kubernetes (Vamsi)
- Slave-2: Docker Compose (Vasanthi)
- Slave-3: Terraform (Balaji)
- Slave-4: Jenkins (Poll SCM, Build Periodically, Webhooks by Balaji)

MASTER INFRASTRUCTURE SETUP:

First we(first person) need to create a master, slave 5. For that first launch a master server with amazon-linux machine image with t2.micro, and allow required security group ports in inbound.



Master is created using above configurations. Now setup Jenkins in this master server. For that first we need to connect this master with terminal using the public Ip address of instance and private key(i.e key pair).

```

ec2-user@ip-172-31-19-225:~$ login as: ec2-user
Authenticating with public key "imported-openssh-key"
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-19-225 ~]$ sudo hostname master
[ec2-user@ip-172-31-19-225 ~]$ exec bash
[ec2-user@master ~]$ sudo yum install java-17-amazon-corretto -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1 will be installed
--> Processing Dependency: java-17-amazon-corretto-headless(x86-64) = 1:17.0.13+11-1.amzn2.1 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libhx11 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libhx11 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXinerama for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrender for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrandr for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXtst for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: giflib for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Running transaction check
--> Package giflib.x86_64 0:4.1.6-9.amzn2.0.2 will be installed
--> Processing Dependency: libIc.so.6() (64bit) for package: giflib-4.1.6-9.amzn2.0.2.x86_64
--> Processing Dependency: libSM.so.6() (64bit) for package: giflib-4.1.6-9.amzn2.0.2.x86_64
--> Package java-17-amazon-corretto-headless.x86_64 1:17.0.13+11-1.amzn2.1.x86_64 will be installed
--> Processing Dependency: jpegs2ps-common >= 1:1.0.0-1 for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: fontconfig for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: dejavu-sans-fonts for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: dejavu-sans-mono-fonts for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: alsalib for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: log4j-cve-2021-44228-cve-mitigations for package: 1:java-17-amazon-corretto-headless-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libX11-1.x86_64 0:1.6.7-3.amzn2.0.5 will be installed
--> Processing Dependency: libX11-common >= 1:1.6.7-3.amzn2.0.5 for package: libX11-1.6.7-3.amzn2.0.5.x86_64
--> Processing Dependency: libX11-xkbcommon >= 0.1.7-1.amzn2.0.2 will be installed
--> Processing Dependency: libXext.x86_64 0:1.7.9-1.amzn2.0.2 will be installed
--> Package libXinerda.x86_64 0:1.5.1-2.amzn2.0.3 will be installed
--> Package libXrender.x86_64 0:0.9.10-1.amzn2.0.2 will be installed

```

- ♣ Here I have changed the hostname to master using “sudo hostname master” and apply the changes using “exec bash”.
- ♣ Install java-17 in master using “sudo yum install java-17-amazon-corretto -y”.
- ♣ Install Jenkins repository and install token to login to the Jenkins.

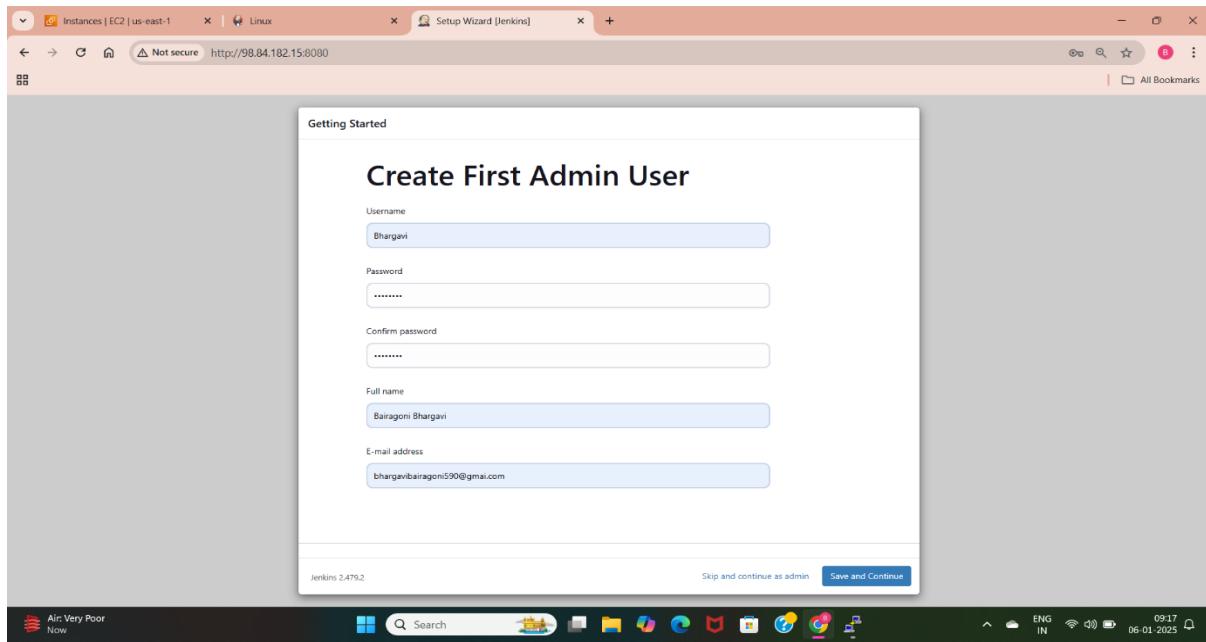
```

ec2-user@ip-172-31-19-225:~$ Install 1 Package
Total download size: 91 M
Installed size: 92 M
Is this ok [y/d/N]: y
Is this ok [y/d/N]: y
Downloaded packages:
jenkins-2.479.2-1.1.noarch.rpm
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : jenkins-2.479.2-1.1.noarch
  Verifying : jenkins-2.479.2-1.1.noarch
Installed:
  jenkins.noarch 0:2.479.2-1.1

Complete!
[ec2-user@master ~]$ sudo systemctl start jenkins
[ec2-user@master ~]$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
7e3de0d50fdc40cdab9ca5afdfcc6dd
[ec2-user@master ~]$
[ec2-user@master ~]$ ^C
[ec2-user@master ~]$ ssh-keygen
Generating RSA private key
Enter file in which to save the key (/home/ec2-user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ec2-user/.ssh/id_rsa.
Your public key has been saved in /home/ec2-user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:82XV8wAAP862g60uLN9scCjJUNqywiseVtjt//7RlAk ec2-user@master
The key's randomart image is:
+---[RSA 2048]---+
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
|          .          |
+---[SHA256]---+
[ec2-user@master ~]$ cd .ssh/
[ec2-user@master .ssh]$ sudo vi id_rsa.pub
[ec2-user@master .ssh]$

```

- ♣ Install Jenkins using “sudo yum install Jenkins -y” and start the Jenkins using “sudo systemctl start jenkins”.
- ♣ Now copy the public ip of master with Jenkins port 8080 paste in browser.
- ♣ Next it will ask for password by showing a path copy that path and use “sudo cat url” it will give password paste that password in Jenkins, save and continue.
- ♣ Now it will shows the Jenkins url, save and continue.
- ♣ Generate key pair using “ssh-keygen”, it will create public and private keys.



- Now create Jenkins user with password and provide full name and email address, save and create.
- Next we will see the Jenkins dashboard which shows new items, manage Jenkins and other options we can create jobs and nodes and install plugins.
- Next I (bhargavi) have shared the Jenkins url with my team members to create jobs and nodes in the Jenkins.

These are the jobs created in the Jenkins master and we are going to deploy the applications using these jobs.

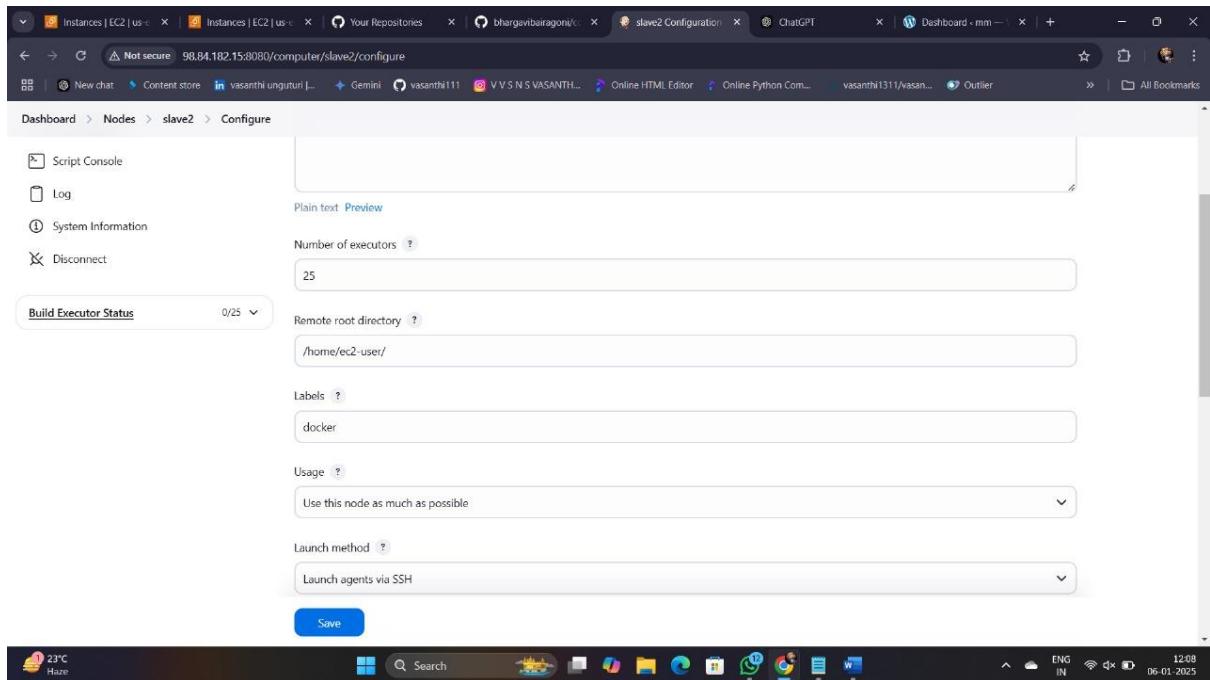
SLAVE-2 :

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, and AMI Catalog. The main content area displays a table of instances. One instance is selected: slave2, with the ID i-0bddc110d53cdeba9. The instance is listed as Running, t2.micro, with 2/2 checks passed, in us-east-1c, and has a Public IPv4 address of ec2-54-197-25. Below the table, a detailed view for the selected instance is shown, including tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under the Details tab, the Public IPv4 address is highlighted with a tooltip: "Public IPv4 address copied". The status bar at the bottom shows various icons and the date 06-01-2025.

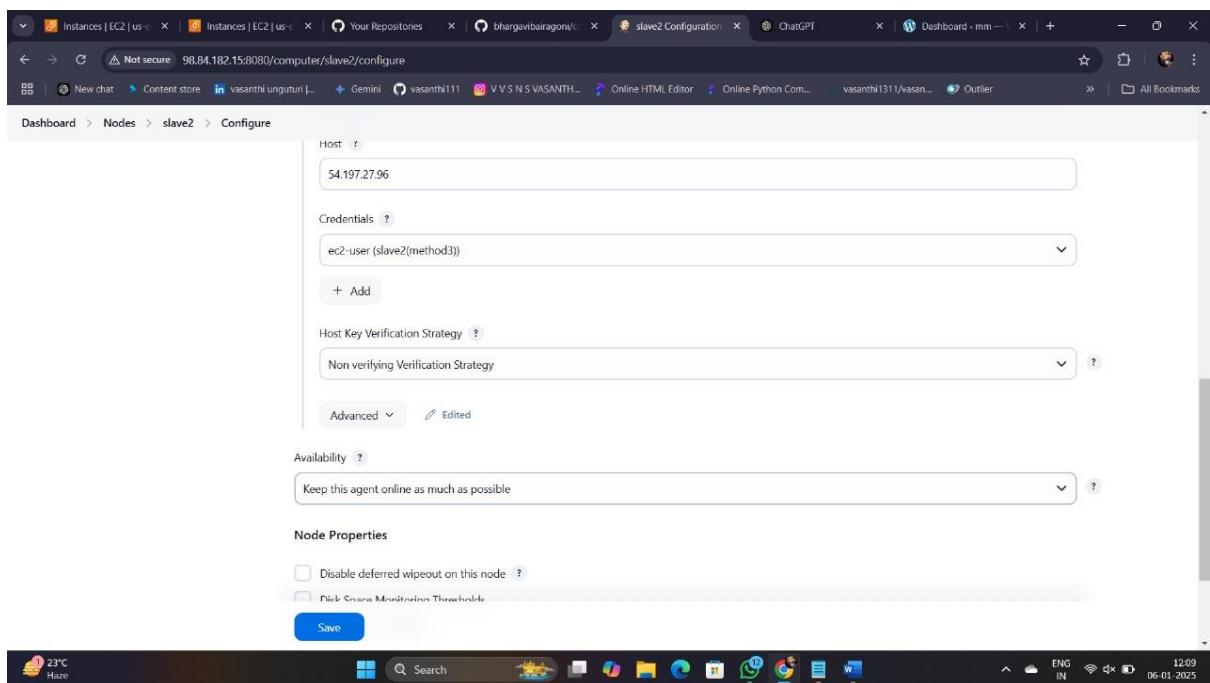
- ♣ Here vasanthi is creating a slave2 server in her aws account.
- ♣ Now she is able to access the Jenkins using the username and password of above created Jenkins credentials.

The screenshot shows the Jenkins 'New node' configuration page. At the top, the URL is http://98.84.182.15:8080/computer/new. The page title is 'New node'. It has a 'Node name' input field containing 'slave2'. Below it, a 'Type' section is expanded, showing the 'Permanent Agent' option selected. A note explains: '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.' At the bottom of the form is a 'Create' button. The status bar at the bottom shows the date 06-01-2025.

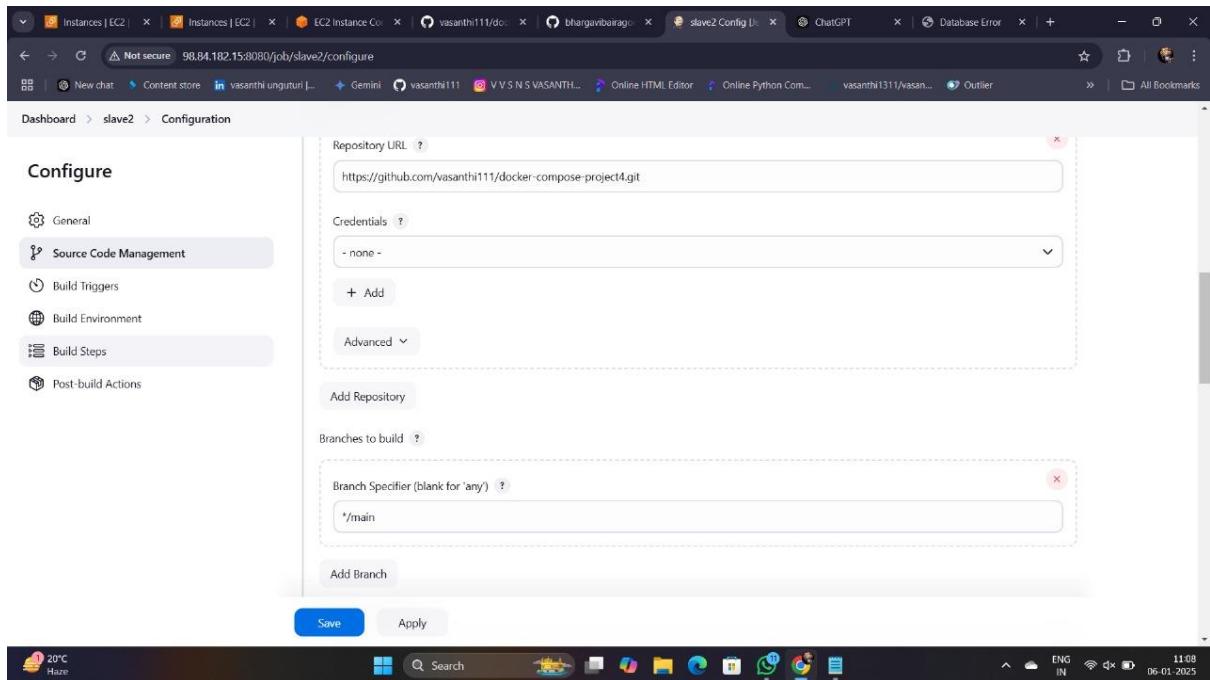
- ♣ Now she is creating slave2 in the Jenkins. For that go to manage Jenkins select nodes and give name for node.



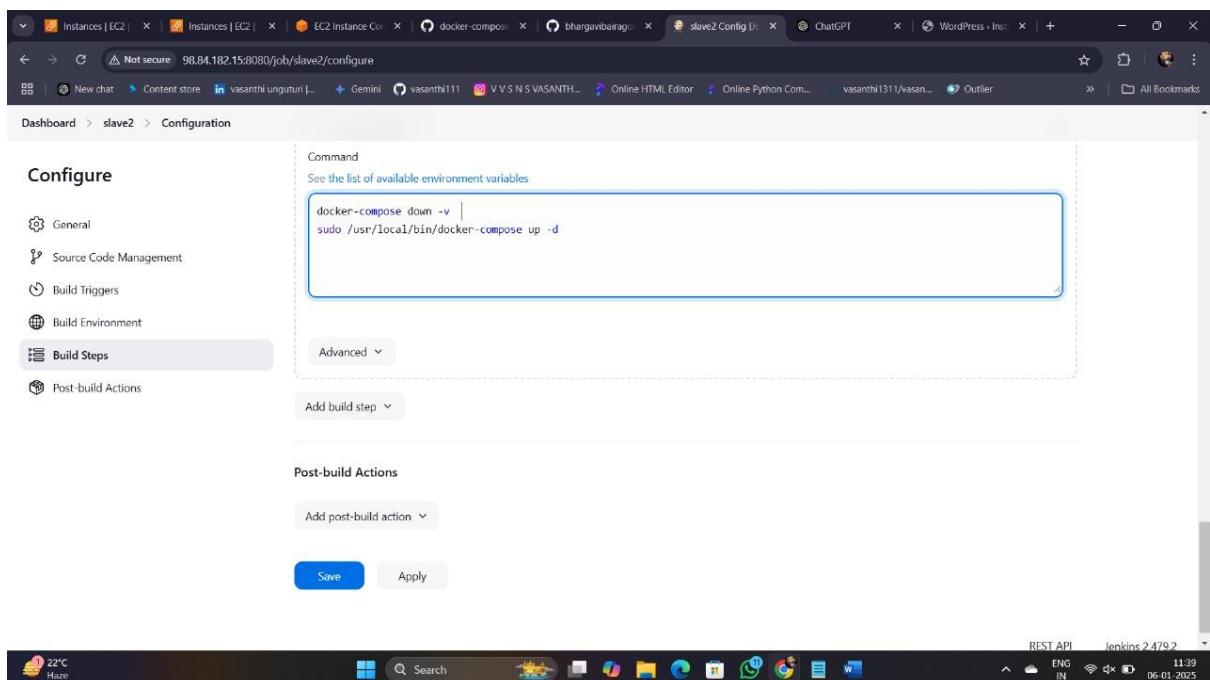
- Now select number of executors for slave 2 node, give remote directory path of slave 2 node and name for lable, select usage and launch method as launch via ssh.



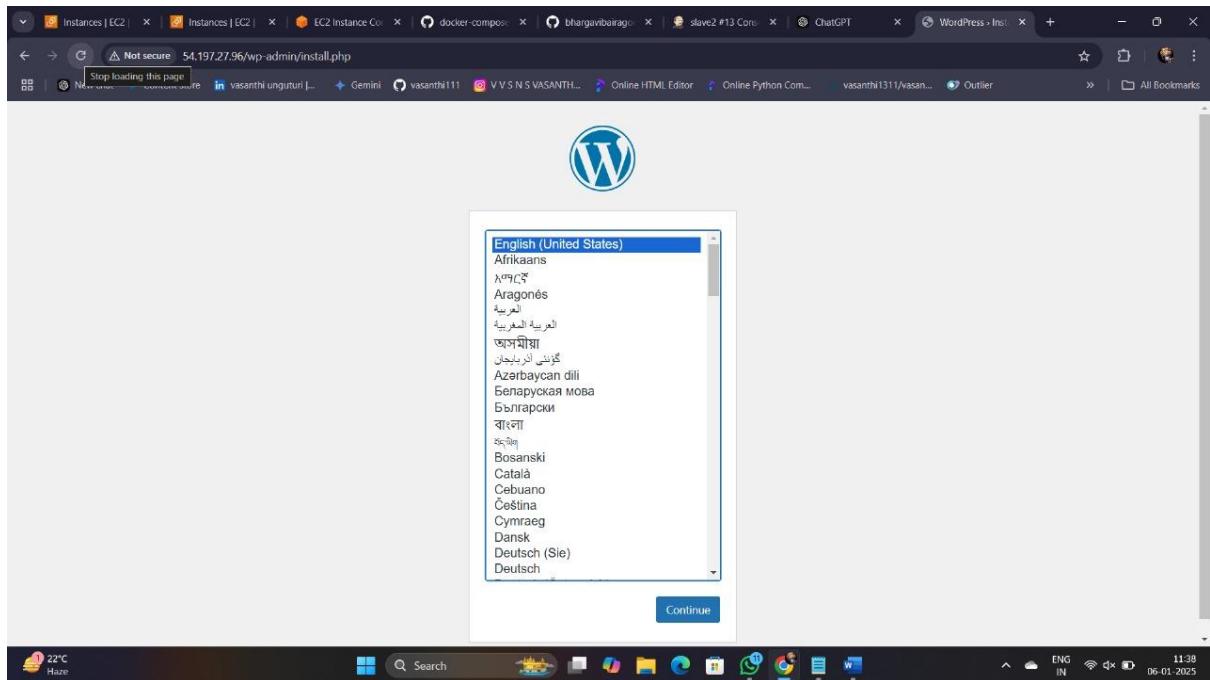
- Here selecting the host as slave2 server public ip because we are using two different accounts so if we take private ip it wont get internet from masters VPC and select verification as non verifying strategy because we are using different key pairs for master and slaves.
- Next create a job in Jenkins.
- Install git, docker and docker-compose in slave server.



- Here a job is created with the name slave2. Taking the github url to deploy wordpress using compose file, select the branch where we are having the code.



- In execute shell run the command to deploy the docker-compose file using “docker-compose up -d”.
- Now save and click on build now option it will show the result of this build as success if there is no error and it shows failure if there are any configurations are missing.

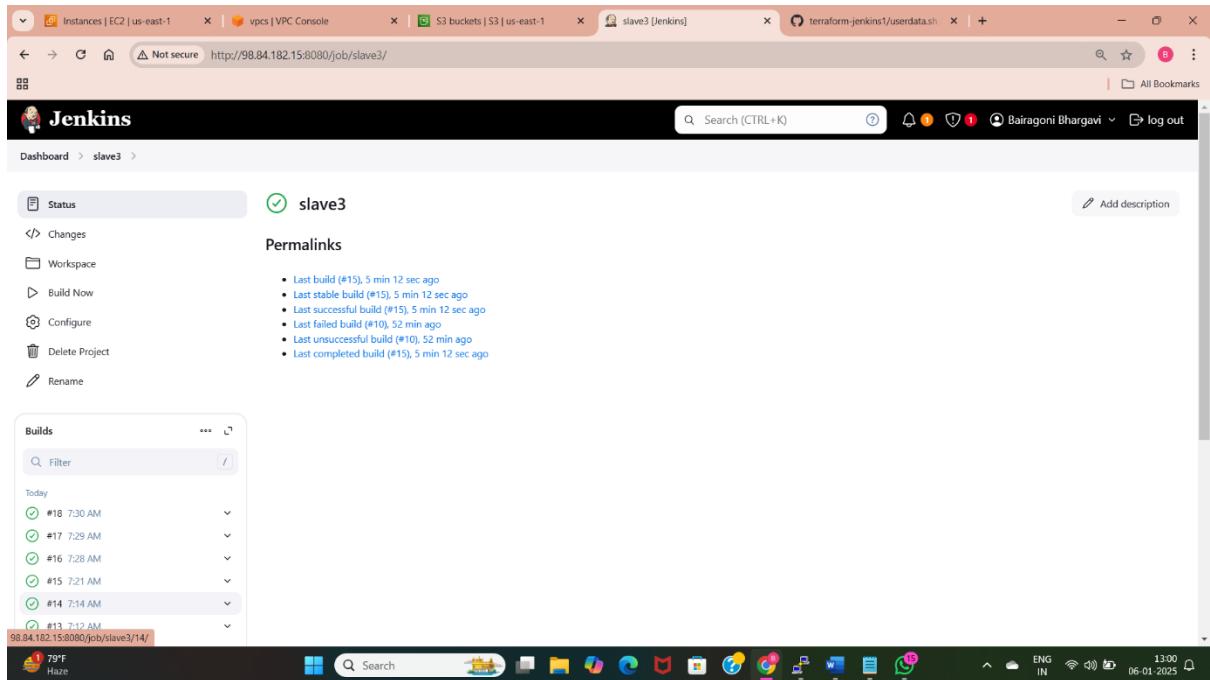


- As the build is success we can able to access the wordpress application which is deployed using slave2 server by using its public IP address.

SLAVE-3:

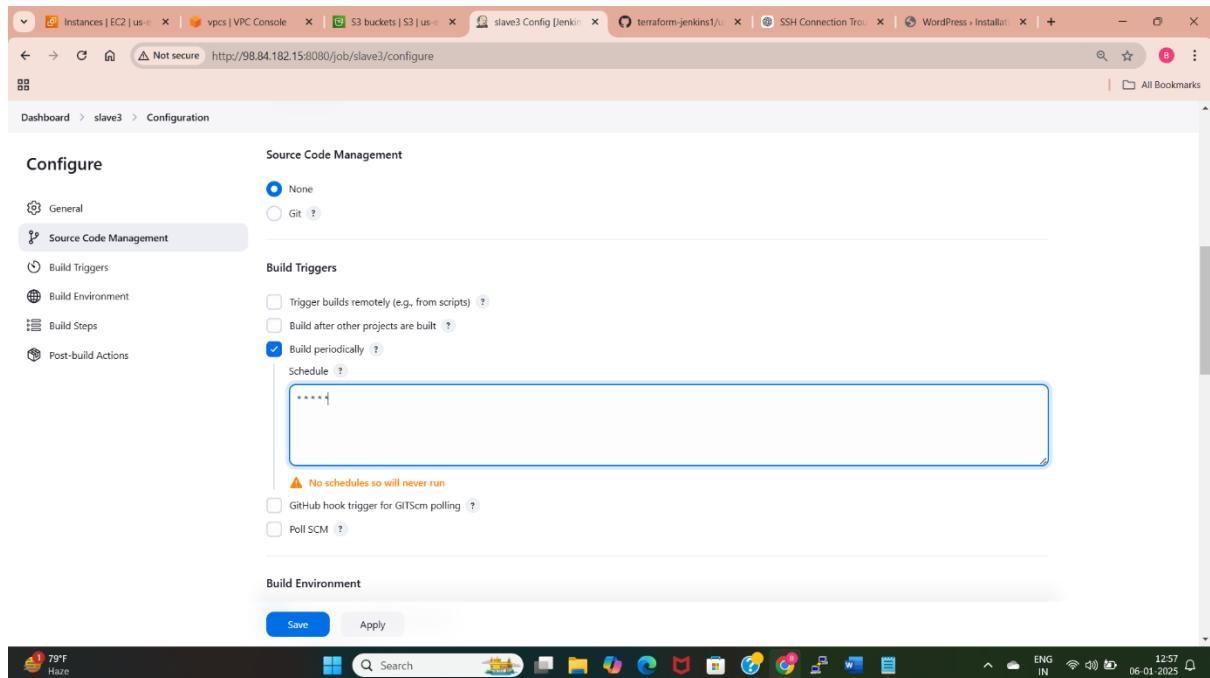
Now connect the slave3 server with the master using the master public key and paste that key in the slave3 authorized keys.

Go to manage Jenkins select nodes and create a new node using the public IP address of the slave and credentials of the master.

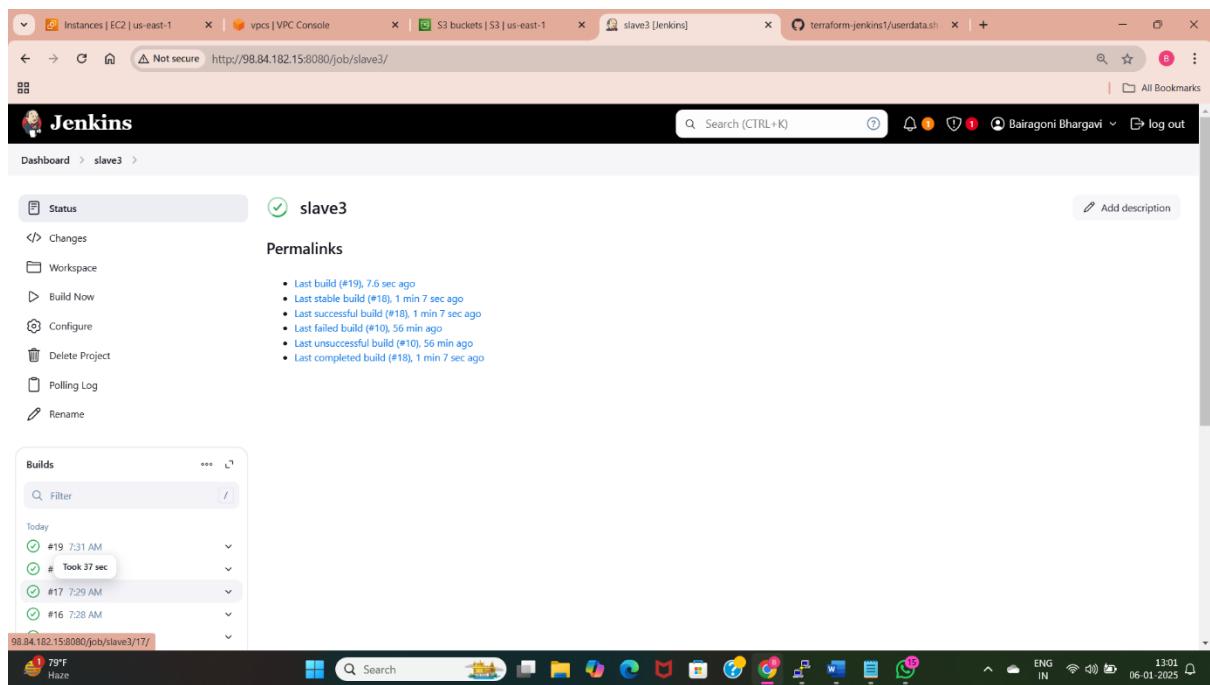


The build is started by the slave3 server and it is success. Now we can able to access the application using this slave server.

SLAVE-4:



Now we need to deploy the application using automation and terraform. For that we need to select build periodically option and give syntax according to the requirement. Here we need the builds for every minute so (* * * * *) is given.



Now save the changes and check for the builds. A new build is started when we save the build periodically option.

New Item

Enter an item name
slave4

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

Create a new job for slave 4 for poll scm(automatic deployment).

Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Build Environment

AWS access key and secret

Access Key Variable
AWS_ACCESS_KEY_ID

Secret Key Variable
AWS_SECRET_ACCESS_KEY

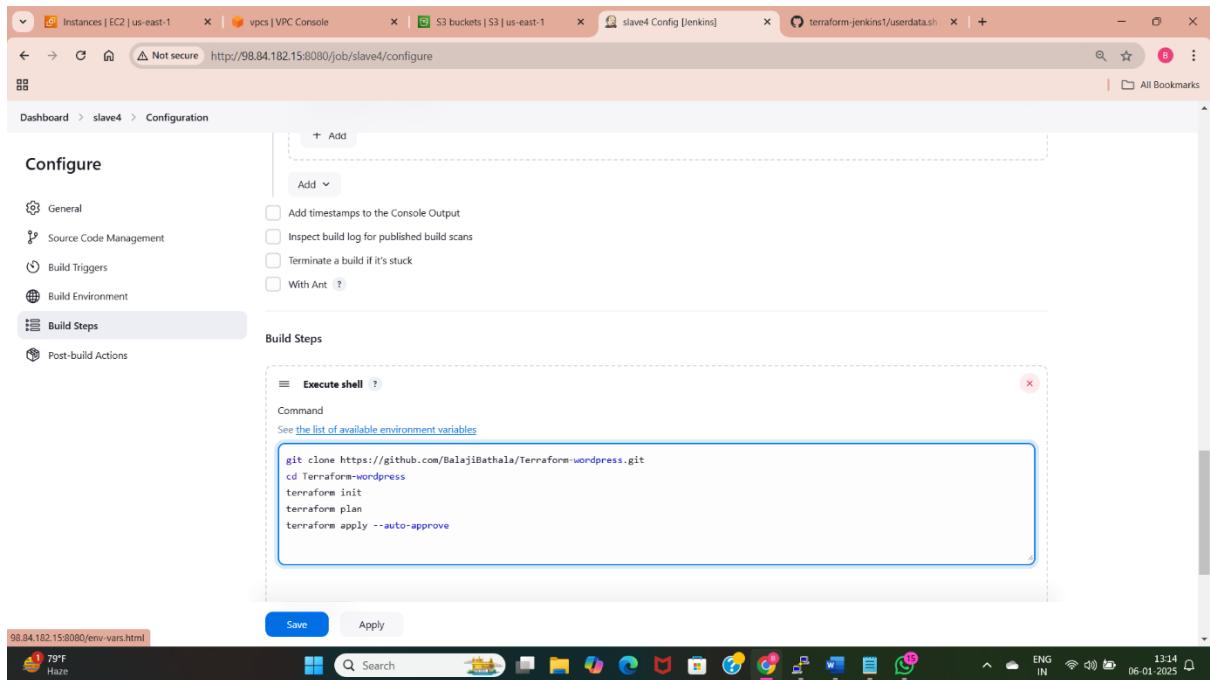
Credentials

Specific credentials Parameter expression

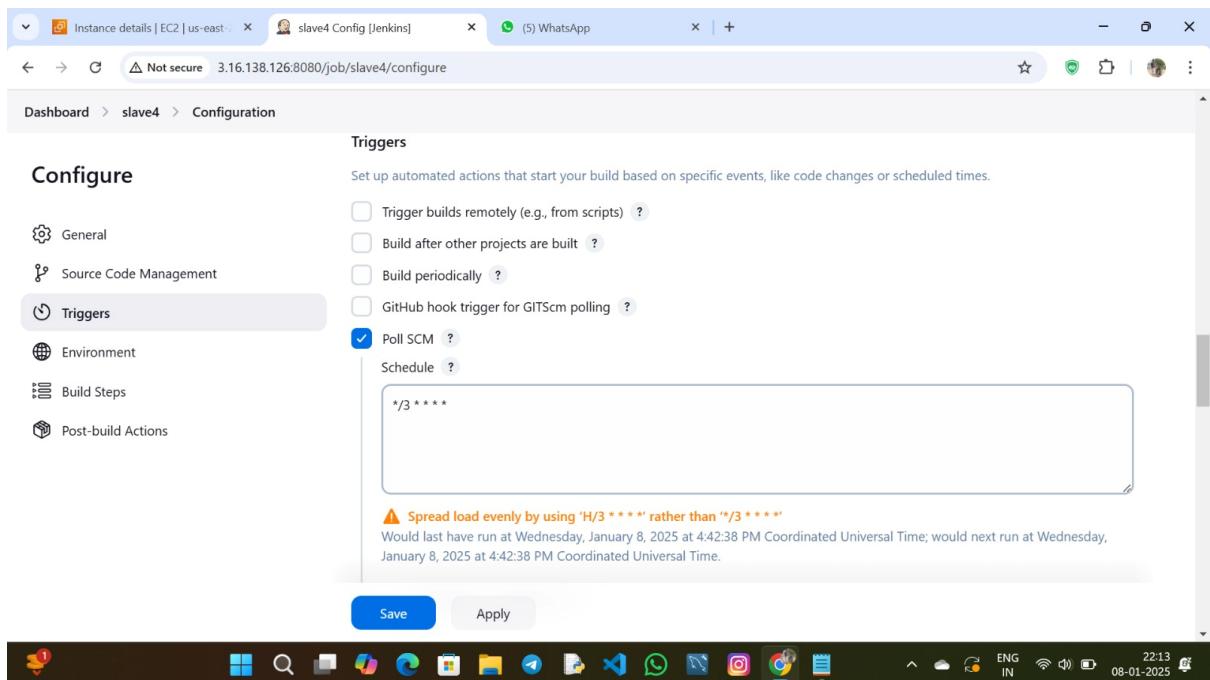
AKIAW3MEEHM4P6VFDDZ

Save Apply

We need to provide the aws credentials to create the aws resources using the terraform automation. For that install the aws credentials plugin and provide credentials.



Now select the execute shell under build steps. Take the code from github, initialize the terraform in that files and see the plan what are going to create using that files and apply the changes using the “terraform apply –auto-approve”.



Now we are going to deploy the wordpress application using the poll scm. For that create a webhook in the github repository(go to github repository>settings>webhook>add webhook>give payload url and application type, create webhook).

The screenshot shows the Jenkins configuration page for a slave node named 'slave4'. The left sidebar lists several configuration sections: General, Source Code Management, Triggers, Environment, Build Steps (which is currently selected), and Post-build Actions. The main content area is titled 'Build Steps' and contains a sub-section titled 'Execute shell'. Under 'Command', there is a text input field containing the following Jenkinsfile code:

```
sudo yum install -y yum-utils shadow-utils  
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo  
sudo yum -y install terraform  
terraform init  
terraform fmt  
terraform validate  
terraform apply --auto-approve
```

At the bottom of the 'Execute shell' panel are 'Save' and 'Apply' buttons.

Go to build steps and select execute shell and give commands to install terraform, and initialize the terraform in the files and apply the changes.

The screenshot shows the Jenkins console output for build #11. The left sidebar has links for Status, Changes, Console Output (which is selected), Edit Build Information, Delete build '#11', Polling Log, Timings, Git Build Data, and Previous Build. The main content area is titled 'Console Output' and displays the following log entries:

```
Started by an SCM change  
Running as SYSTEM  
Building remotely on slave4 in workspace /home/ec2-user/workspace/slave4  
[WS-CLEANUP] Deleting project workspace...  
[WS-CLEANUP] Deferred wipeout is used...  
[WS-CLEANUP] Done  
The recommended git tool is: NONE  
No credentials specified  
Cloning the remote Git repository  
Cloning repository https://github.com/bhargavibairagoni/terraform-jenkins1.git  
> git init /home/ec2-user/workspace/slave4 # timeout=10  
Fetching upstream changes from https://github.com/bhargavibairagoni/terraform-jenkins1.git  
> git --version # timeout=10  
> git --version # 'git version 2.40.1'  
> git fetch --tags --force --progress -- https://github.com/bhargavibairagoni/terraform-jenkins1.git  
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

Now check for the builds. The build is started by SCM change(when we change code in the github) and creates the resources.

SLAVE-5:

The screenshot shows the AWS EC2 Instances page. The left sidebar includes options like 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. The main content area displays 'Instances (1/2) info' with a table showing two entries:

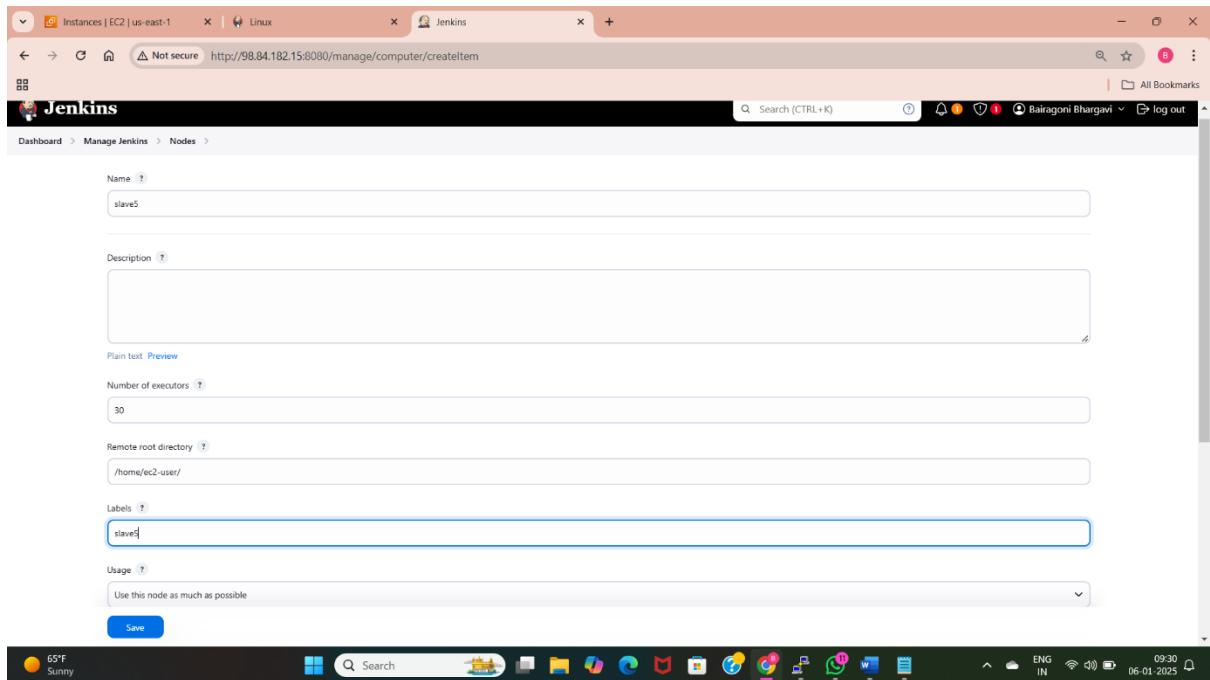
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability zone	Public IPv4 address	Public IPv4 DNS	Elastic IP	IPv6
Master	i-05601b2...	Running	t2.micro	2/2 checks p...	View alarms +	us-east-1c	ec2-98-84...	98.84.182.15	-	-
slave5	i-07b950de...	Running	t2.micro	Initializing	View alarms +	us-east-1d	ec2-54-15...	54.159.67.204	-	-

The instance details for 'slave5' are shown in the expanded view, including its Public IP (54.159.67.204), Private IP (172.31.45.240), and DNS name (ec2-54-159-67-204.compute-1.amazonaws.com). The instance is currently running.

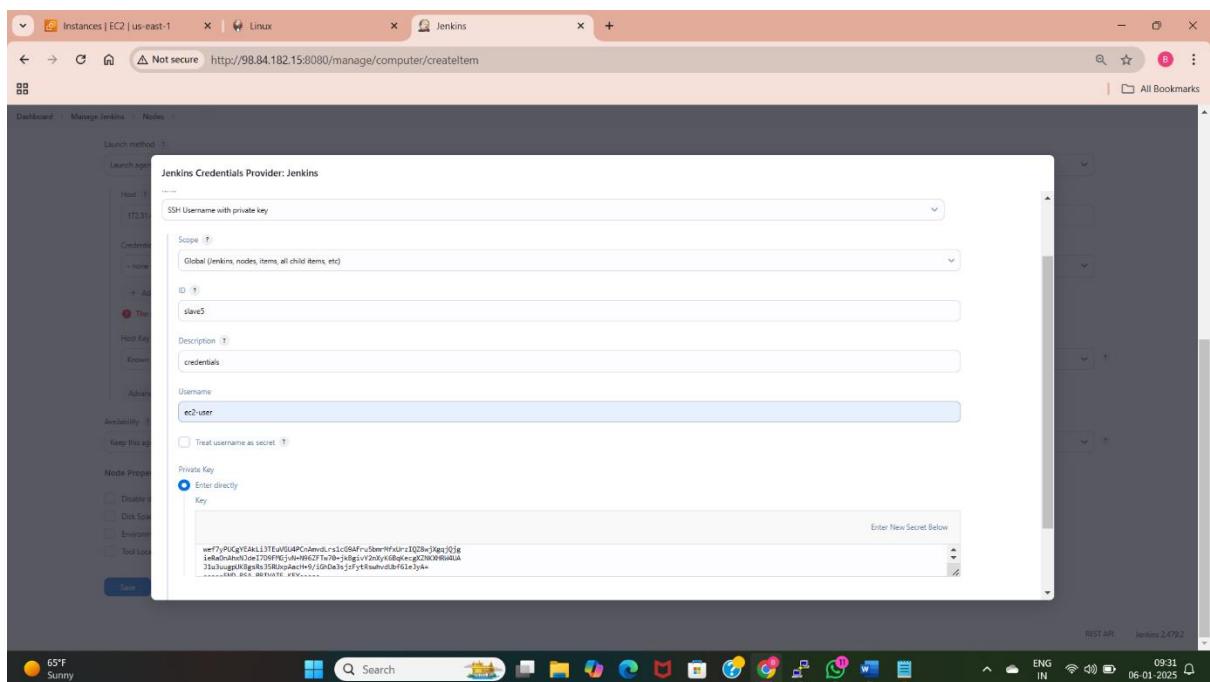
Create a new slave server and connect it with the master.

The screenshot shows the Jenkins 'New node' configuration page. The URL is http://98.84.182.15:8080/manage/computer/new. The page has a 'Node name' field containing 'slave5', a 'Type' section with 'Permanent Agent' selected (radio button is checked), and a note explaining it adds a plain, permanent agent to Jenkins. A 'Create' button is at the bottom.

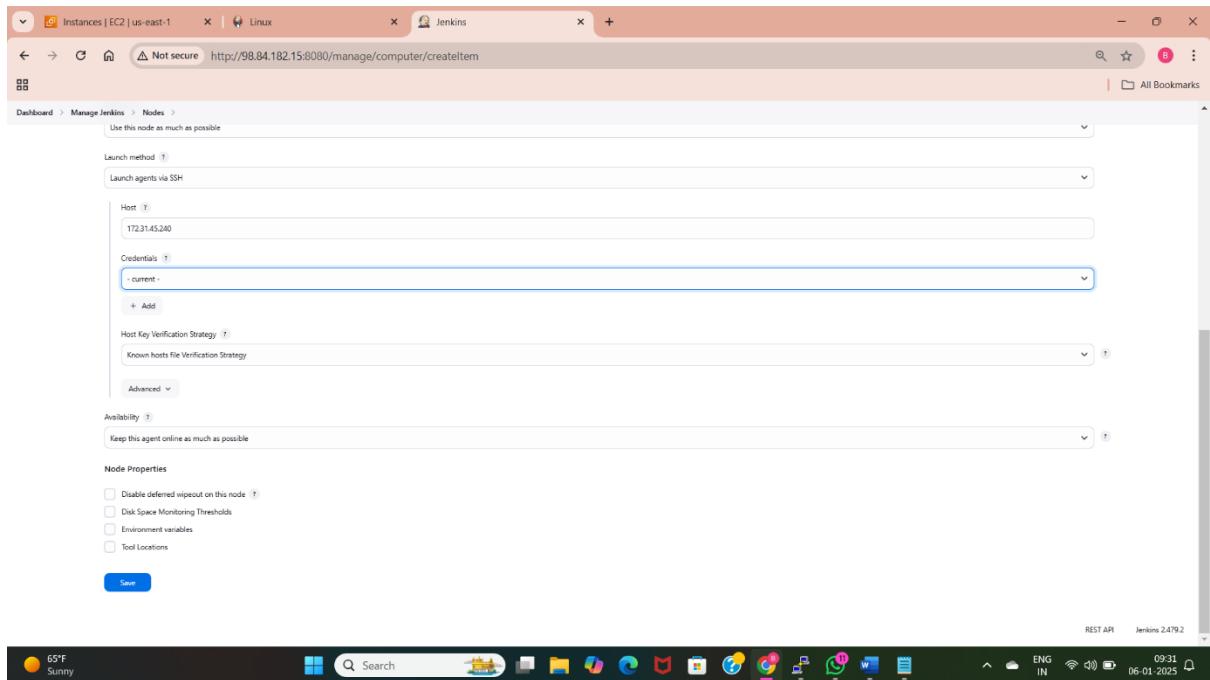
Now create a slave node by moving into manage Jenkins and select nodes and create new node.



For creating new slave5 node give the node name, select number of executors and give remote root directory of the slave5 node, give label name.



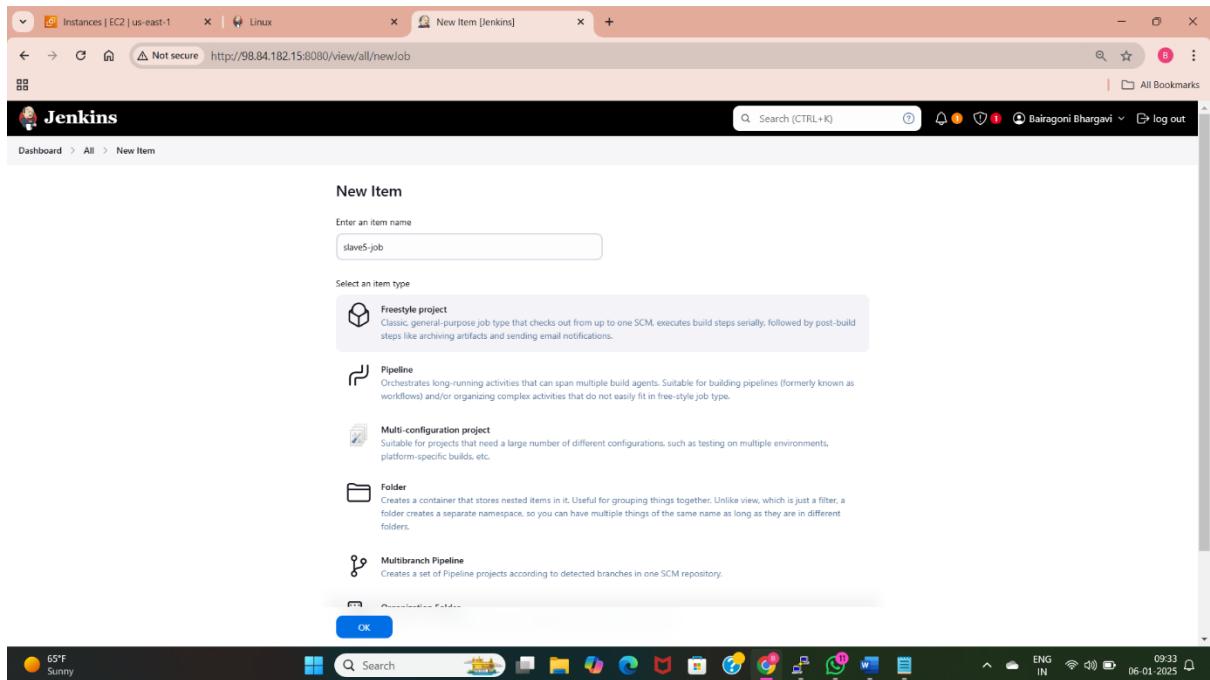
Now provide the credentials using SSH username with private key. Give id as any name, give description for credentials, give the username of the slave5 server, and select the option to enter directly private key



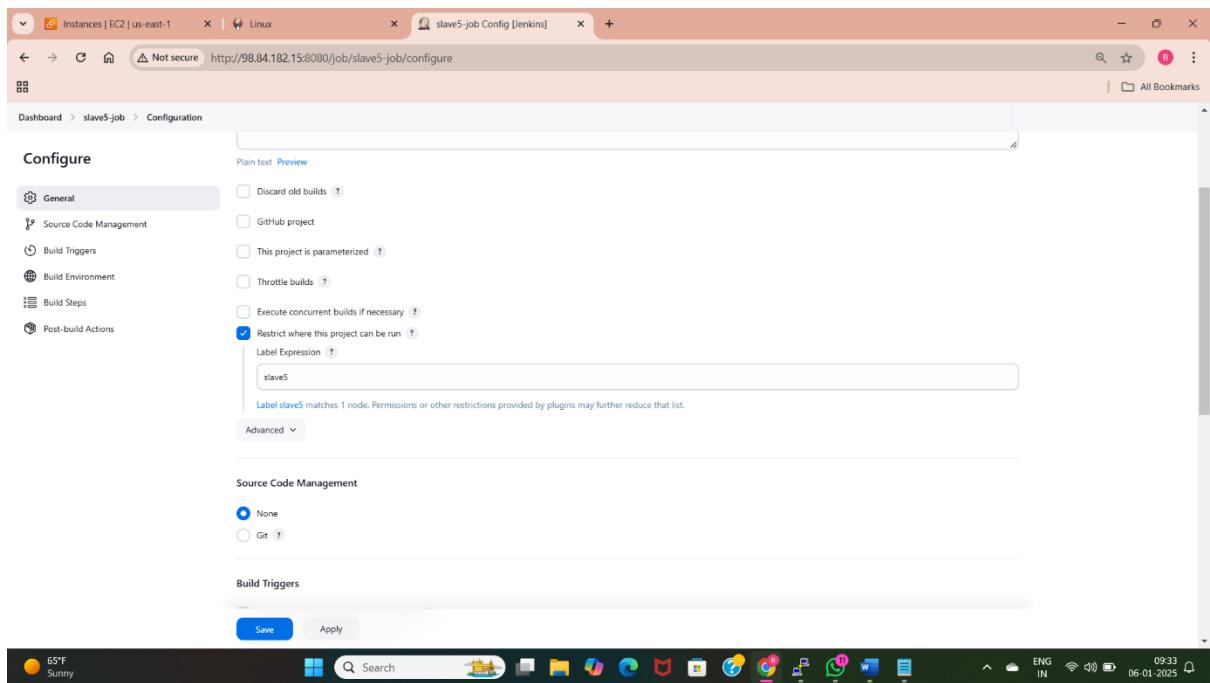
Now provide the public or private ip address of slave server as host. Give the above created credentials and provide host key verification for creating the node.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	16.07 GiB	0 B	16.07 GiB	0ms
	slave1	Linux (amd64)	In sync	10.89 GiB	0 B	10.89 GiB	81ms
	slave2	Linux (amd64)	In sync	3.92 GiB	0 B	3.92 GiB	65ms
	slave3	Linux (amd64)	In sync	5.85 GiB	0 B	5.85 GiB	132ms
	slave5	Linux (amd64)	In sync	3.77 GiB	0 B	3.77 GiB	65ms
	last checked	40 min	40 min	40 min	40 min	40 min	40 min

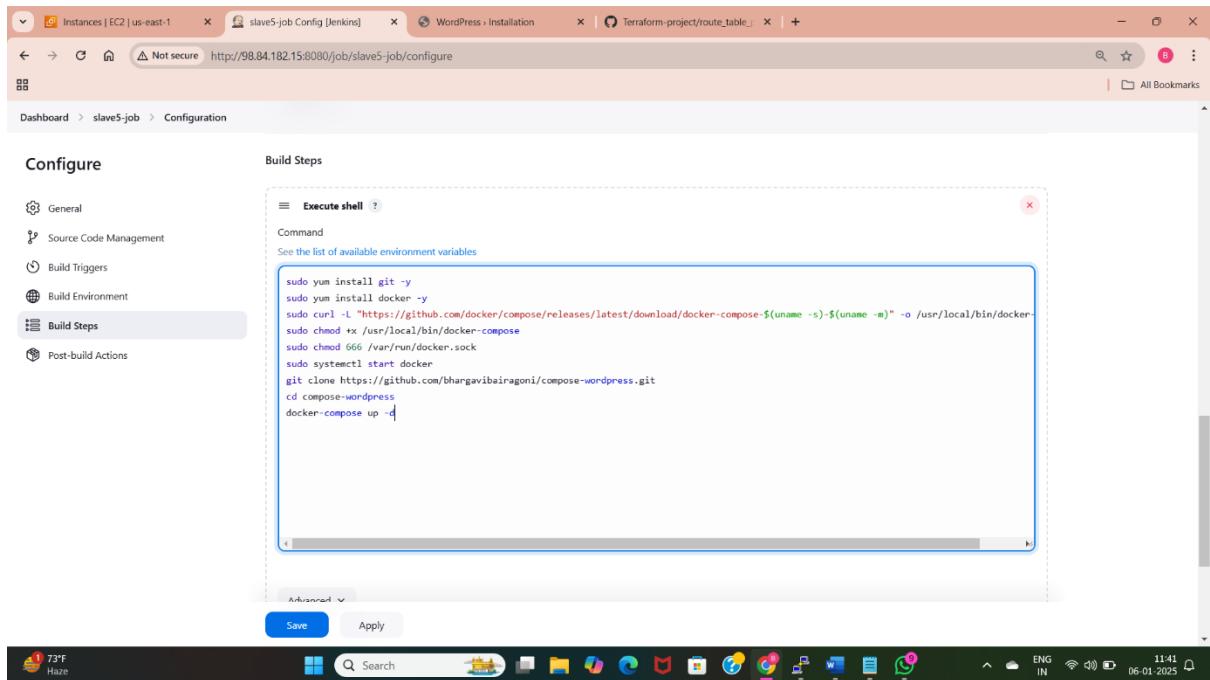
These are the created nodes by all the slave servers and they are launched successfully.



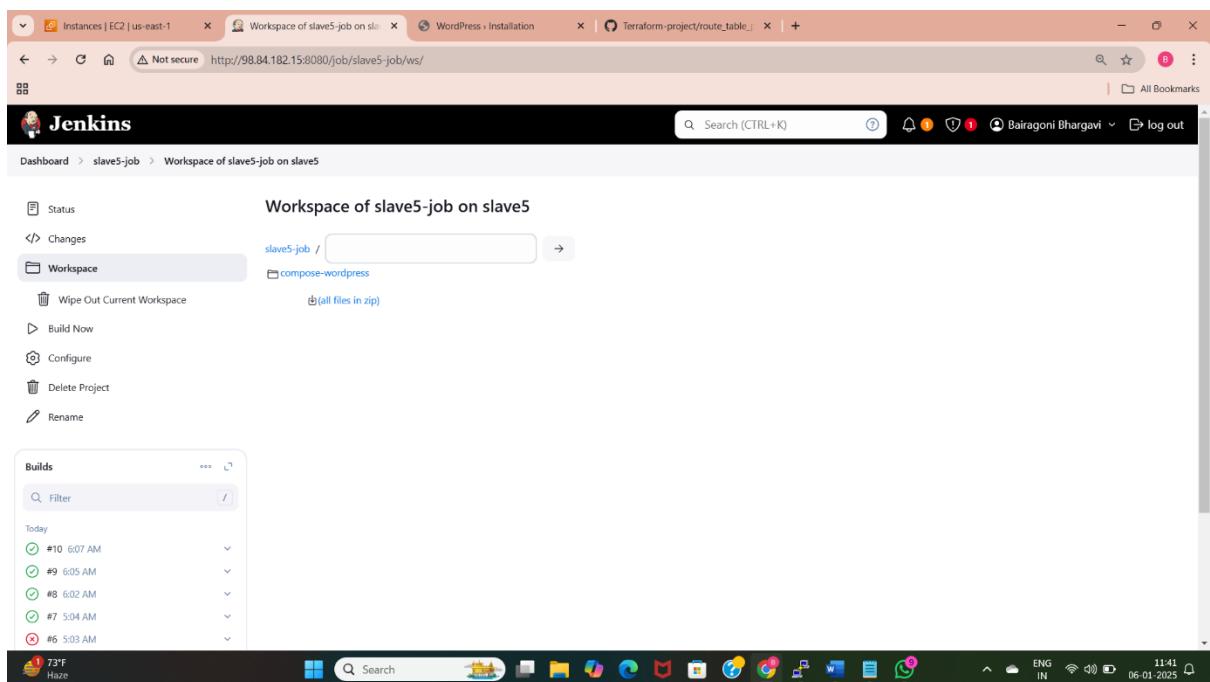
Now create a new freestyle job for the slave5 server to deploy the application.



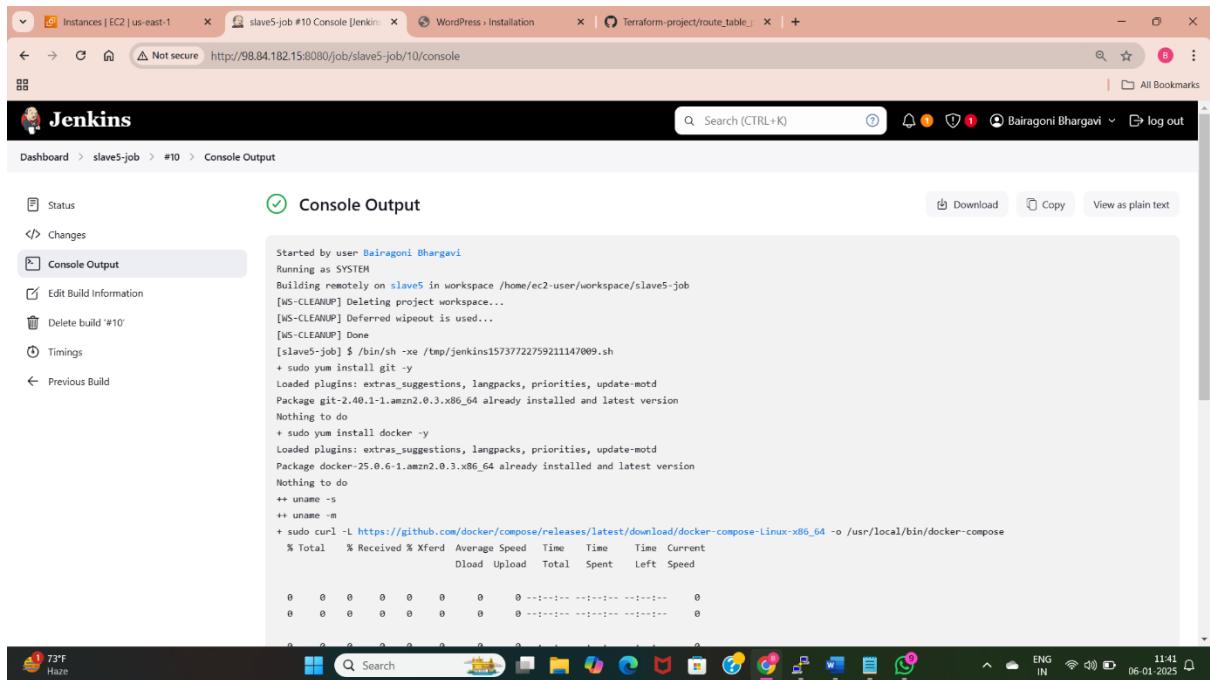
Now select the above created label to build the code(so all the build will be performed by slave 5 server node).



Now under the build steps select execute shell and write the bash script which are used to deploy the wordpress application using the docker.

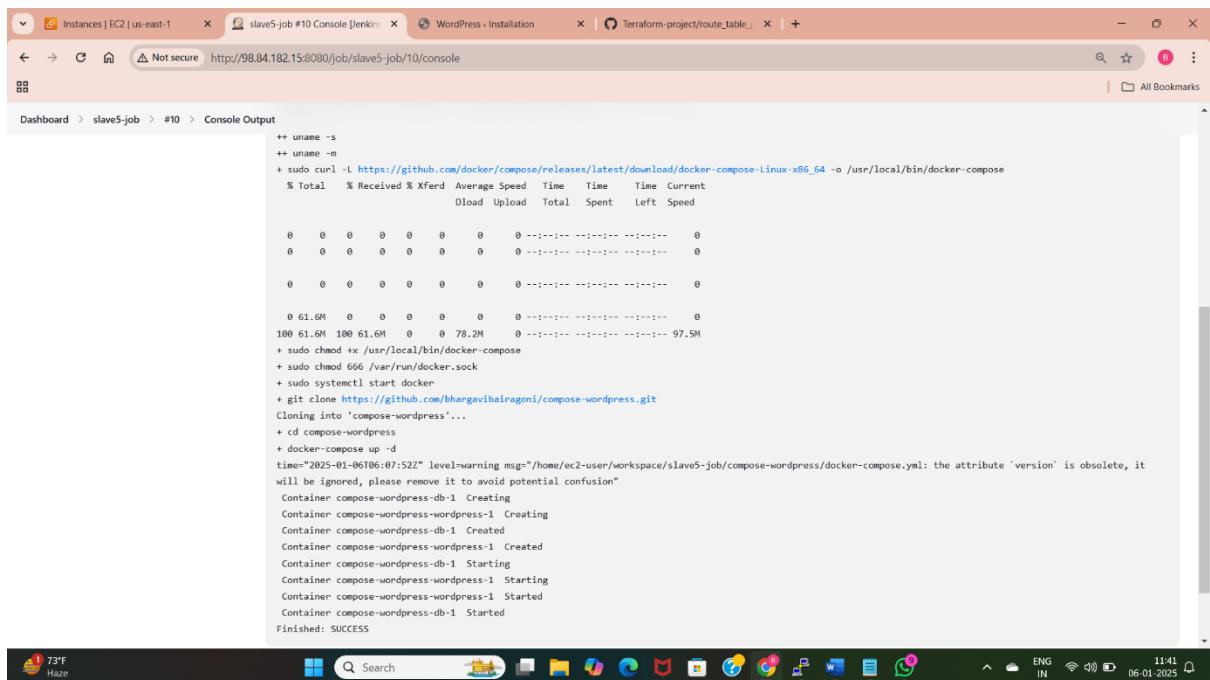


Now save and click on build now option. The build is success and we can see the compose-worpress file which is having the code.



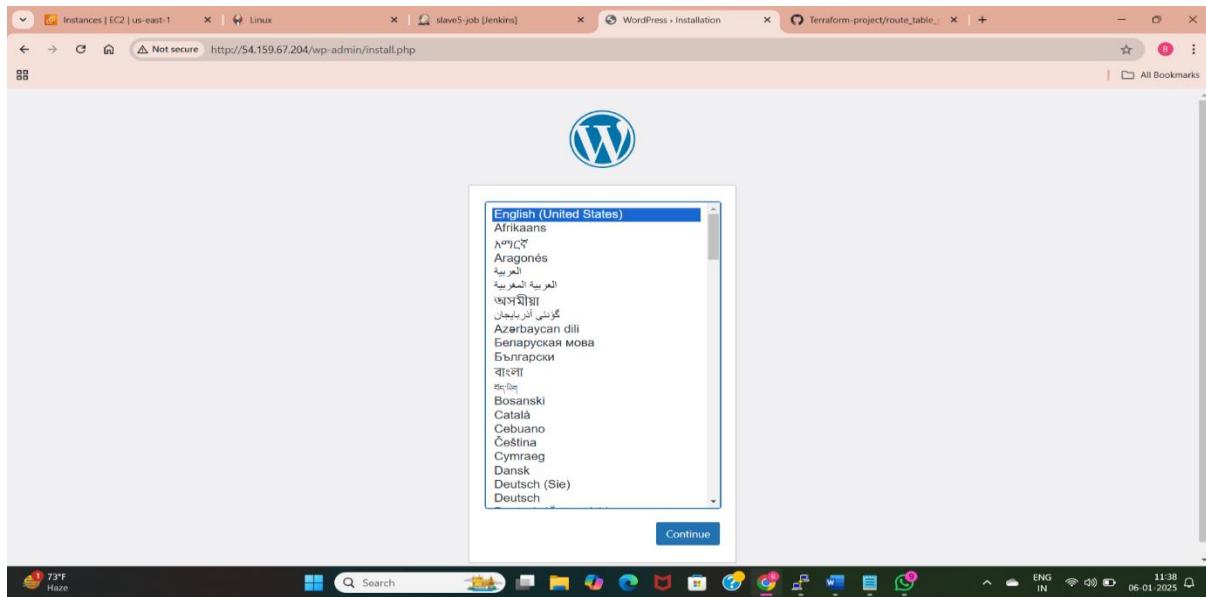
Started by user Bairagoni Bhargavi
Running as SYSTEM
Building remotely on **slave5** in workspace /home/ec2-user/workspace/slave5-job
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[slave5-job] \$ /bin/sh -xe /tmp/jenkins15737722759211147009.sh
+ 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 docker -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package docker-25.0.6-1.amzn2.0.3.x86_64 already installed and latest version
Nothing to do
++ uname -s
++ uname -m
+ sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-Linux-x86_64 -o /usr/local/bin/docker-compose
% Total % Received % Xferd Average Speed Time Time Current
 Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0

We can find the detailed information related to the build which is started by the user bhargavi running on slave5 node.



```
++ uname -s
++ uname -m
+ sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-Linux-x86\_64 -o /usr/local/bin/docker-compose
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
0 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
0 61.6M 0 0 0 0 0 0 0 ---:---:---:---:---:---:---:--- 0
100 61.6M 100 61.6M 0 0 78.2M 0 ---:---:---:---:---:---:---:--- 97.5M
+ sudo chmod +x /usr/local/bin/docker-compose
+ sudo chmod 666 /var/run/docker.sock
+ sudo systemctl start docker
+ git clone https://github.com/bhargavibairagi/compose-wordpress.git
Cloning into 'compose-wordpress'...
+ cd compose-wordpress
+ docker-compose up -d
time="2025-01-06T06:07:52Z" level=warning msg="/home/ec2-user/workspace/slave5-job/compose-wordpress/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
Container compose-wordpress-db-1 Creating
Container compose-wordpress-wordpress-1 Creating
Container compose-wordpress-db-1 Created
Container compose-wordpress-wordpress-1 Created
Container compose-wordpress-db-1 Starting
Container compose-wordpress-wordpress-1 Started
Container compose-wordpress-db-1 Started
Container compose-wordpress-db-1 Started
Finished: SUCCESS
```

We can see the commands which are provided in the execute shell are running step by step.

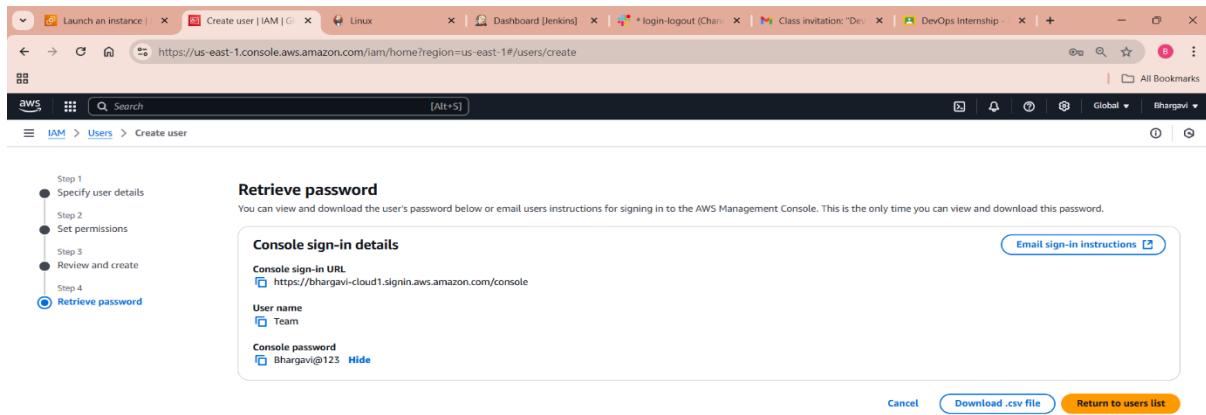


Now copy and paste the public IP address of the instance and paste in browser we can able to access the wordpress application.

METHOD-4 :

In method 4 we are going to deploy the wordpress application using the single person account. Here the master and the slaves are in same account.

- Master, slave-1: deploying wordpress application using k8s – Bhargavi
- Slave-2: deploying wordpress application using docker and docker-compose- Vamsi
- Slave-3,4: deploying wordpress application using terraform manually and automatically- Vasanthi
- Slave-5: deploying wordpress application using the bash script with docker and-docker compose – Balaji



Now am creating a IAM user to give access to my team members, am going to share this account with my team to deploy different applications using different servers.

The screenshot shows the AWS Management Console EC2 Instances page. A success message at the top indicates the termination of instance i-0aa051a2989f7023. The main table lists seven instances:

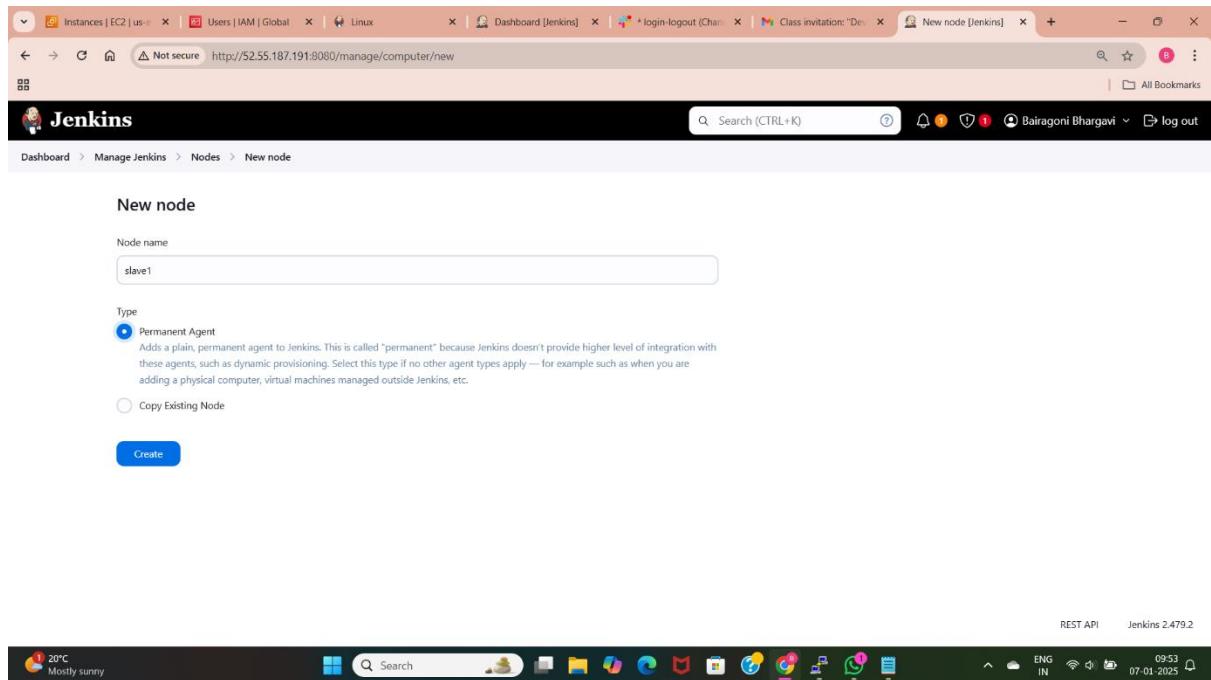
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability zone	Public IPv4 address	Elastic IP	IPv6
slave1	i-0f605d9b...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-100-2...	100.26.104.76	-
slave2	i-0fdcb4b9...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-5-91-5...	3.91.57.103	-
slave3	i-0df25c8b...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-3-84-2...	5.84.232.105	-
slave4	i-05d472c4...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-5-92-0...	3.92.0.68	-
master-same account	i-0fe0b03...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1c	ec2-52-55-...	52.55.187.191	-
slave5	i-0d72b68...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-54-14...	54.147.46.163	-
Master	i-0ddff820...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-54-17...	54.172.27.233	-

Here five slave servers with one master server is created by the team.

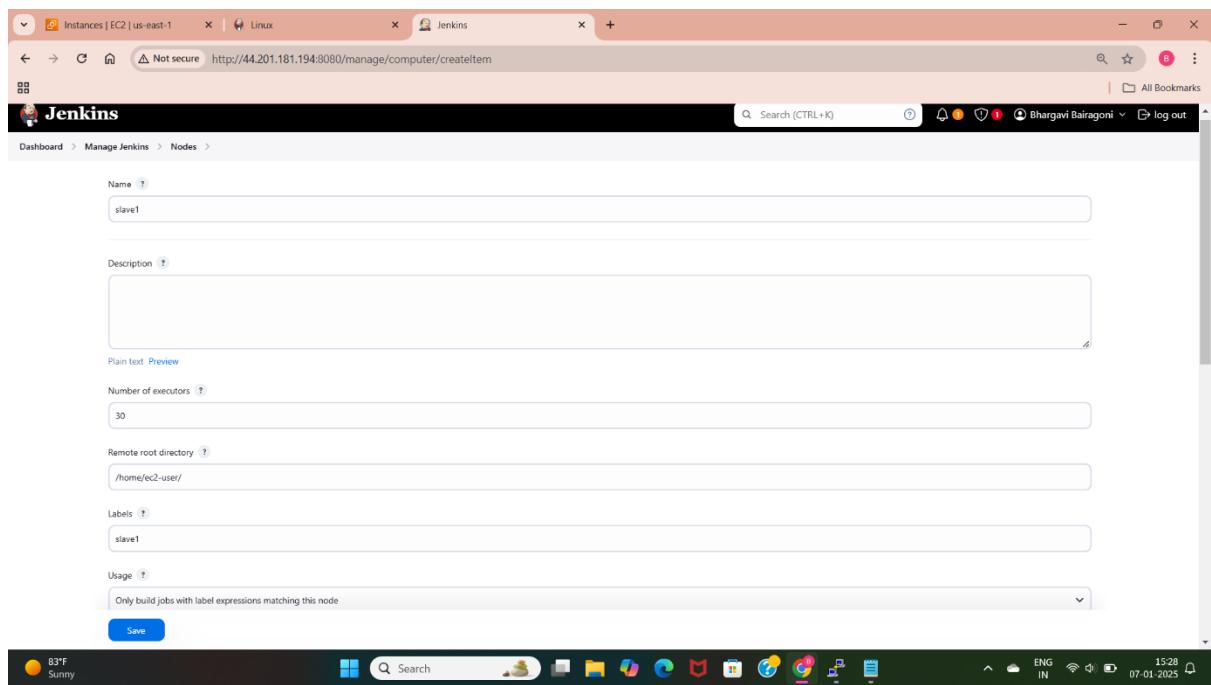
The screenshot shows the Jenkins 'Getting Started' screen titled 'Unlock Jenkins'. It instructs the user to copy the password from the file /var/lib/jenkins/secrets/initialAdminPassword. A redacted input field is provided for pasting the password.

Now install the Jenkins in master server and start the Jenkins, access the Jenkins by giving Jenkins password.

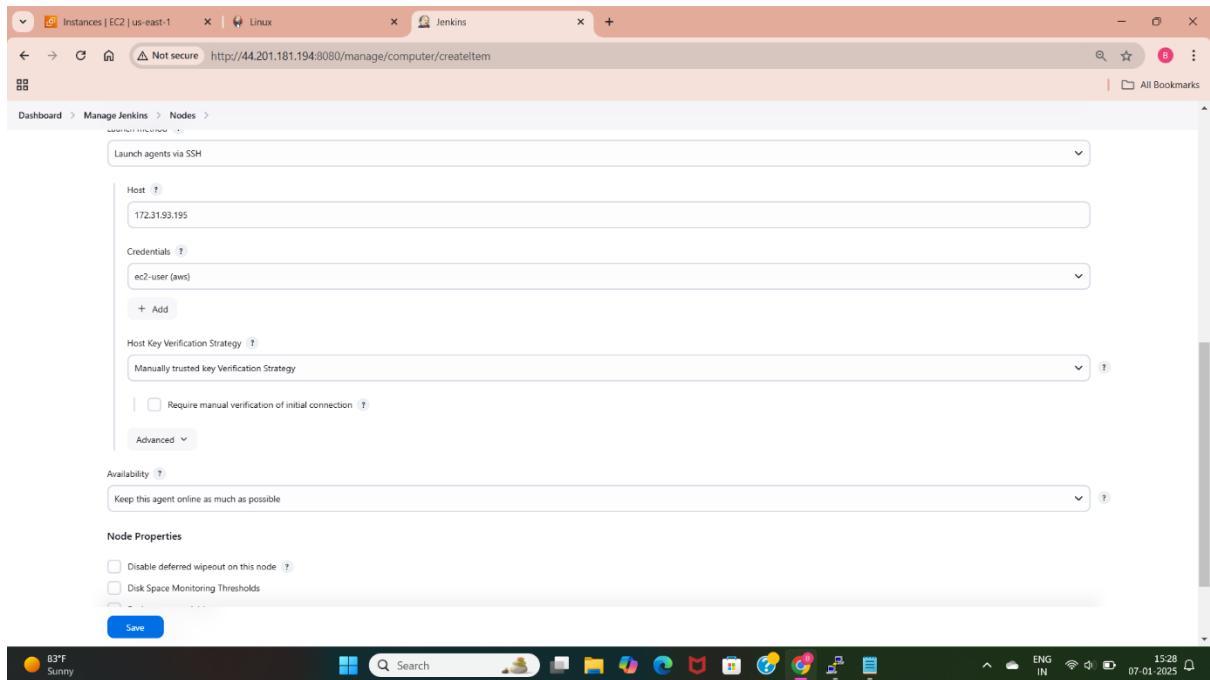
SLAVE-1:



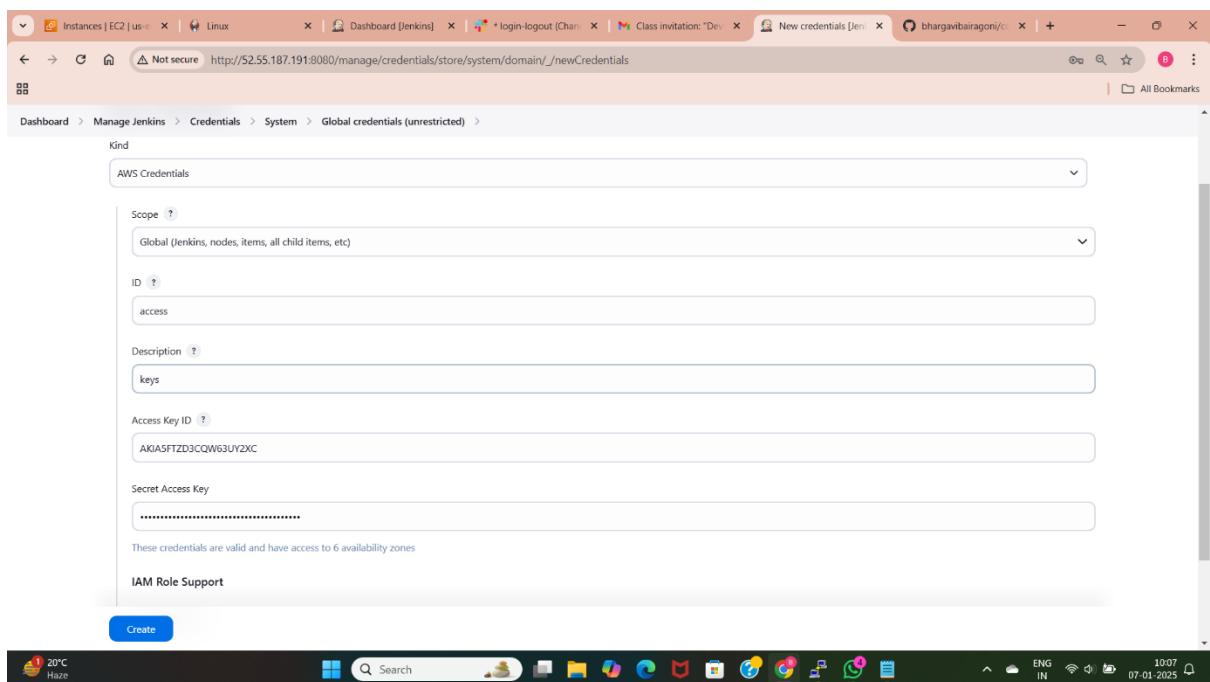
Now create a slave node by selecting manage Jenkins> nodes and give name for the node.



Creating a slave 1 server, giving number of executers, root directory of the slave server, create label to identify.



Now select the launch method as agents via SSH, give host, and credentials of the master to connect with the slave and select host key verification strategy.



Here am providing the aws credentials by giving access keys and secret access keys by installing aws credentials plugin.

New Item

Enter an item name
slave1

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.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

Creating a new free style job in Jenkins for slave1.

General

Enabled

Configure

General

Description

Plain text Preview

Discard old builds ?

GitHub project

This project is parameterized ?

Throttle builds ?

Execute concurrent builds if necessary ?

Restrict where this project can be run ?

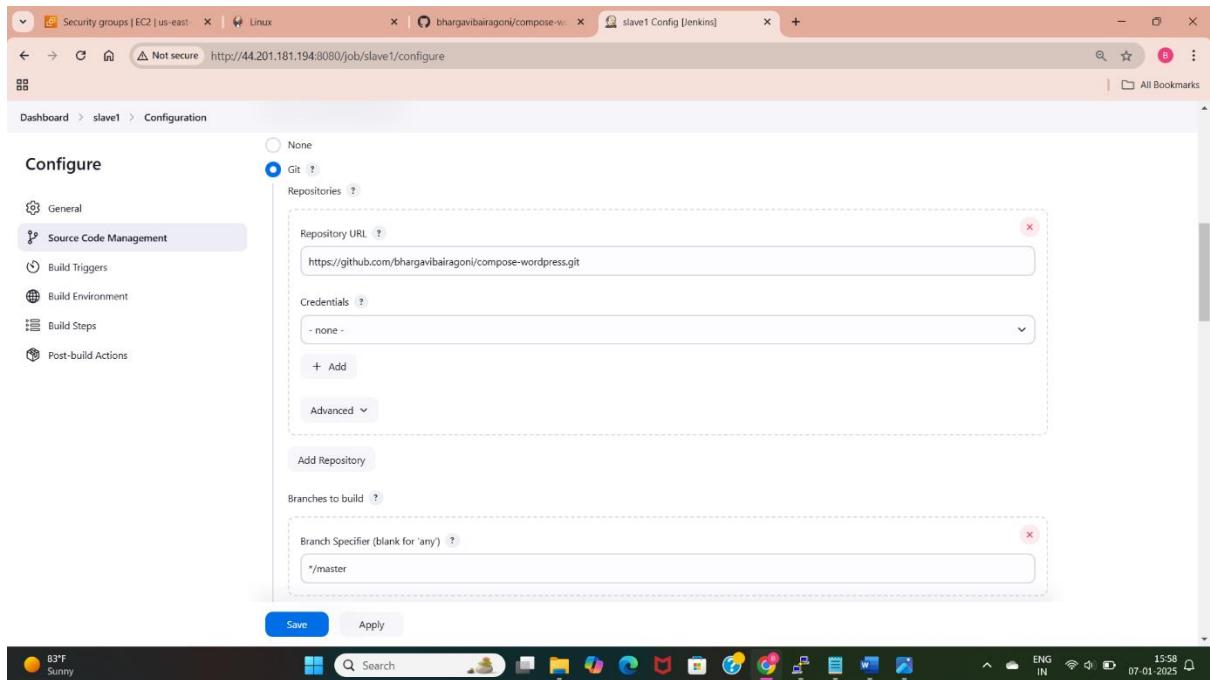
Label Expression ?
slave1

Label slave1 matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

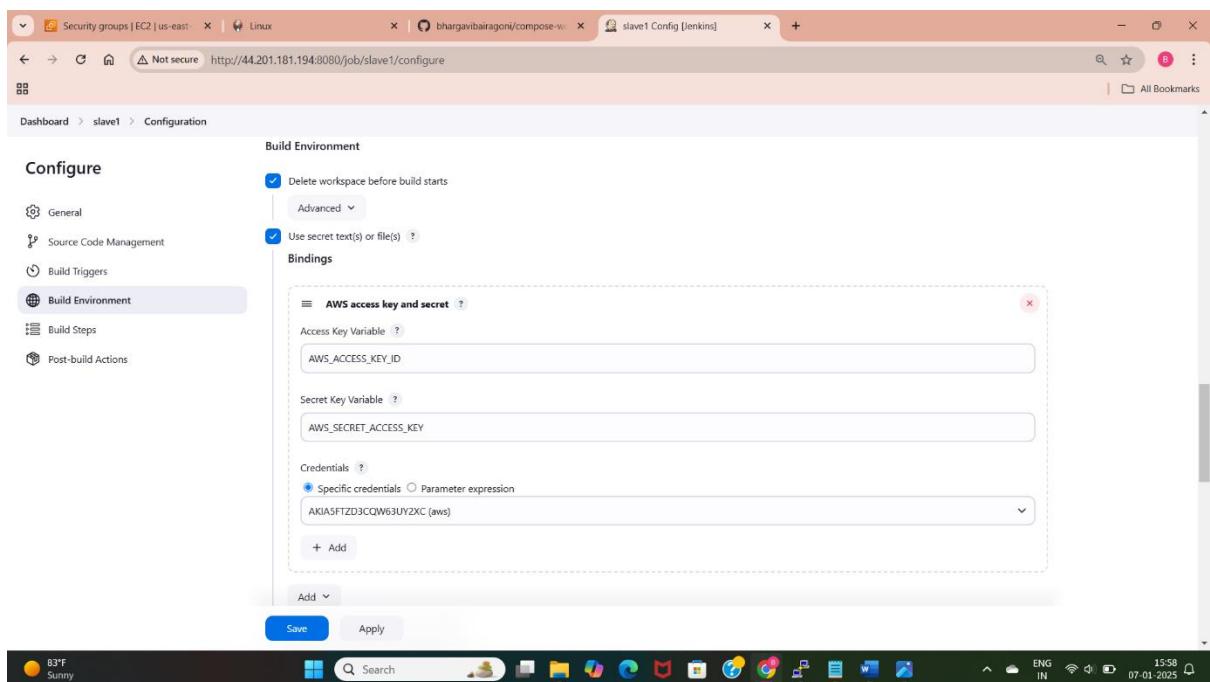
Advanced

Save Apply

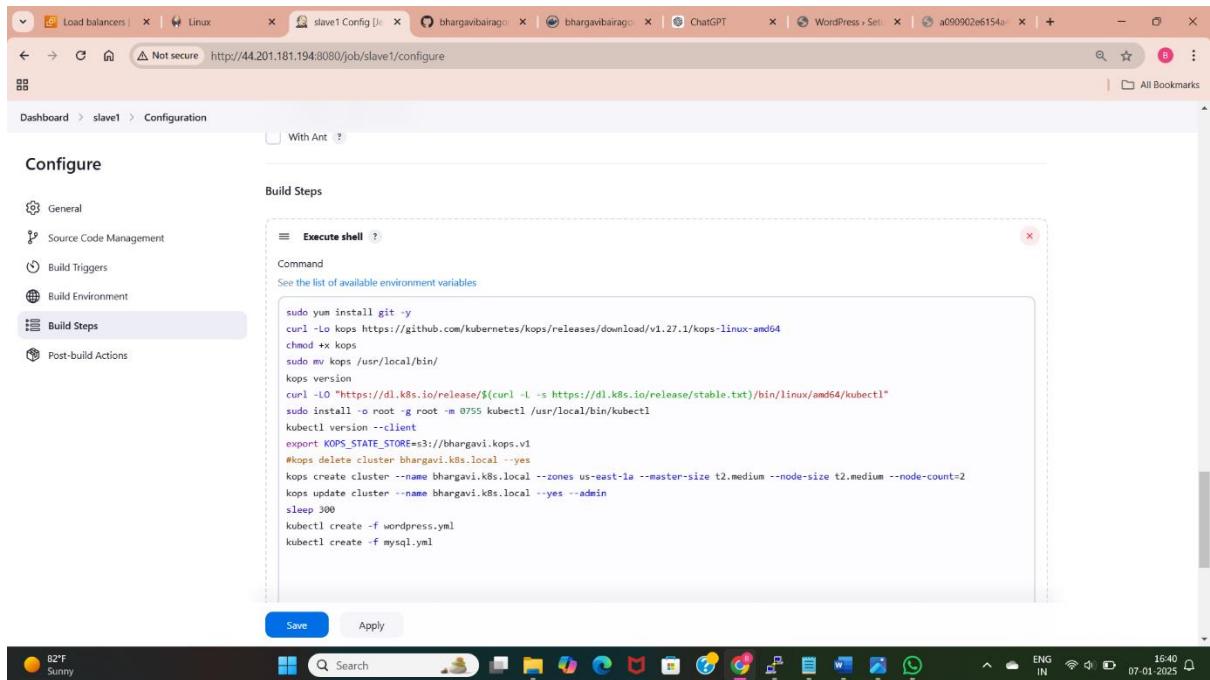
Here am restricting the builds on Jenkins by selecting slave1 label.



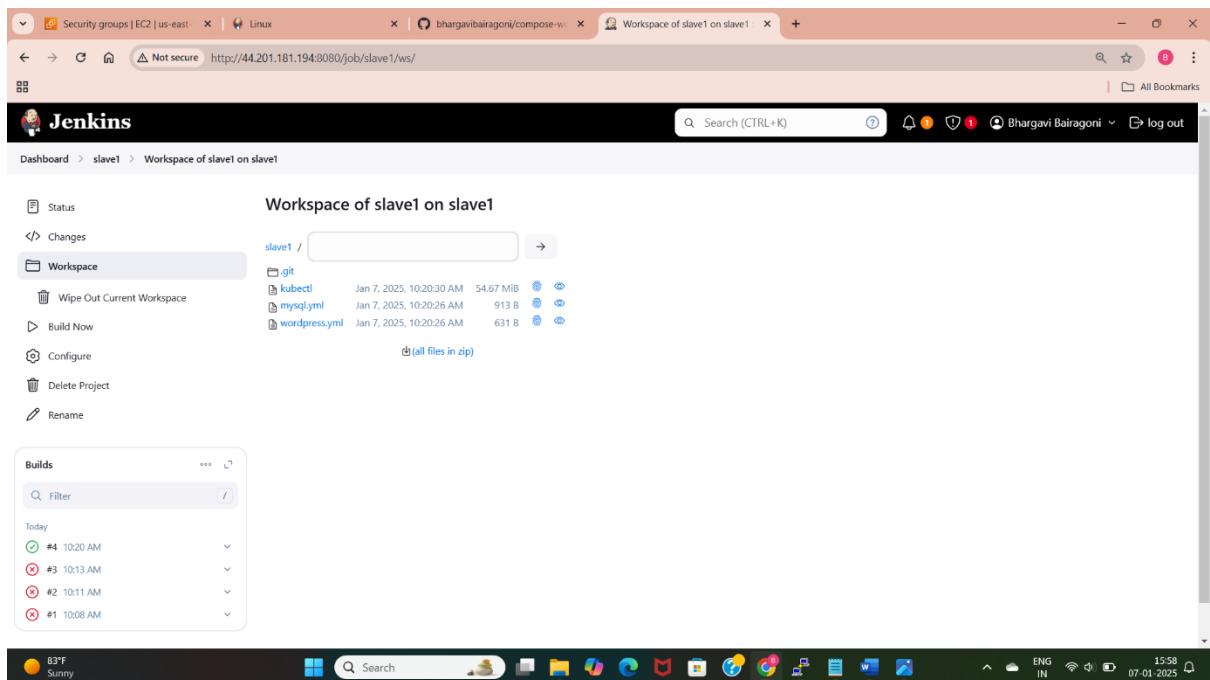
Select git option under source code management and give repository url and branch name where we are having the files.



Under build environment select the delete workspace before build starts(it will deletes the before build files and newly downloads the files). Give the access keys and secret access keys by selecting “use secret text or files”.



Select execute shell under build steps and give shell commands to install git, kops, kubectl , create a s3 bucket, create clusters(nodes and control plane) using kops create deployment and service files.



The build is success, and the files related to deployment and service are saved in the workspace of this slave1 job.

```

Started by user Bhargavi Bairagoni
Running as SYSTEM
Building remotely on slave1 in workspace /home/ec2-user/workspace/slave1
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/bhargavibairagoni/compose-wordpress.git
> git init /home/ec2-user/workspace/slave1 # timeout=10
Fetching upstream changes from https://github.com/bhargavibairagoni/compose-wordpress.git
> git --version # timeout=10
> git fetch --tags --force --progress -- https://github.com/bhargavibairagoni/compose-wordpress.git +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url https://github.com/bhargavibairagoni/compose-wordpress.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 3fb3b470666aa8f3623917d0cb61d4a251f4d73e (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 3fb3b470666aa8f3623917d0cb61d4a251f4d73e # timeout=10
Commit message: "Delete docker-compose.yml"
> git rev-list --no-walk 3fb3b470666aa8f3623917d0cb61d4a251f4d73e # timeout=10

```

We can see the detailed steps of execute shell like what are the packages, softwares, and files are going to install in this slave.

```

I0107 10:22:08.968792 18086 vfs_keystorereader.go:143] CA private key was not found
I0107 10:22:09.090352 18086 keypair.go:226] Issuing new certificate: "service-account"
I0107 10:22:09.110405 18086 keypair.go:226] Issuing new certificate: "kubernetes-ca"
I0107 10:22:11.369235 18086 executor.go:111] Tasks: 44 done / 97 total; 19 can run
I0107 10:22:12.559296 18086 executor.go:111] Tasks: 63 done / 97 total; 24 can run
I0107 10:22:13.541863 18086 executor.go:111] Tasks: 87 done / 97 total; 2 can run
I0107 10:22:13.672840 18086 executor.go:111] Tasks: 89 done / 97 total; 4 can run
I0107 10:22:14.213973 18086 executor.go:111] Tasks: 93 done / 97 total; 4 can run
I0107 10:22:15.534853 18086 executor.go:111] Tasks: 96 done / 97 total; 1 can run
I0107 10:22:16.879865 18086 executor.go:111] Tasks: 97 done / 97 total; 0 can run
I0107 10:22:16.927153 18086 update_cluster.go:323] Exporting kubeconfig for cluster
KOps has set your kubectl context to bhargavi.k8s.local

Cluster is starting. It should be ready in a few minutes.

Suggestions:
* validate cluster: kops validate cluster --wait 10m
* list nodes: kubectl get nodes --show-labels
* ssh to a control-plane node: ssh -i ~/.ssh/id_rsa ubuntu@
* the ubuntu user is specific to Ubuntu. If not using Ubuntu please use the appropriate user based on your OS.
* read about installing addons at: https://kops.sigs.k8s.io/addons.

+ sleep 300
+ kubectl create -f wordpress.yml
deployment.apps/word-deployment created
service/word-service created
+ kubectl create -f mysql.yml
deployment.apps/mysql-deployment created
service/mysql-service created
Finished: SUCCESS

```

Here we can see that nodes are created and the downloaded files are going to deploy by provided commands.

```
ec2-user@slave1:~/workspace/slave1

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: mysql
  template:
    metadata:
      labels:
        tier: mysql
    spec:
      containers:
        - name: mysql
          image: bhargavibairagoni/mysql:mysqlimage
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "wordpress"
            - name: MYSQL_DATABASE
              value: "databaseword"
            - name: MYSQL_USER
              value: "admin"
            - name: MYSQL_PASSWORD
              value: "ADMIN123"

---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    tier: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  type: LoadBalancer
```

This is the deployment and service file for creating mysql database. We can see 2 replicas of this deployment.

```
ec2-user@slave1:~/workspace/slave1

apiVersion: apps/v1
kind: Deployment
metadata:
  name: word-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: wordpress
  template:
    metadata:
      labels:
        tier: wordpress
    spec:
      containers:
        - name: wordpress-container
          image: bhargavibairagoni/wordpress:wordimage
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: word-service
spec:
  selector:
    tier: wordpress
  ports:
    - protocol: TCP
      port: 80          # External port
      targetPort: 80   # Port inside container
  type: LoadBalancer
```

This is the deployment and service file for wordpress. It is running on the port number 80 with two replicas.

The screenshot shows the AWS EC2 Instances page with the following details:

Name	Instance ID	Instance state	Type	Status check	Alarm status	Availability zone	Public IPv4	Elastic IP	IPv6
slave1	i-01661bbe21de805e1	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-44-20...	44.204.228.94	-
nodes-us-east-1a.b...	i-049fbff1...	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-3-231...	3.231.204.25	-
nodes-us-east-1a.b...	i-02cb304f...	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-18-20...	18.208.202.99	-
master	i-022bb56...	Running	t2.micro	2/2 checks pass	View alarms	us-east-1b	ec2-44-20...	44.201.181.194	-
control-plane-us-e...	i-0c36cab8...	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-3-236...	3.236.192.16	-

i-01661bbe21de805e1 (slave1)

- Details**
- Status and alarms**
- Monitoring**
- Security**
- Networking**
- Storage**
- Tags**

Instance summary

Instance ID: i-01661bbe21de805e1
 Public IPv4 address: 44.204.228.94 | open address
 Instance state: Running
 Private IPv4 addresses: 172.31.93.195
 Public IPv4 DNS: ec2-44-204-228-94.compute-1.amazonaws.com | open address

These are the clusters created by the Kubernetes code.

The screenshot shows the AWS Load Balancers page with the following details:

Name	DNS name copied	State	VPC ID	Availability Zones	Type	Date created
a0902e6154a4eb88a...	a0902e6154a4eb88a0fb967edb86d-1515568669.us-east...	-	vpc-09defc29fef345501	us-east-1a (use1-az1)	classic	January
a4b5cc49f91bf45ac89b...	a4b5cc49f91bf45ac89bd6d51c771705-92087743.us-east-1...	-	vpc-09defc29fef345501	us-east-1a (use1-az1)	classic	January
api-bhargavi-k8s-local-s...	api-bhargavi-k8s-local-s631t9-7db3d694c9c65816.elb.us-eas...	Active	vpc-09defc29fef345501	us-east-1a (use1-az1)	network	January

Load balancer:

These are the load balancers created using kubernetes. To access the application we need to paste any one of the url in the browser.

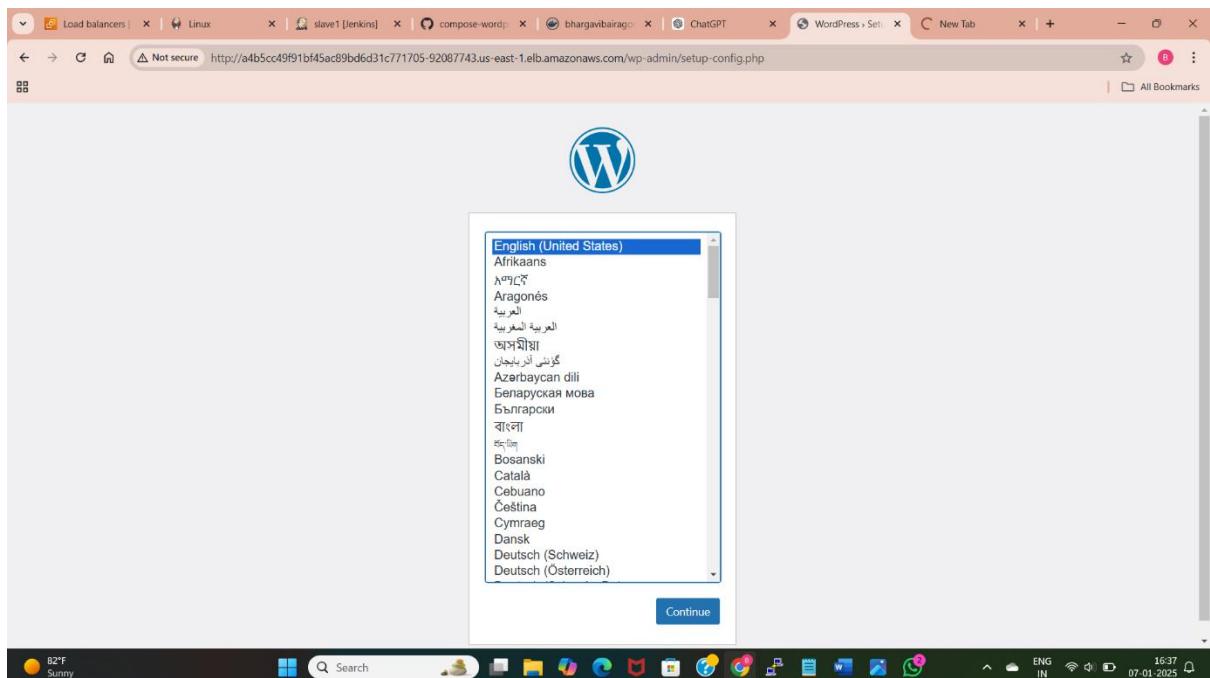
```

[ec2-user@slave1:~/workspace/slave1]
[ec2-user@slave1 slave1]$ kubectl get svc
NAME            TYPE      CLUSTER-IP  EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP  100.64.0.1  <none>        443/TCP       14m
[ec2-user@slave1 slave1]$
└── Login as: ec2-user
└── Authenticating with public key "importedOpensshKey"
Last login: Tue Jan  7 10:32:28 2023 from 205.254.168.131

          _\_\_ 
         / \ \ \ 
        /   \ \ 
       /     \ 
      /       \ 
     /         \ 
    /           \ 
   /             \ 
  /               \ 
 /                 \ 
/                   \ 
Amazon Linux 2
~~~ - \###\ AL2 End of Life is 2025-06-30.
~~~ \###\ 
~~~ \###\ 
~~~ V-^--> A newer version of Amazon Linux is available!
~~~ /--\ 
~~~ / \ 
/ \ Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@slave1 ~]$ ls 11
total 1364
drwxrwxr-x 4 ec2-user ec2-user 34 Jan  7 09:58 remoting
-rw-rw-r-- 1 ec2-user ec2-user 1393083 Jan  7 09:58 remoting.jar
drwxrwxr-x 3 ec2-user ec2-user 20 Jan  7 10:54 workspace/
[ec2-user@slave1 workspace]$ ll
total 0
drwxrwxr-x 3 ec2-user ec2-user 71 Jan  7 10:54 slave1
[ec2-user@slave1 slave1]$ cd slave1/
[ec2-user@slave1 slave1]$ ls 11
total 5592
-rw-rw-r-- 1 ec2-user ec2-user 57323672 Jan  7 10:54 kubectl
-rw-rw-r-- 1 ec2-user ec2-user 913 Jan  7 10:54 mysql.yaml
-rw-rw-r-- 1 ec2-user ec2-user 606 Jan  7 10:54 wordpress.yaml
[ec2-user@slave1 slave1]$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
mysql-deployment-54fbcc85d9-t2sqg  1/1    Running   0          7m23s
mysql-deployment-54fbcc85d9-tjpwz  1/1    Running   0          7m23s
word-deployment-5855dbd89b-8k4j7   1/1    Running   0          7m23s
word-deployment-5855dbd89b-blxfb  1/1    Running   0          7m23s
[ec2-user@slave1 slave1]$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment  2/2    2           2           7m37s
word-deployment  2/2    2           2           7m37s
[ec2-user@slave1 slave1]$ kubectl get svc
NAME            TYPE      CLUSTER-IP  EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP  100.64.0.1  <none>        443/TCP       8m49s
mysql-service   LoadBalancer 100.67.167.147 a090902e6154a4eb88a0fbf967edb86d-1513568669.us-east-1.elb.amazonaws.com  3306:30095/TCP  7m42s
word-service    LoadBalancer 100.65.76.171 a4b5cc49f91bf45ac89bd6d31c771705-92087743.us-east-1.elb.amazonaws.com  80:32046/TCP  7m42s
[ec2-user@slave1 slave1]$

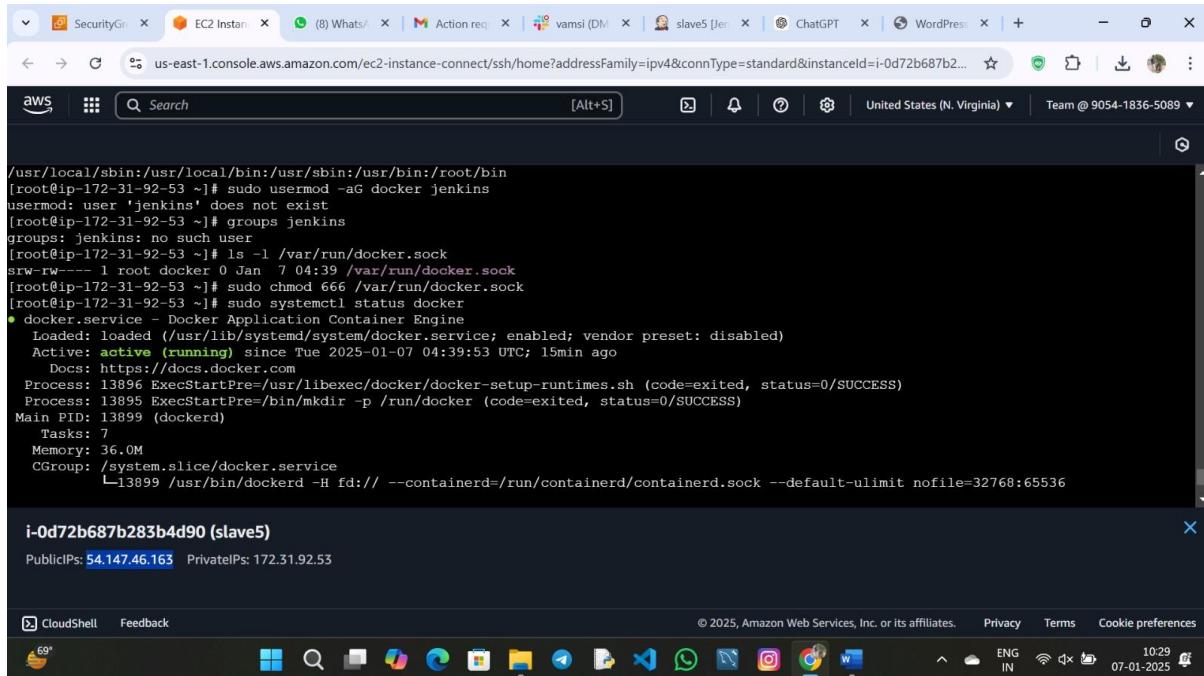
```

We can also see the external-ip(load balancer urls) in the terminal by checking the service using “kubectl get svc”.



This is the wordpress application deployed using Kubernetes.

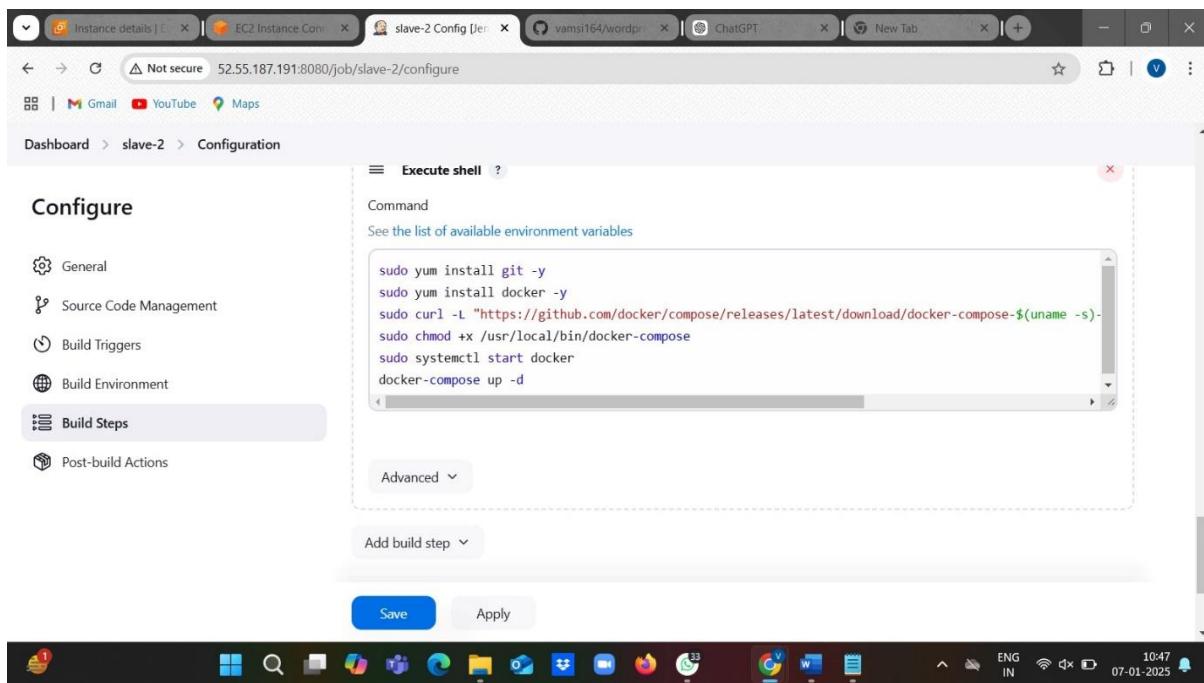
SLAVE-2:



```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@ip-172-31-92-53 ~]# sudo usermod -aG docker jenkins
usermod: user 'jenkins' does not exist
[root@ip-172-31-92-53 ~]# groups jenkins
groups: jenkins: no such user
[root@ip-172-31-92-53 ~]# ls -l /var/run/docker.sock
srw-rw--- 1 root docker 0 Jan 7 04:39 /var/run/docker.sock
[root@ip-172-31-92-53 ~]# sudo chmod 666 /var/run/docker.sock
[root@ip-172-31-92-53 ~]# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2025-01-07 04:39:53 UTC; 15min ago
     Docs: https://docs.docker.com
  Process: 13896 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
  Process: 13895 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 13899 (dockerd)
   Tasks: 7
  Memory: 36.0M
    CGroup: /system.slice/docker.service
           └─13899 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

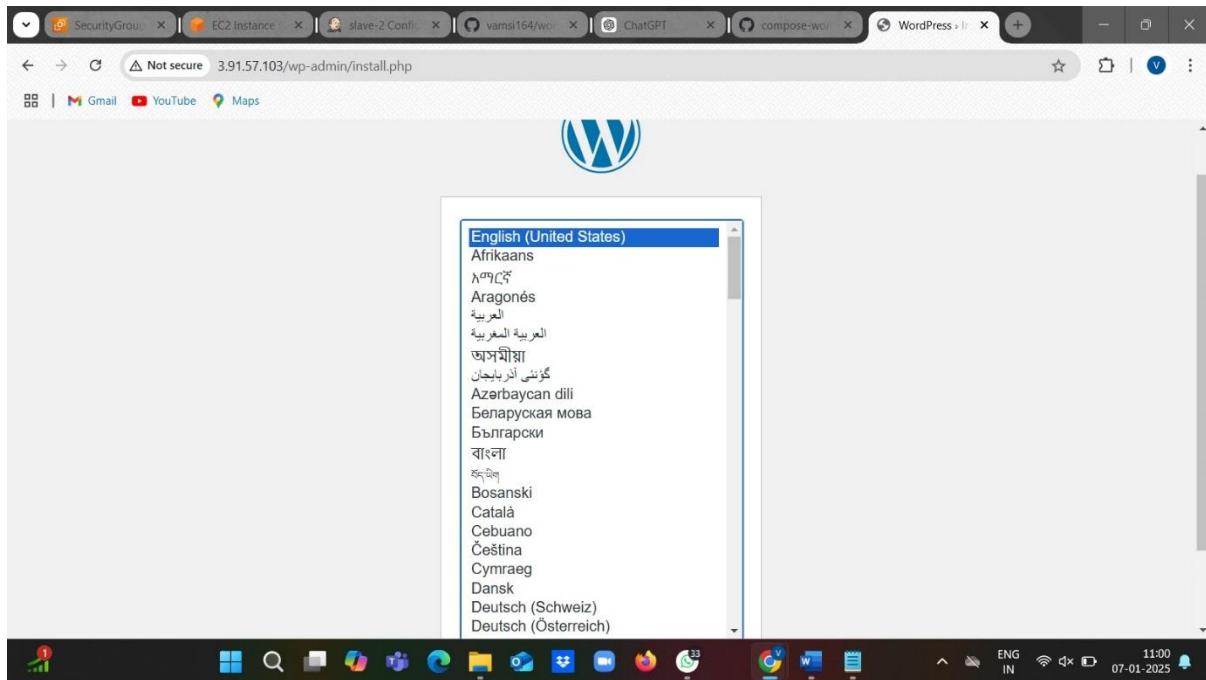
i-0d72b687b283b4d90 (slave5)
PublicIPs: 54.147.46.163 PrivateIPs: 172.31.92.53
```

Now the slave2 node is created by Vamsi and he connected the slave with the master and deploying the wordpress application using the docker.



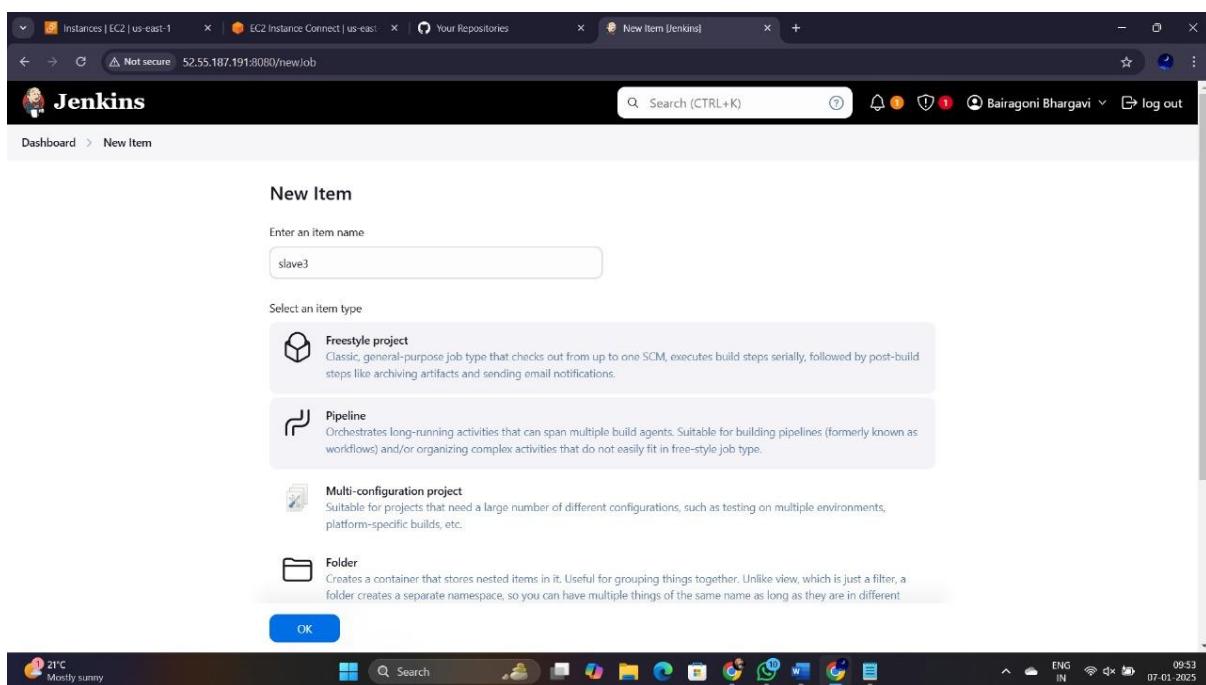
These are the commands used to install git, docker, docker compose and provided the required permissions and started the docker.

He deployed the wordpress by using the command “docker-compose up -d”.

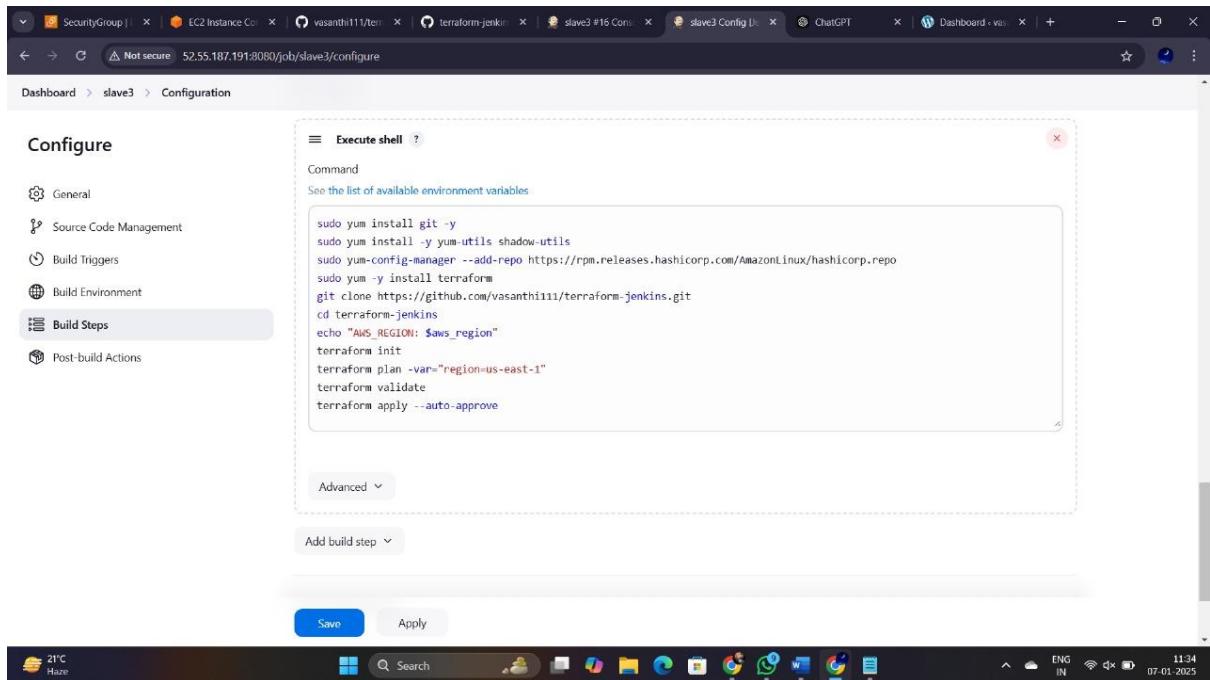


Now he had taken the public IP address of the slave and pasted it on the browser to check whether the app deployed successfully or not. It is deployed and we can able to see the application.

SLAVE-3:

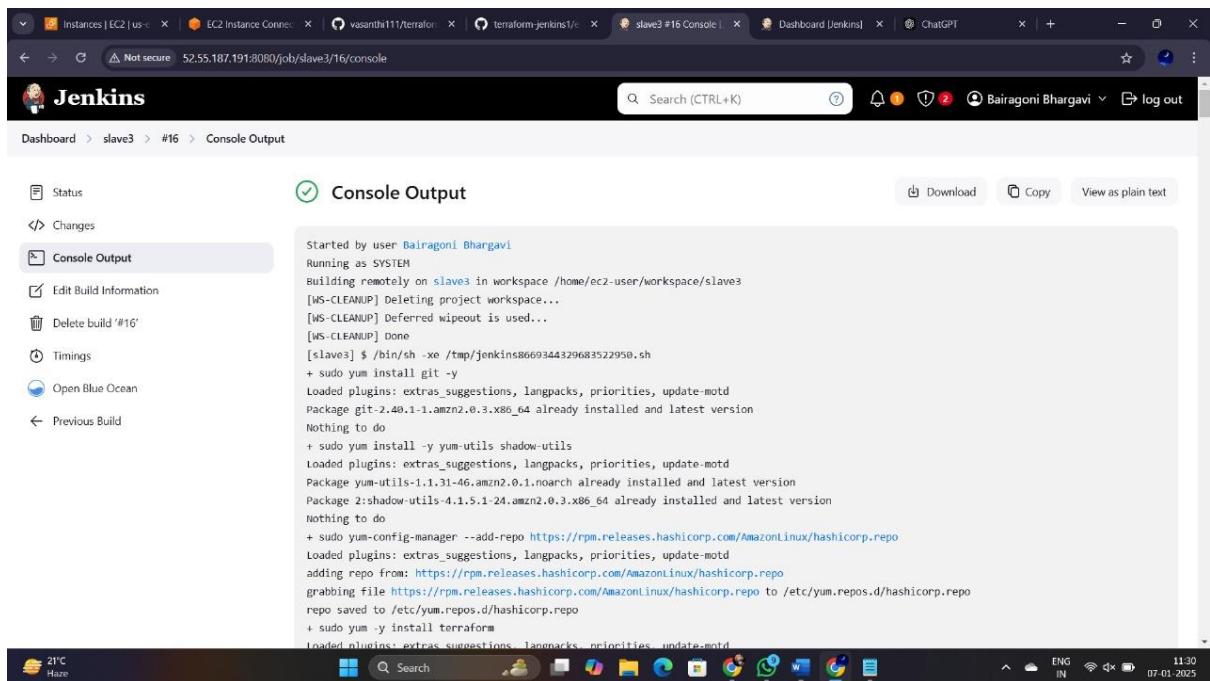


Vasantha created a slave3 job in Jenkins to deploy the wordpress using the terraform with manually and automatically.



These are the commands she had used in the execute shell to create the terraform files and deploy the application.

First she installed git, terraform, cloned the terraform files from github, provided the aws region.



Now she had written commands in execute shell and executed the build. The build is success and we can see the step by step process in the console output.

Slave-4:

The screenshot shows the Jenkins configuration page for job 'slave3'. Under 'Source Code Management', 'None' is selected. In the 'Build Triggers' section, 'Build periodically' is checked with the cron expression 'H/2 * * * *'. Other triggers like 'Trigger builds remotely' and 'Build after other projects are built' are unselected.

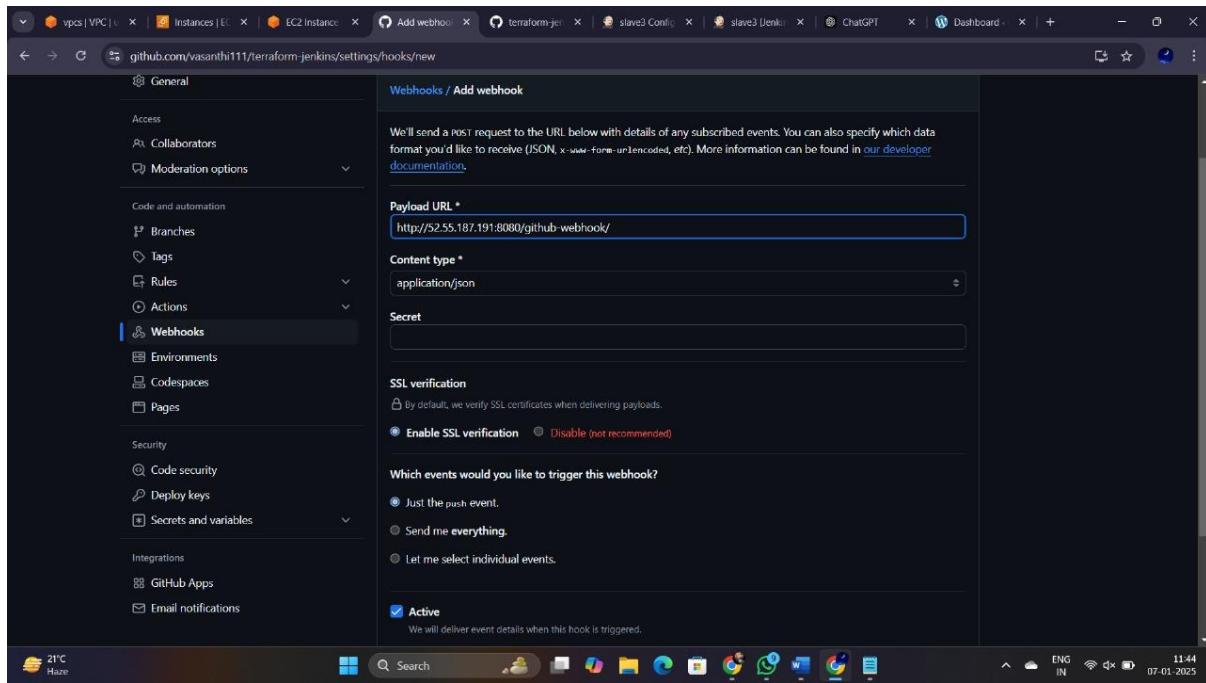
To automate the builds first she choosed the build periodically option with the syntax to deploy the application using terraform files.

The screenshot shows the Jenkins dashboard for job 'slave3'. The 'Builds' section displays the history of builds:

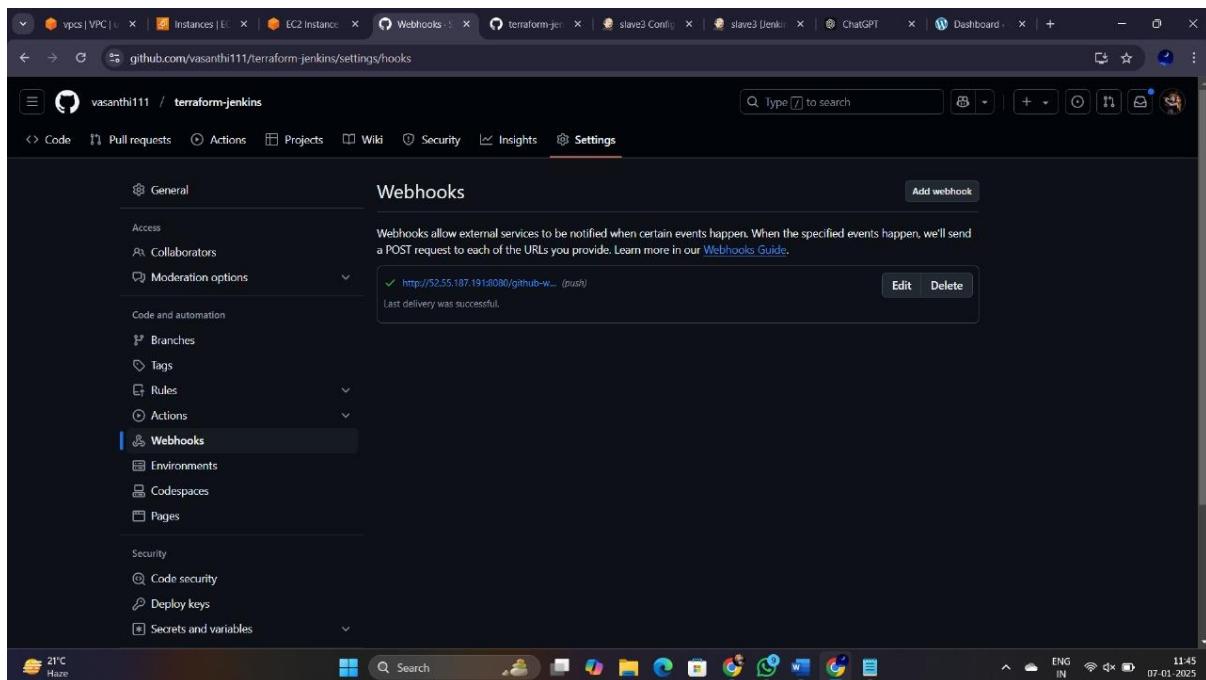
- Last build (#16), 6 min 11 sec ago
- Last stable build (#16), 6 min 11 sec ago
- Last successful build (#16), 6 min 11 sec ago
- Last failed build (#15), 9 min 16 sec ago
- Last unsuccessful build (#15), 9 min 16 sec ago
- Last completed build (#16), 6 min 11 sec ago

Build #17 is currently in progress (ETA: 16 sec).

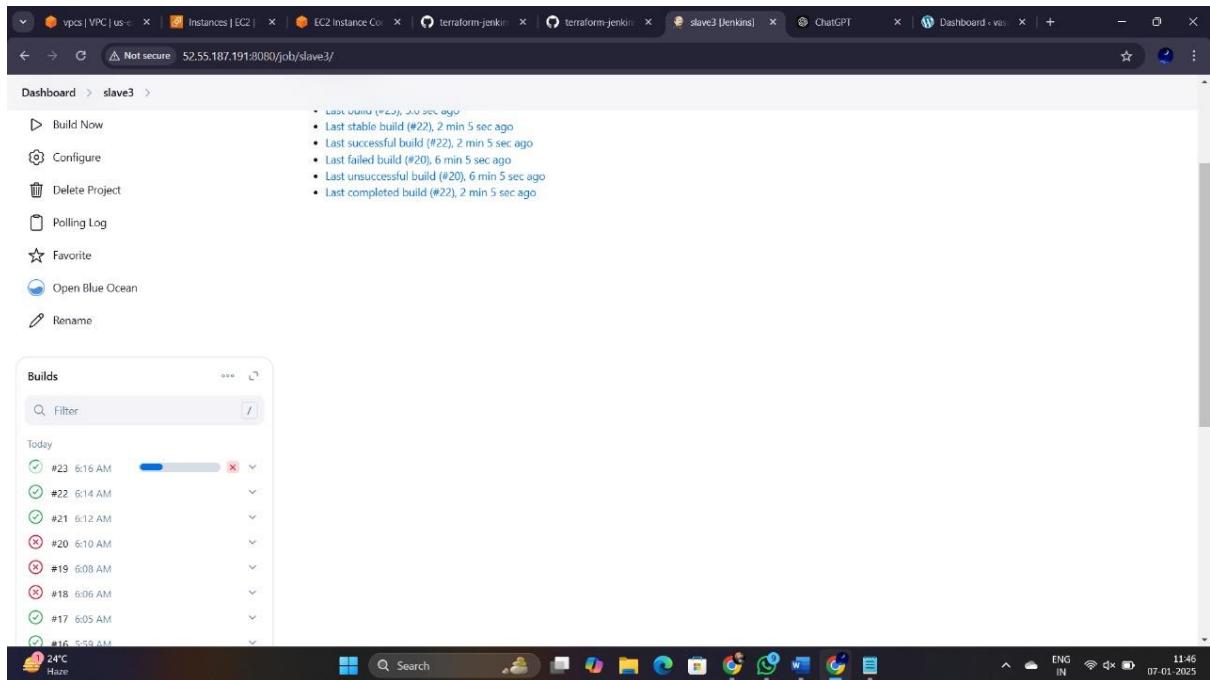
The builds are started automatically when she saved the above action. Now we can see builds are running related to provided the syntax.



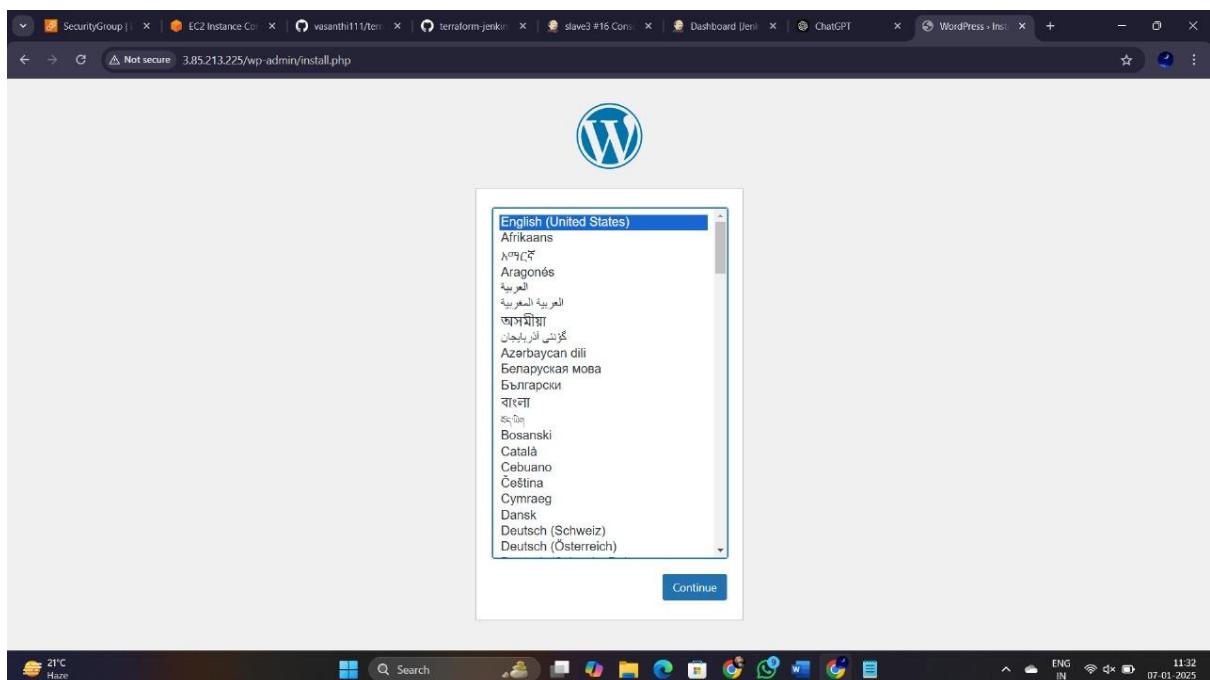
Creating Webhooks in the github where she is having the terraform files. She had provided the Jenkins url here to give updates to that project.



She had created a webhook in the github repository by selecting settings> webhooks> add webhooks to check with automatic deployment using the terraform.



She had written the commands in execute shell and performed the build using the poll scm. We can see the builds are occurring for every two minutes using github webhooks.



This is the wordpress application she had deployed using the terraform manually and automatically.

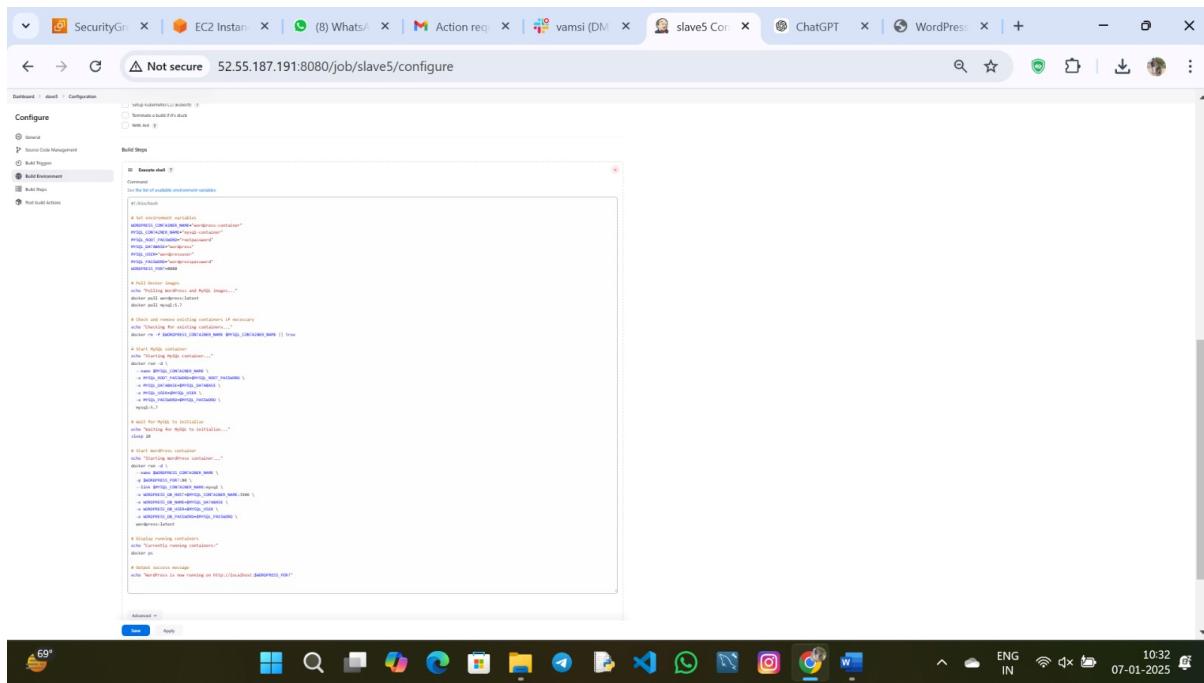
SLAVE-5:

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@ip-172-31-92-53 ~]# sudo usermod -aG docker jenkins
usermod: user 'jenkins' does not exist
[root@ip-172-31-92-53 ~]# groups jenkins
groups: jenkins: no such user
[root@ip-172-31-92-53 ~]# ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Jan 7 04:39 /var/run/docker.sock
[root@ip-172-31-92-53 ~]# sudo chmod 666 /var/run/docker.sock
[root@ip-172-31-92-53 ~]# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2025-01-07 04:39:53 UTC; 15min ago
     Docs: https://docs.docker.com
  Process: 13896 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
  Process: 13895 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 13899 (dockerd)
    Tasks: 7
   Memory: 36.0M
      CGroup: /system.slice/docker.service
              └─13899 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

i-0d72b687b283b4d90 (slave5)
PublicIPs: 54.147.46.163 PrivateIPs: 172.31.92.53
```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 10:29 07-01-2025

Now Balaji is deploying the wordpress application using docker compose file. For that he installed docker and docker compose. Provided the required permissions.



In execute shell he written the compose file which is having the mysql database engine and wordpress username with password, the command which is used to deploy the application i.e “docker-compose up -d”.

Jenkins

slave5

Status

Permalinks

- Last build (#1), 8 min 10 sec ago
- Last stable build (#1), 8 min 10 sec ago
- Last successful build (#1), 8 min 10 sec ago
- Last completed build (#1), 8 min 10 sec ago

Builds

Next he moved to slave5 job and executed the bulid by clicking on “build now” button. We can able to see that the build is success.

English (United States)

Afrikaans
ଅମ୍ବାଦ୍ର
Aragonés
العربية المغاربية
অসমীয়া
گۈنئى ئېرىپەن
Azerbaijani dili
Беларуская мова
Български
ସାଂକ୍ଷ୍ରତିକ
Bosanski
Català
Cebuano
Čeština
Cymraeg
Dansk
Deutsch (Sie)
Deutsch
Deutsch (Österreich)

Next he accessed the application using the public IP address of the slave5 server.

Same Account Creation

- In the context of master-slave systems, same account creation refers to the scenario where both the master and slave nodes belong to the same user account.
- **Use Case:** This is typical in setups where nodes reside in the same AWS account, cloud service, or system environment.
- **Advantages:**
 - Simplified management since all resources are under one account.
 - Easier to implement and maintain access policies and permissions.
- **Disadvantages:**
 - Single-point failure: If the account is compromised, all nodes are at risk.

METHOD: 5

In method 5 we are setting master slave configuration using vasanthi aws account. Here we are creating 5 slaves with one master. Vasanthi is master and she is setting the Jenkins.

Slave1: Deployed wordpress application using k8s by vasanthi

Slave2: Deployed wordpress application using docker-compose by Balaji

Slave3,4: Deployed wordpress application using terraform manually and automatically by Bhargavi.

Slave5: Deployed wordpress application using docker compose bash script by vamsi.

The screenshot shows the AWS EC2 Instances page with the following details:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Pul
master	i-03254fc8c557600648	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	ec2
slave1	i-01c373c5889b98783	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	ec2
slave2	i-0889be359b00bcfb3	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	ec2
slave3	i-012b50aa234c3639	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	ec2
slave5	i-0c9b1195deab1adc	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	ec2
nodes-us-east-1a.vasanthi.k8s.local	i-0e83b9129976f2aa9	Running	t2.medium	Initializing	View alarms +	us-east-1a	ec2
nodes-us-east-1a.vasanthi.k8s.local	i-05b1475f0d0f36697	Running	t2.medium	Initializing	View alarms +	us-east-1a	ec2
control-plane-us-east-1a.masters.vasanthi.k8s.local	i-049e1bbecca719bb7	Running	t2.medium	Initializing	View alarms +	us-east-1a	ec2

In this account 5 slave servers are created with one master server.

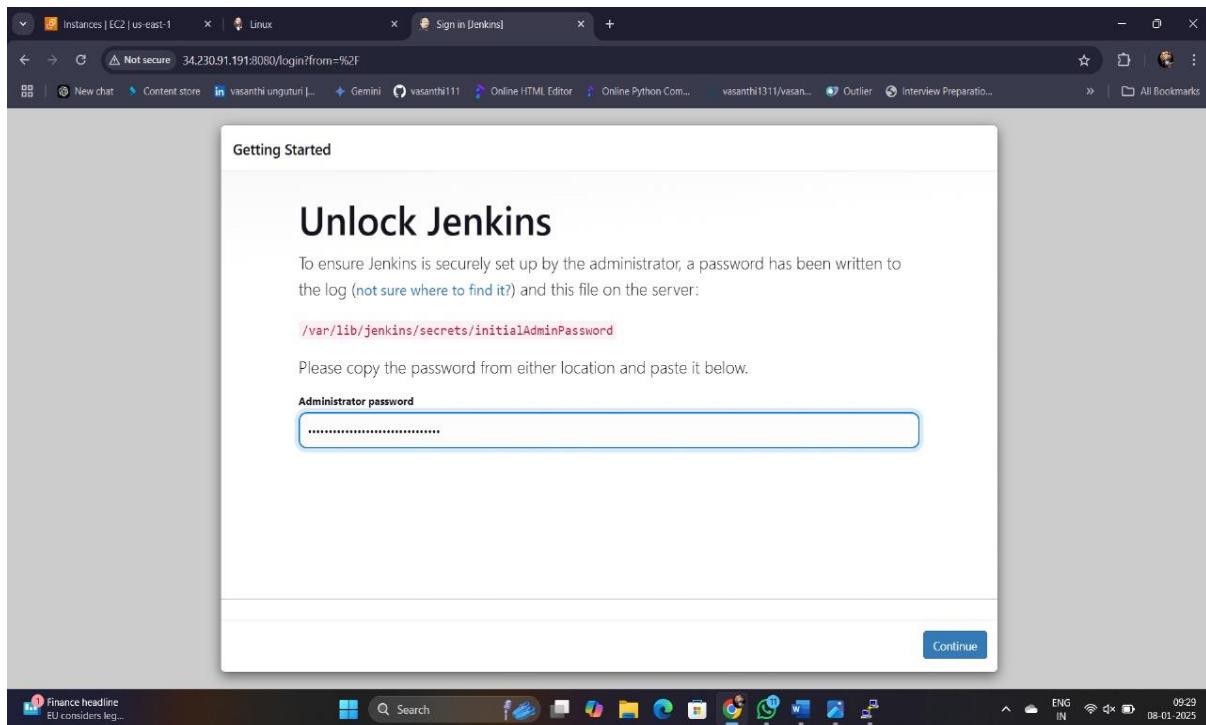
```

ec2-user@ip-172-31-24-50:~/ssh
$ login as: ec2-user
Authenticating with public key "imported-openssh-key"
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

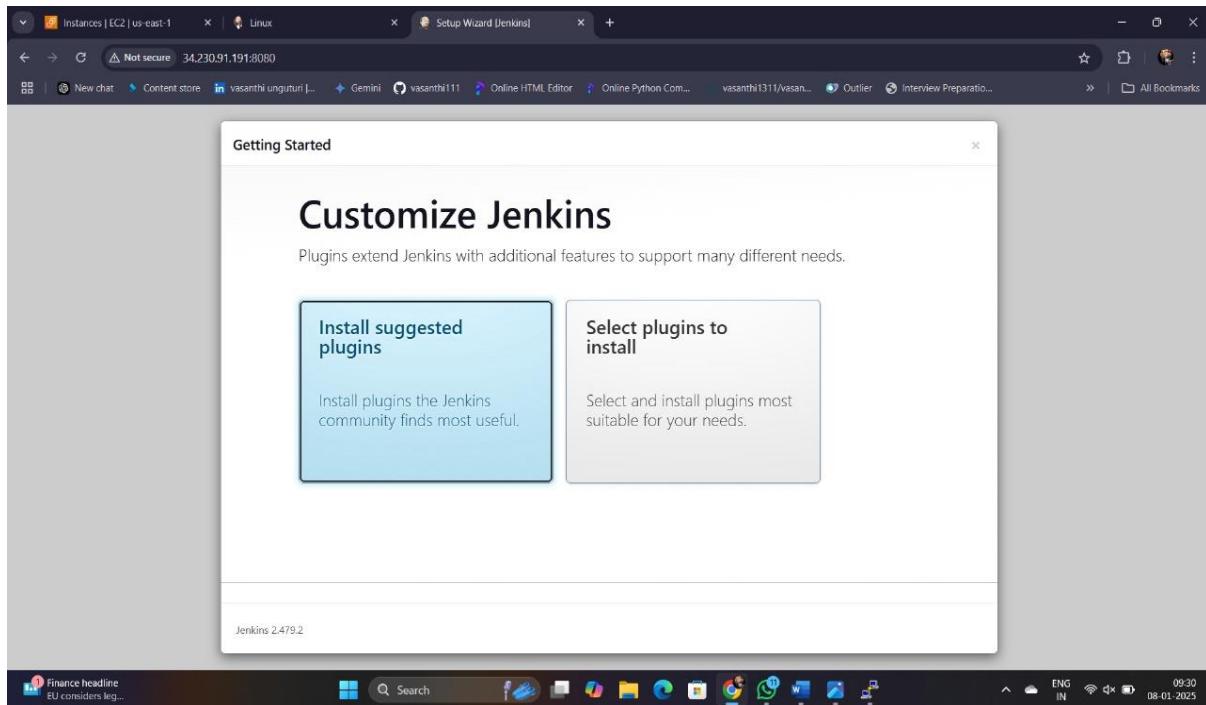
[ec2-user@ip-172-31-24-50 ~]$ sudo hostname master
[ec2-user@ip-172-31-24-50 ~]$ exec bash
[ec2-user@master ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ec2-user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ec2-user/.ssh/id_rsa.
Your public key has been saved in /home/ec2-user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0pKlt5OYbdKXl2Qwwwu4vgyrd0IPdAsJslxmV5cdY ec2-user@master
The key's randomart image is:
+---[RSA 2048]---+
| .++o |
| ..=oE |
| . .oo* .|
| + = .t*. |
| + S ..+=*. |
| = ..o* .|
| . ..o. |
| . .+oI |
| . .+oI |
+---[SHA256]---+
[ec2-user@master ~]$ sudo yum install java-17-amazon-corretto -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1 will be installed
--> Processing Dependency: java-17-amazon-corretto-headless(x86-64) = 1:17.0.13+11-1.amzn2.1 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libEXF11 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libfontconfig for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrender for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrandr for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXtst for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64

```

This is the master server. Here generated the key pairs using “ssh-keygen”. Installed the java 17 for Jenkins.



First Jenkins is installed and started by the master server.



Installed required plugins in Jenkins to create jobs.

SLAVE-1:

```

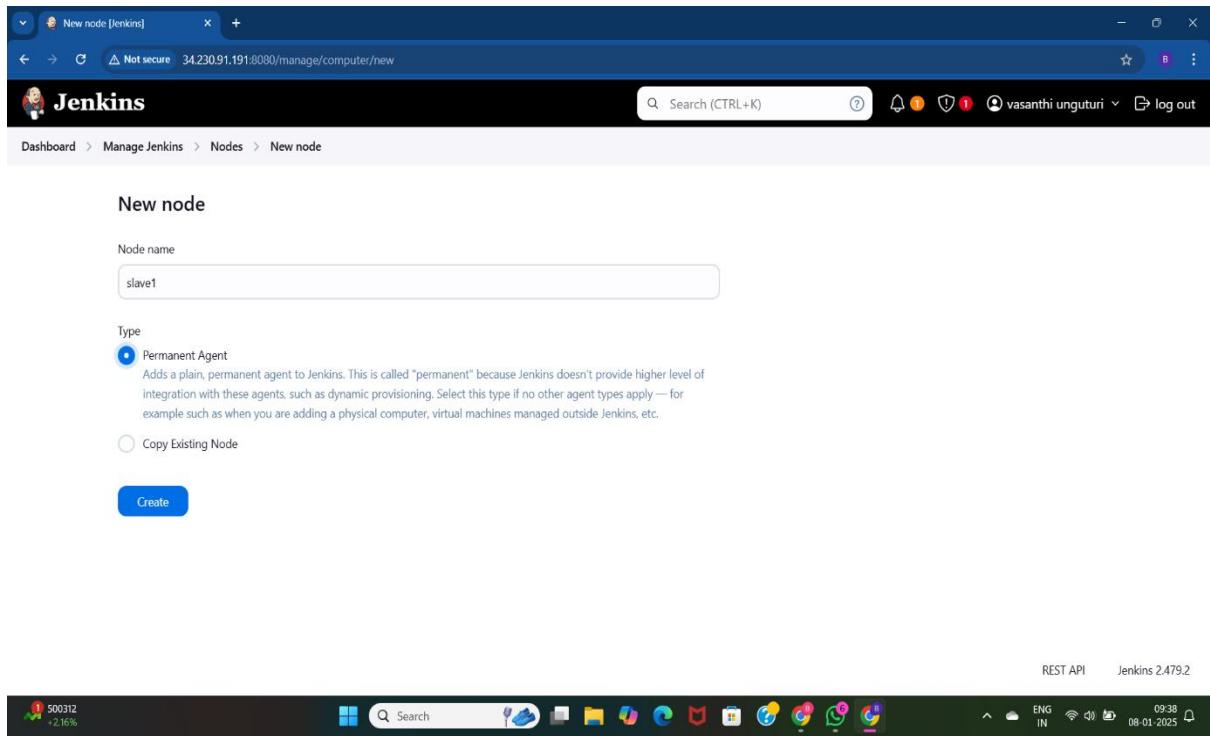
Transaction test succeeded
Running transaction
  Installing : git-core-2.40.1-1.amzn2.0.3.x86_64
  Installing : git-core-doc-2.40.1-1.amzn2.0.3.noarch
  Installing : perl-Error-0.17020-2.amzn2.noarch
  Installing : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64
  Installing : perl-Git-2.40.1-1.amzn2.0.3.noarch
  Installing : git-2.40.1-1.amzn2.0.3.x86_64
  Verifying : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64
  Verifying : git-2.40.1-1.amzn2.0.3.x86_64
  Verifying : perl-Error-0.17020-2.amzn2.noarch
  Verifying : git-core-2.40.1-1.amzn2.0.3.x86_64
  Verifying : git-core-doc-2.40.1-1.amzn2.0.3.noarch
  Verifying : perl-git-2.40.1-1.amzn2.0.3.noarch

Installed:
  git.x86_64 0:2.40.1-1.amzn2.0.3

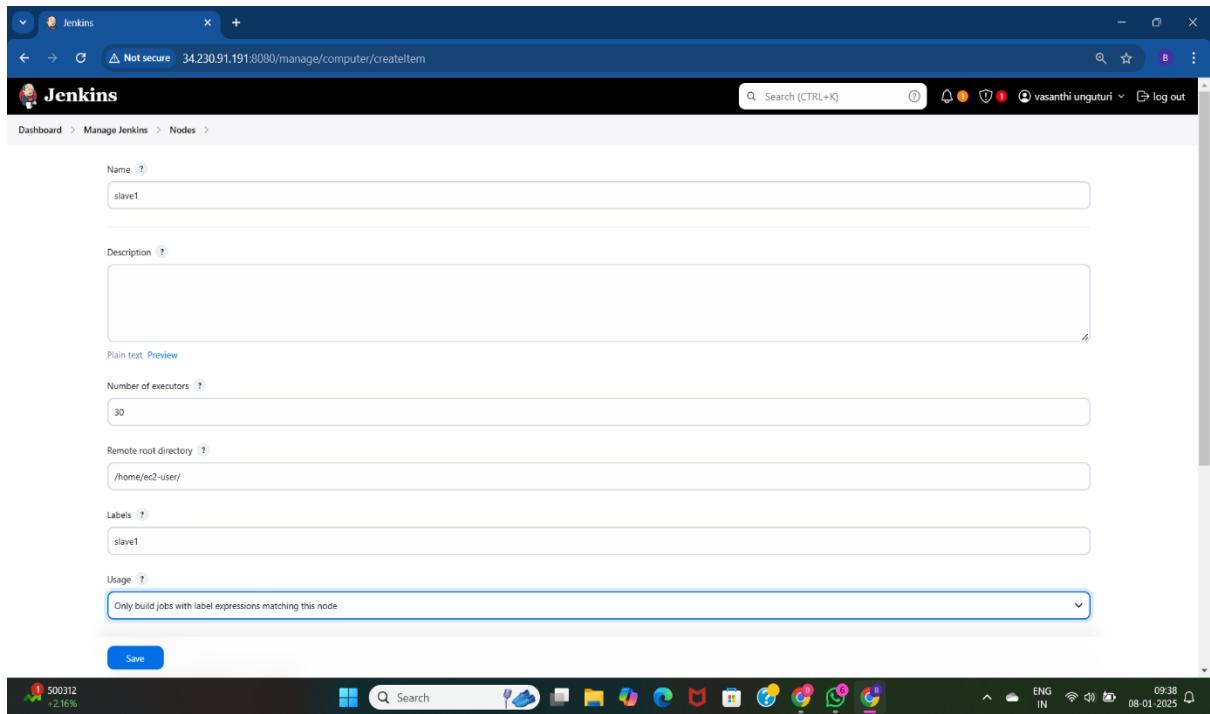
Dependency Installed:
  git-core.x86_64 0:2.40.1-1.amzn2.0.3           git-core-doc.noarch 0:2.40.1-1.amzn2.0.3           perl-Error.noarch 1:0.17020-2.amzn2           perl-Git.noarch 0:2.40.1-1.amzn2.0.3

Complete!
[ec2-user@ip-172-31-19-244 ~]$ history
 1 cd .ssh/
 2 sudo vi authorized_keys
 3 cd
 4 sudo yum install java-17-amazon-corretto -y
 5 sudo yum install git -y
 6 history
[ec2-user@ip-172-31-19-244 ~]$ i-01c373c5889b98783 (slave1)
PublicIPs: 54.226.180.170 PrivateIPs: 172.31.19.244
  
```

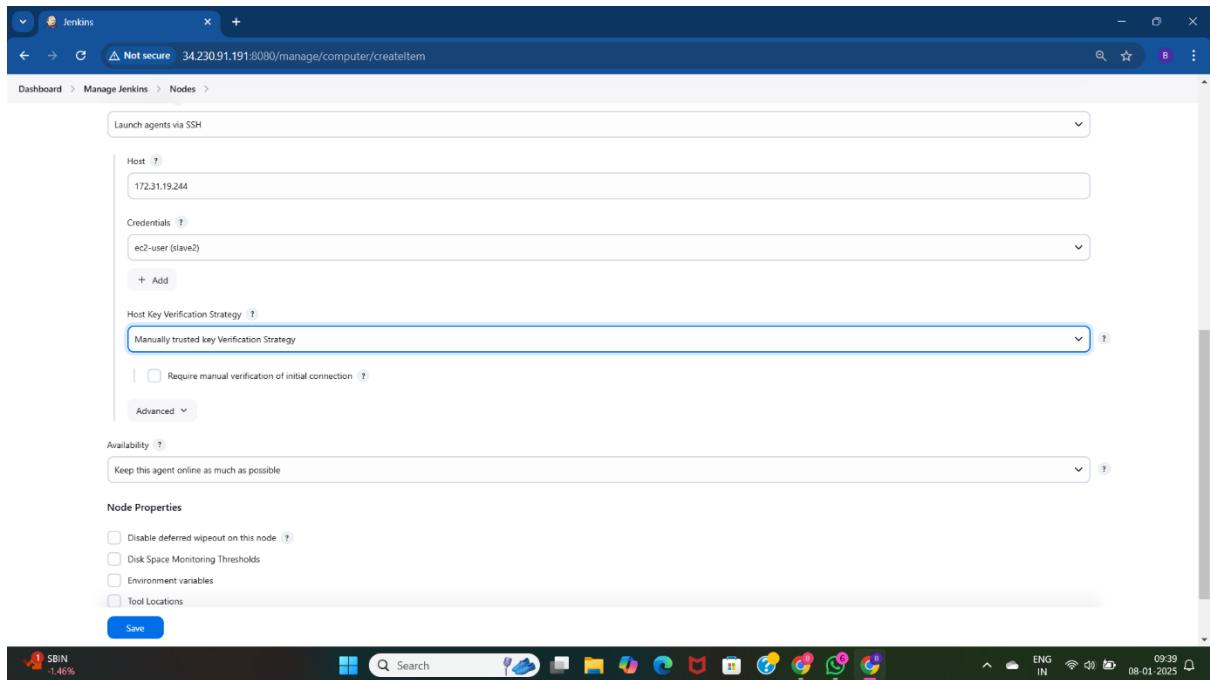
Installed the above mentioned commands here.



Creating a slave1 node in Jenkins by moving into manage Jenkins> nodes > give name for node amd select permanent agent and click on create.



Now give name for the node with number of executors, remote root directory, label name and usage.

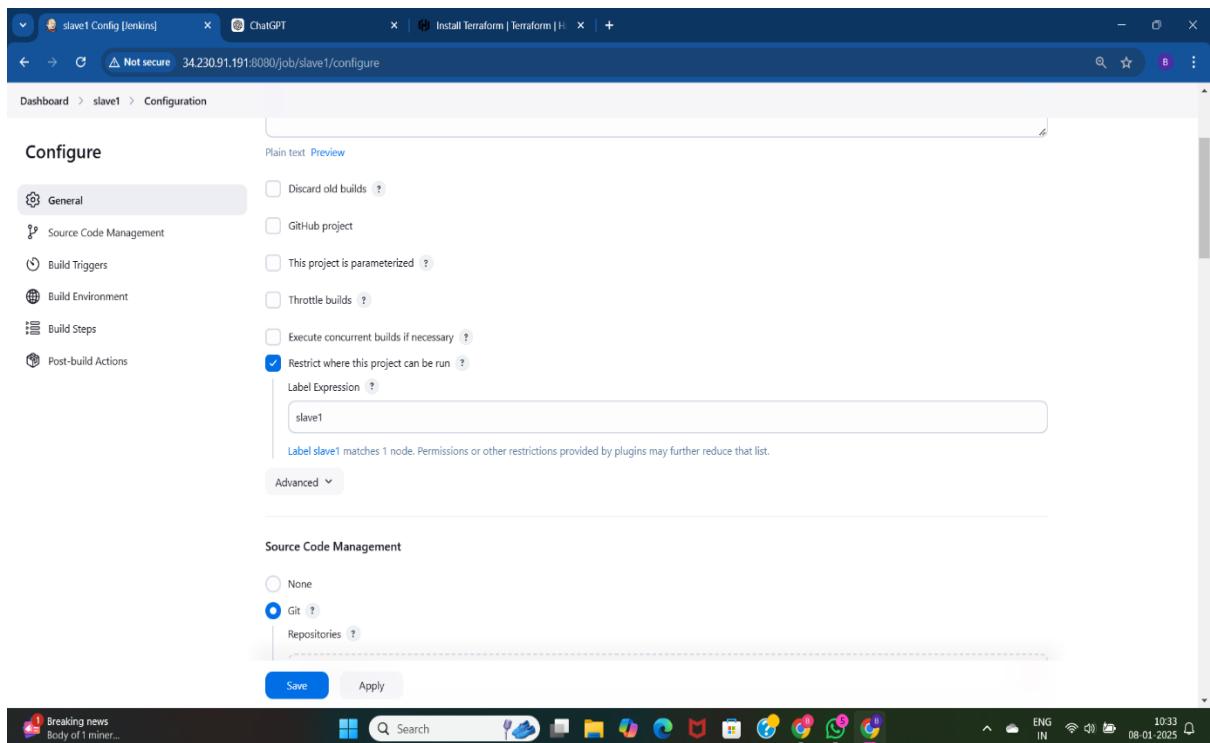


Provide the host as private ip address of the slave server, give credentials to access the slave server and select the key verification strategy.

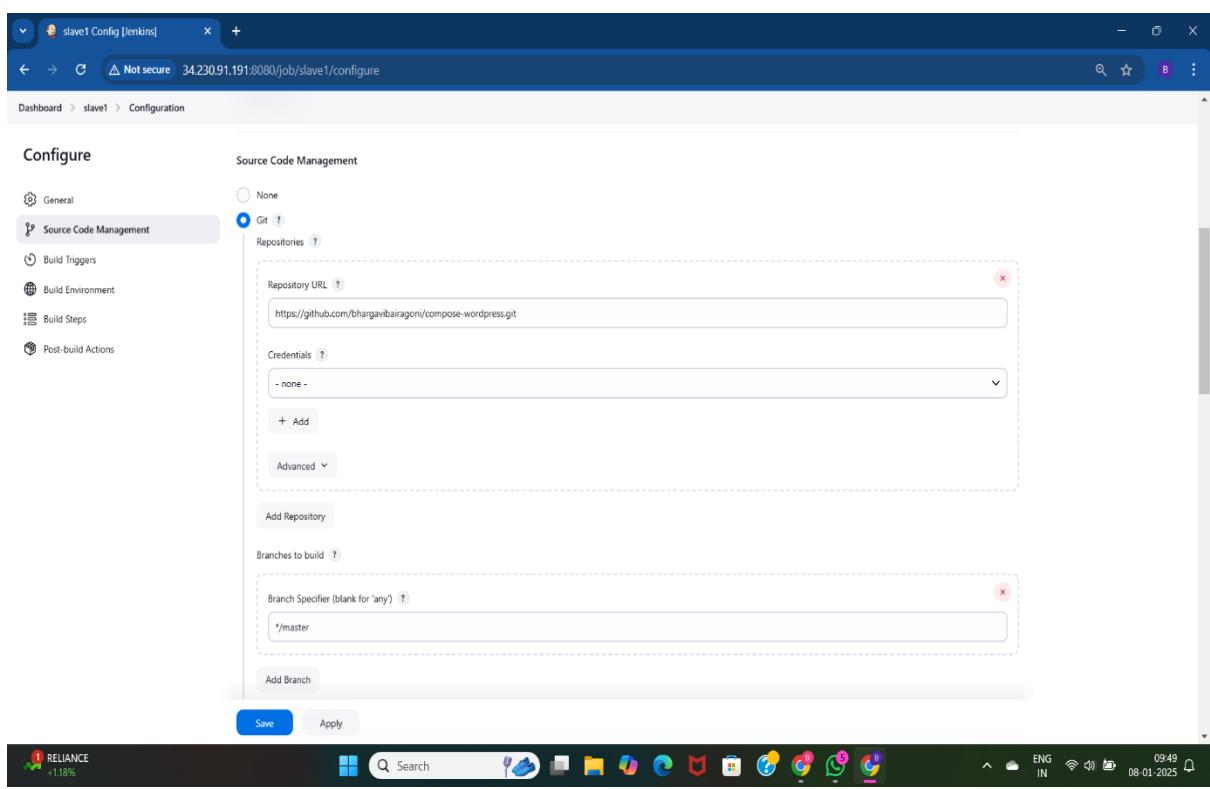
S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	Built-In Node	Linux (amd64)	In sync	5.47 GiB	0.8 GiB	5.47 GiB	0ms
2	slave1	Linux (amd64)	In sync	5.90 GiB	0.8 GiB	5.90 GiB	25ms
3	slave2	Linux (amd64)	In sync	5.62 GiB	0.8 GiB	5.62 GiB	22ms

The screenshot shows the Jenkins 'Nodes' page. It lists three nodes: 'Built-In Node', 'slave1', and 'slave2'. The 'slave2' node is currently 'Data obtained'. The table includes columns for Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. A 'New Node' button is available at the top right.

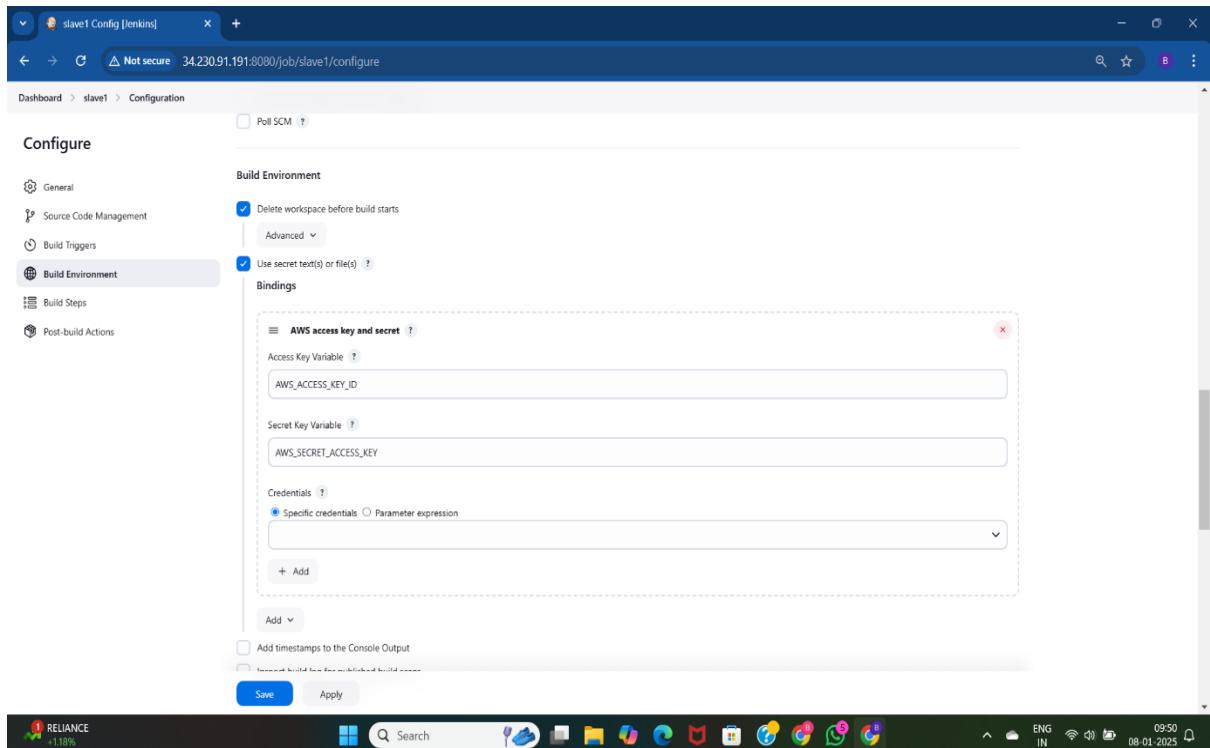
Here we can see two nodes are created.



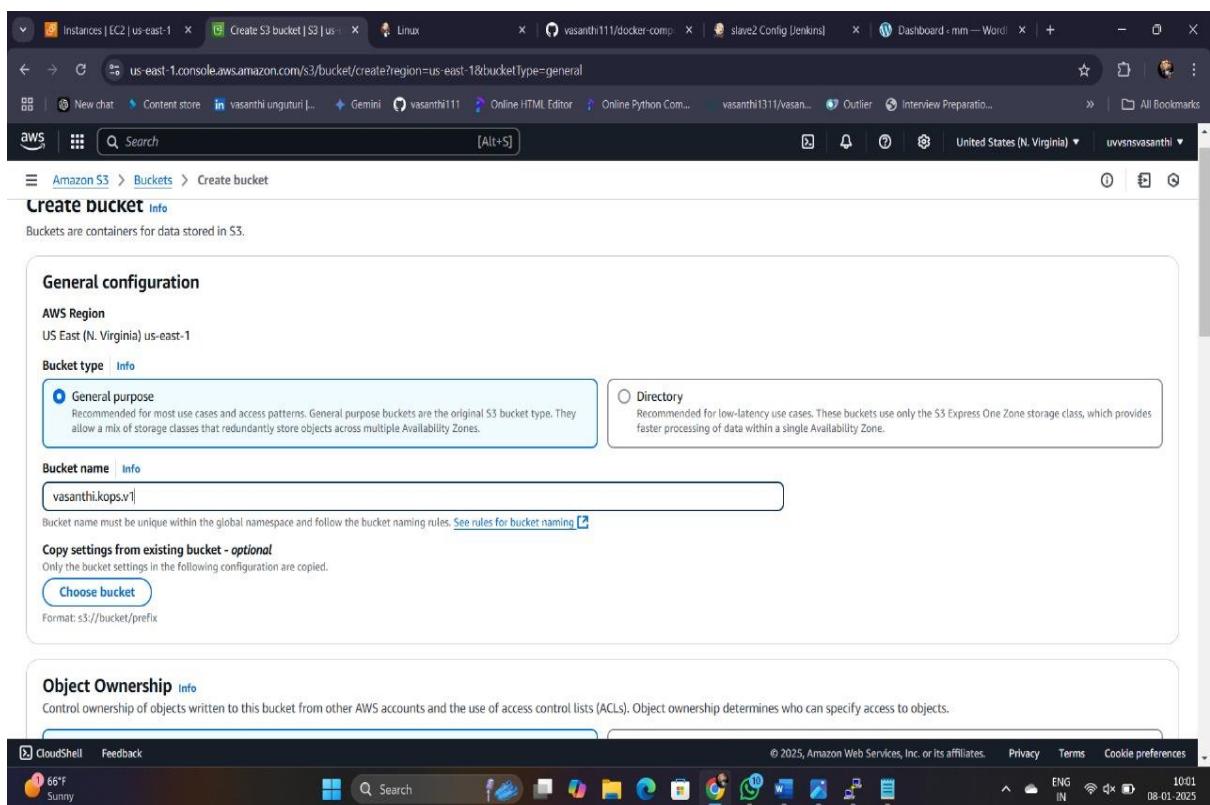
Create a Jenkins free style project, select the created label to execute builds.



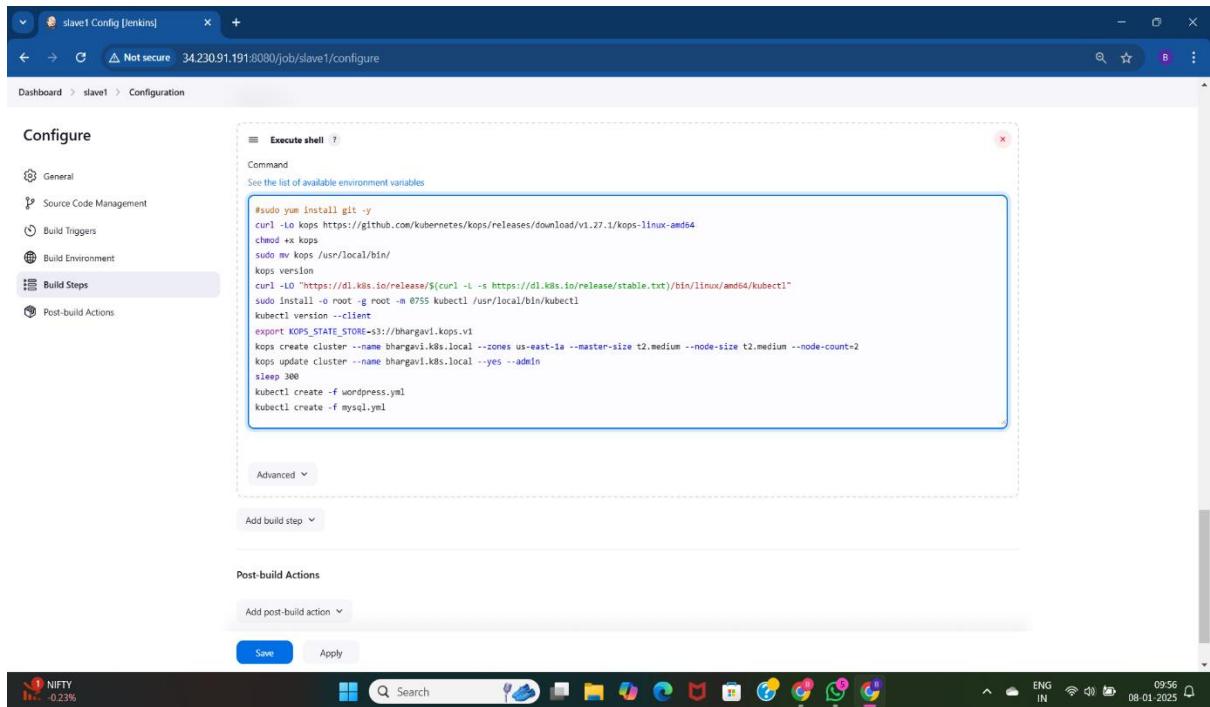
Give the gihub repository url and branch name to access the code and deploy the application.



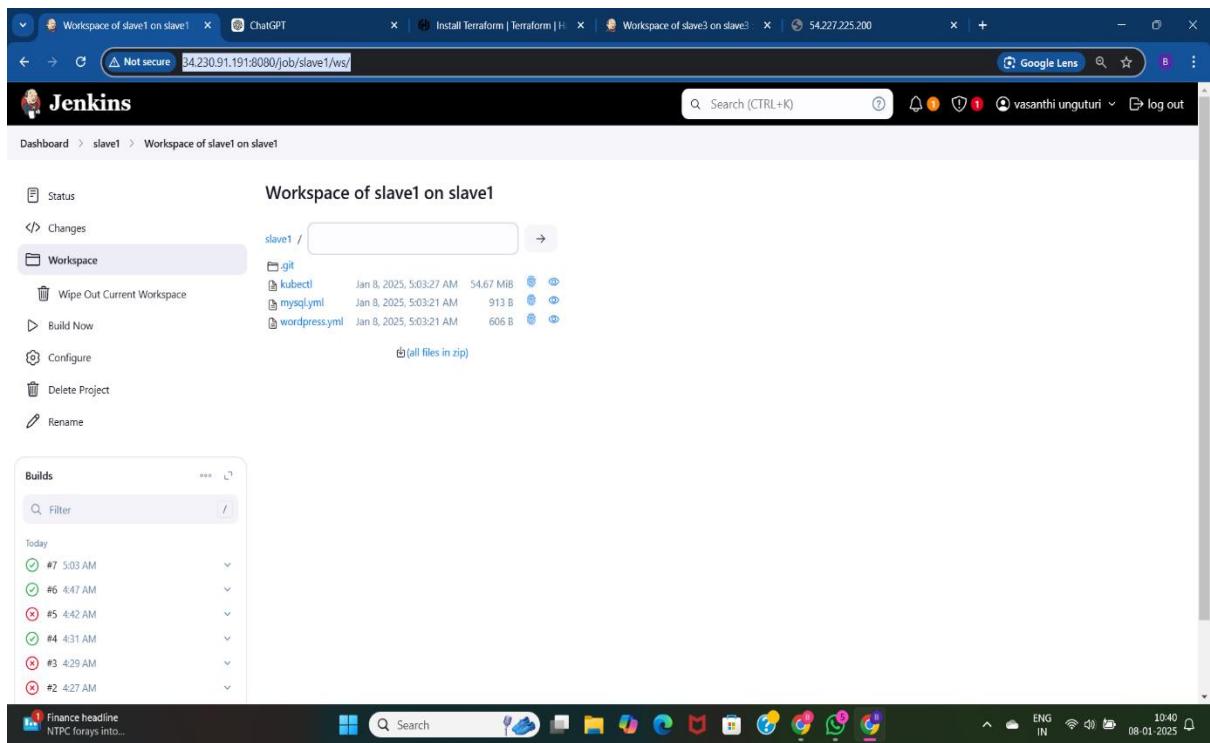
Here give the aws credentials by installing aws credentials plugin. It is used for other slaves also.



Now create a s3 bucket to deploy the application using k8s.



In execute shell write commands to install kops, kubectl, and export the created s3 bucket to store the data. Create the clusters using kops and create the deployment and service files.



The build is success and we can see the files related to deployment and service of the wordpress application.

The screenshot shows the Jenkins interface with the following details:

- Project Path:** slave1 #7 Console [Jenkins]
- Build Number:** #7
- Console Output:**

```

Started by user vasanthi unguturi
Running as SYSTEM
Building remotely on slave1 in workspace /home/ec2-user/workspace/slave1
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/bhargavibairagi/compose-wordpress.git
> git init /home/ec2-user/workspace/slave1 # timeout=10
Fetching upstream changes from https://github.com/bhargavibairagi/compose-wordpress.git
> git --version # timeout=10
> git -version # 'git' version 2.40.1
> git fetch --tags --force --progress -- https://github.com/bhargavibairagi/compose-wordpress.git +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url https://github.com/bhargavibairagi/compose-wordpress.git # timeout=10
> git config core.sparsecheckout # timeout=10
> git checkout -f d0539fa4d17a2f11585bd8e0915aa937d61efd # timeout=10
Commit message: "Update wordpress.yml"
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision d0539fa4d17a2f11585bd8e0915aa937d61efd (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f d0539fa4d17a2f11585bd8e0915aa937d61efd # timeout=10
Commit message: "Update wordpress.yml"
> git rev-list --no-walk d0539fa4d17a2f11585bd8e0915aa937d61efd # timeout=10
[slave1]$ /bin/sh -xe /tmp/jenkins10614516033689176798.sh
+ curl -Lo kops https://github.com/kubernetes/kops/releases/download/v1.27.1/kops-linux-amd64
+ Total % Received % Xferd Average Speed Time Time Current
          0   0   0  0  0  0 --:--:-- --:--:-- --:--:--
```
- System Status:** Breaking news: Body of 1 miner...
- System Bar:** Shows ENG IN, 10:33, 08-01-2025.

This is the step by step process of the build which is executing every step mentioned in the execute shell.

The screenshot shows the AWS CloudWatch Load Balancers console with the following details:

- Region:** us-east-1
- Load Balancers:** (1/3) **Load balancers (1/3)**
- Table Headers:** Name, DNS name, State, VPC ID, Availability Zones, Type, Date created
- Data:**

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
aca589681d1804aa781...	aca589681d1804aa781a29...	-	vpc-0660d82ef4d56fcda	us-east-1a (use1-az1)	classic	January 8, 2025, 10:08 (UTC+05:30)
a0e1ba0e5c664404895...	a0e1ba0e5c664404895386...	-	vpc-0660d82ef4d56fcda	us-east-1a (use1-az1)	classic	January 8, 2025, 10:08 (UTC+05:30)
api-vasanthi-k8s-local-...	api-vasanthi-k8s-local-pbqs...	Active	vpc-0660d82ef4d56fcda	us-east-1a (use1-az1)	network	January 8, 2025, 10:02 (UTC+05:30)
- Selected Load Balancer:** aca589681d1804aa781a29e971d33a1e
- Details Tab:**
 - Load balancer type:** Classic
 - Status:** 1 of 1 instance in service
 - VPC:** vpc-0660d82ef4d56fcda
 - Date created:** January 8, 2025, 10:08 (UTC+05:30)
 - Scheme:** Internet-facing
 - Hosted zone:** Z355XDTRQ7X7K
 - Availability Zones:** subnet-0da9c419870f1eede us-east-1a (use1-az1)
- System Bar:** Shows CloudShell, Feedback, Air: Moderate Now, ENG IN, 10:10, 08-01-2025.

The load balancers are created. Now we can access the application using these DNS names of loadbalancers.

The screenshot shows the AWS CloudWatch Metrics console. At the top, there are tabs for 'Metrics' and 'Logs'. Below the tabs, a search bar and a date range selector ('From: 08-01-2025 To: 08-01-2025') are visible. The main area displays a table of metric data with columns: Metric Name, Value, Unit, and Time. One row is highlighted in blue, showing a value of 1.0 for the 'Value' column.

Auto scaling group is also created with the clusters to generate new instances while the load/traffic is increasing.

The screenshot shows a Windows desktop environment. A browser window is open, displaying a WordPress setup configuration page. The URL in the address bar is aca589601d1804aa781a29e971d33a1e.1095903828.us-east-1.elb.amazonaws.com/wp-admin/setup-config.php. A dropdown menu for language selection is open, showing various options including English (United States), Afrikaans, অসমীয়া, Aragonés, العربية, বাংলা, Bosnian, Català, Cebuano, Čeština, Cymraeg, Dansk, Deutsch (Schweiz), and Deutsch (Österreich). A 'Continue' button is at the bottom of the dropdown.

I have taken one of the load balancers DNS and accessed the wordpress application.

SLAVE-2:

```

root@slave2:~#
Dependency Installed:
containerd.x86_64 0:1.7.23-1.amzn2.0.2           libcgroup.x86_64 0:0.41-21.amzn2                  pigz.x86_64 0:2.3.4-1.amzn2.0.1                runc.x86_64 0:1.1.14-1.amzn2

Complete!
[root@slave2 ~]# sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time   Time  Current
                                 Dload  Upload Total Spent  Left Speed
0     0     0     0     0     0     0 --:--:-- --:--:-- --:--:-- 0
0     0     0     0     0     0     0 --:--:-- --:--:-- --:--:-- 0
67 61.7M  67 41.8M  0     0  27.6M  0 --:--:-- --:--:-- 77.0M
[root@slave2 ~]# curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time   Time  Current
                                 Dload  Upload Total Spent  Left Speed
0     0     0     0     0     0     0 --:--:-- --:--:-- --:--:-- 0
0     0     0     0     0     0     0 --:--:-- --:--:-- --:--:-- 0
100 61.7M 100 61.7M  0     0  90.8M  0 --:--:-- --:--:-- 90.8M
[root@slave2 ~]# sudo chmod +x /usr/local/bin/docker-compose
[root@slave2 ~]# docker-compose --version
docker Compose version v3.2.2
[root@slave2 ~]# sudo usermod -aG docker ec2-user
[root@slave2 ~]# sudo in -s /usr/local/bin/docker-compose /usr/bin/docker-compose
[root@slave2 ~]# sudo systemctl start docker
[root@slave2 ~]# sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
  Active: active (running) since Wed 2025-01-08 04:20:18 UTC; 159 ago
    Docs: https://docs.docker.com
      Process: 12426 ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
      Main PID: 12429 (dockerd)
        Tasks: 7
       Memory: 36.2M
      CGroup: /system.slice/docker.service
             └─[12429 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Jan 08 04:20:18 slave2 systemd[1]: Starting Docker Application Container Engine...
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.302376346Z" level=info msg="Starting up"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.320901523Z" level=info msg="Loading containerd"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.542178441Z" level=info msg="Loading container PROJECT_4512 - Protected Volume"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.557105302Z" level=warning msg="WARNING: bridge"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.557513532Z" level=warning msg="WARNING: bridge"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.557710390Z" level=info msg="Docker daemon" c...
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.557931342Z" level=info msg="Daemon has completed initialization"
Jan 08 04:20:18 slave2 dockerd[12429]: time="2025-01-08T04:20:18.594290706Z" level=info msg="API listen on /run/dockerd/12429"
Jan 08 04:20:18 slave2 systemd[1]: Started Docker Application Container Engine.
Hint: Some lines were ellipsized, use -l to show in full.
[root@slave2 ~]#

```

The screenshot shows a terminal window with a black background and white text. It displays the output of several commands related to Docker and its configuration. At the bottom, there's a taskbar with various icons and system status indicators like battery level (66%), weather (Sunny), and network.

In slave2 we need to deploy the wordpress application using the docker and docker compose. Provided the permissions for docker compose. Started the docker service.

```

#includedir /etc/sudoers.d
jenkins ALL=(ALL) NOPASSWD:ALL
-- INSERT --

```

The screenshot shows a terminal window with a black background and white text. It displays the contents of the `/etc/sudoers.d/jenkins` file, which contains a single line: `jenkins ALL=(ALL) NOPASSWD:ALL`. Below this line, there is a placeholder for additional commands, indicated by three dashes and the word "INSERT". At the bottom, there's a taskbar with various icons and system status indicators like battery level (66%), weather (Sunny), and network.

In Jenkins give the sudo permissions to install any required packages or softwares in the slave servers.

The screenshot shows the Jenkins configuration page for a job named 'slave2'. In the 'General' section, the 'Restrict where this project can be run' checkbox is selected, and the 'Label Expression' field contains 'slave2'. Under 'Source Code Management', the 'Get' option is chosen, and a GitHub repository URL is entered: <https://github.com/vasanthi111/docker-compose-project4.git>. The Jenkins interface includes a toolbar at the top and a taskbar at the bottom.

Select the created label and give the github repository url here.

The screenshot shows the Jenkins dashboard for the 'slave2' job. The status is listed as 'slave2' with a green circle icon. The 'Build Now' button is visible. The 'Builds' section shows two recent builds: #2 (04:28) and #1 (04:25). The build history on the right lists several builds, all marked as 'Last build' or 'Last stable build' with a timestamp of '59 sec ago'. The Jenkins interface includes a toolbar at the top and a taskbar at the bottom.

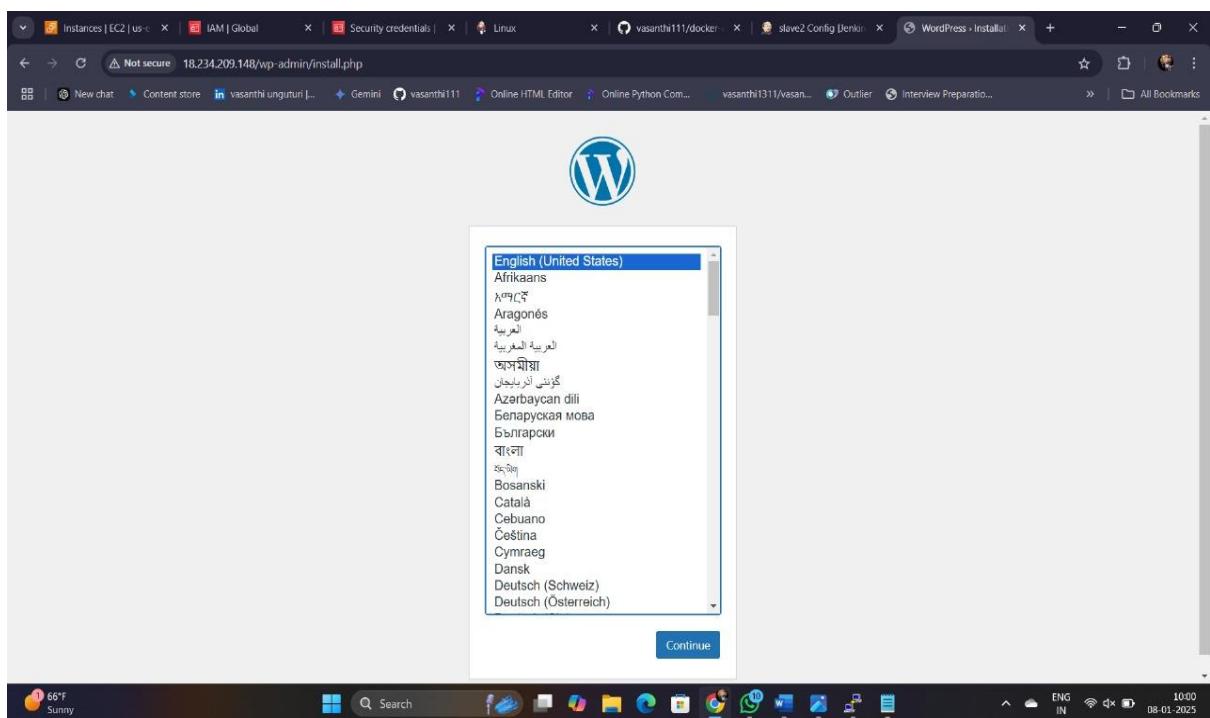
Now execute the build and check the status of the build.

```

Started by user vasanthi unguturi
Running as SYSTEM
Building remote on slave2 in workspace /home/ec2-user/workspace/slave2
The following git tool is: NONE
No credentials specified
Cloning the remote Git repository
  Cloning repository https://github.com/vasanthi111/docker-compose-project4.git
    > git init /home/ec2-user/workspace/slave2 # timeout=10
  Fetching upstream changes from https://github.com/vasanthi111/docker-compose-project4.git
    > git --version # timeout=10
    > git --version # 'git' version 2.46.1
    > git fetch --tags --force --progress -- https://github.com/vasanthi111/docker-compose-project4.git +refs/heads/*:refs/remotes/origin/* # timeout=10
    > git config remote.origin.url https://github.com/vasanthi111/docker-compose-project4.git # timeout=10
    > git config remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
  Avoid second fetch
    > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
  Checking out revision f9643047e4210ecfc13b3b70d63372f3467 (refs/remotes/origin/main)
    > git config core.sparsecheckout # timeout=10
    > git checkout -f f9643047e4210ecfc13b3b70d63372f3467 # timeout=10
Commit message: "Update docker-compose.yml"
First time build. Skipping changelog.
[slave2] $ /bin/sh -xe /tmp/jenkins38920317797736164.sh
+ docker-compose down -v
time=2025-01-08T09:46:53Z level=warning msg="* /home/ec2-user/workspace/slave2/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
Volume slave2_m_data Removing
Volume slave2_m_data Removed
+ docker-compose/migrations/docker-compose up -d
time=2025-01-08T09:46:53Z level=warning msg="* /home/ec2-user/workspace/slave2/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
on Pulling
wordpress Pulling
emacsc391987 Pulling fs layer
bb54da79fed Pulling fs layer
972688f02079 Pulling fs layer
48f5f589717 Pulling fs layer
1c3df560485 Pulling fs layer
emo13c0800fe Pulling fs layer

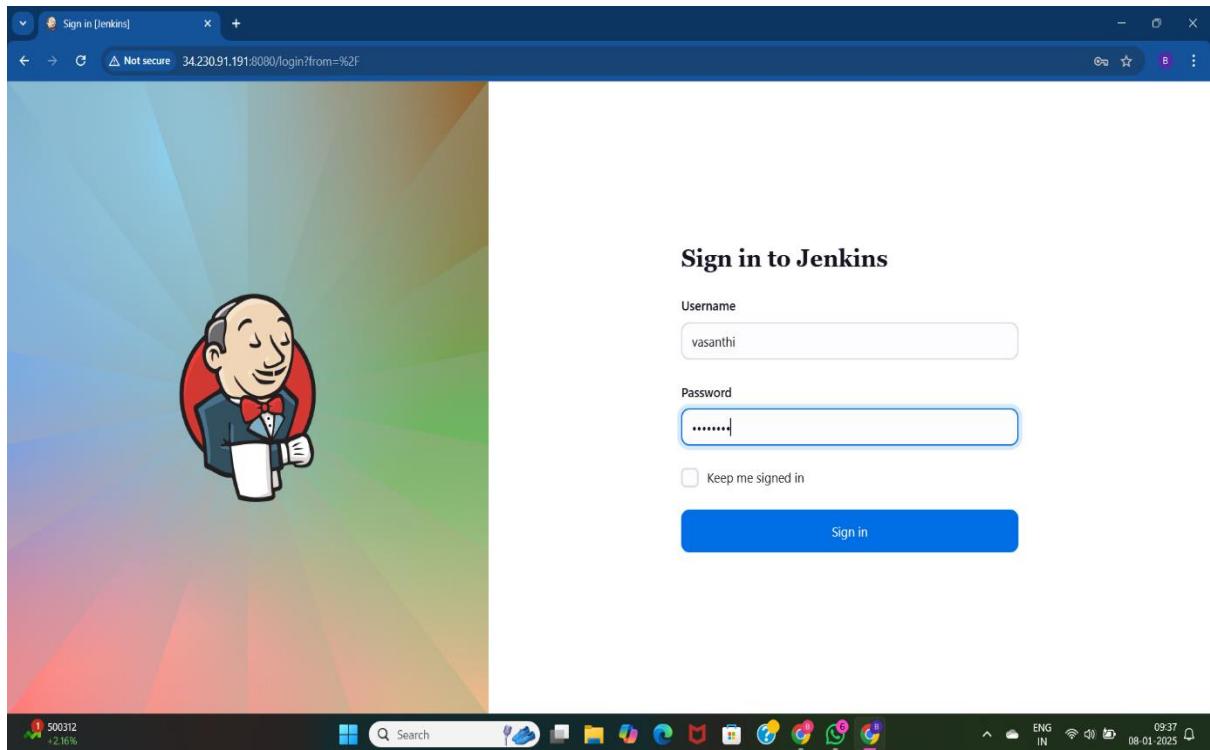
```

In the console output we can check the detailed information of the project.



Now access the deployed application using the public IP address of the slave server.

SLAVE3:



As the master is different person the Jenkins url is shared with the team. Here we are accessing the Jenkins using the username and password.

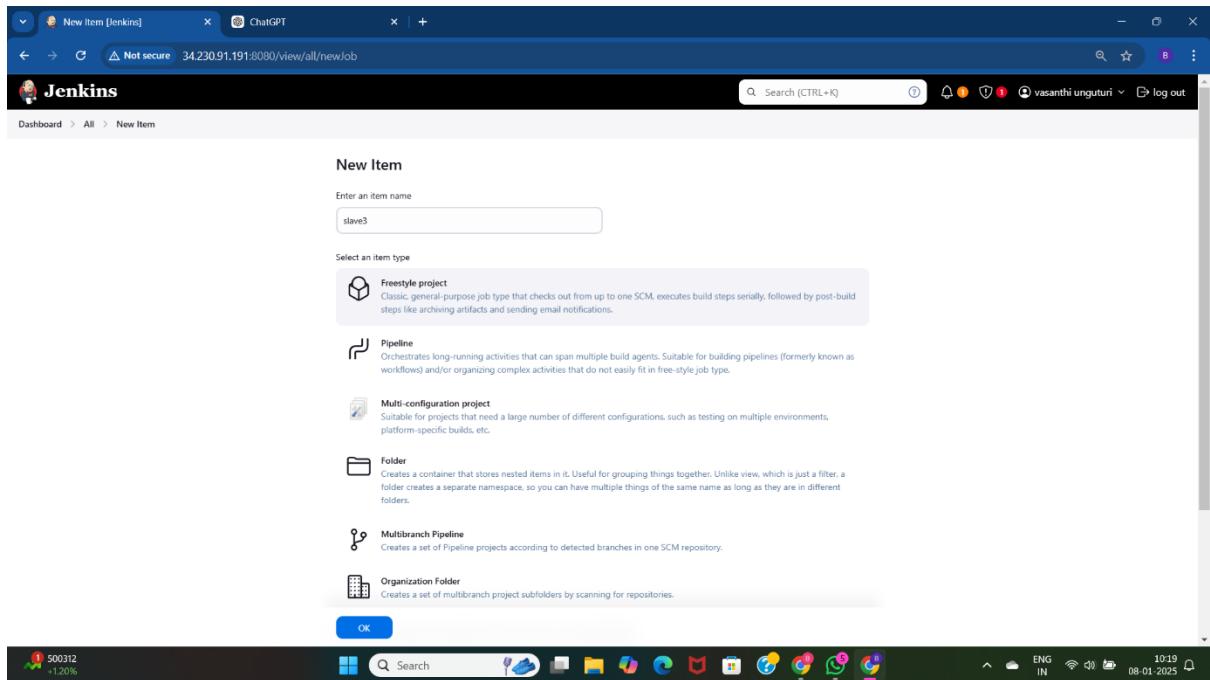
```
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-29-229 ~]$ cd .ssh/
[ec2-user@ip-172-31-29-229 .ssh]$ sudo vi authorized_keys
[ec2-user@ip-172-31-29-229 .ssh]$ 
[ec2-user@ip-172-31-29-229 ~]$ sudo yum install java-17-amazon-corretto -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
Resolving Dependencies
--> Running transaction check
--> Package java-17-amazon-corretto.x86_64 1:17.0.13+11-1.amzn2.1 will be installed
--> Processing Dependency: java-17-amazon-corretto-headless(x86-64) = 1:17.0.13+11-1.amzn2.1 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXMI1 for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXi for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXinerama for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXt for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrender for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXrandr for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: libXtst for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Processing Dependency: giflib for package: 1:java-17-amazon-corretto-17.0.13+11-1.amzn2.1.x86_64
--> Running transaction check
--> Package giflib.x86_64 0:4.1.6-9.amzn2.0.2 will be installed

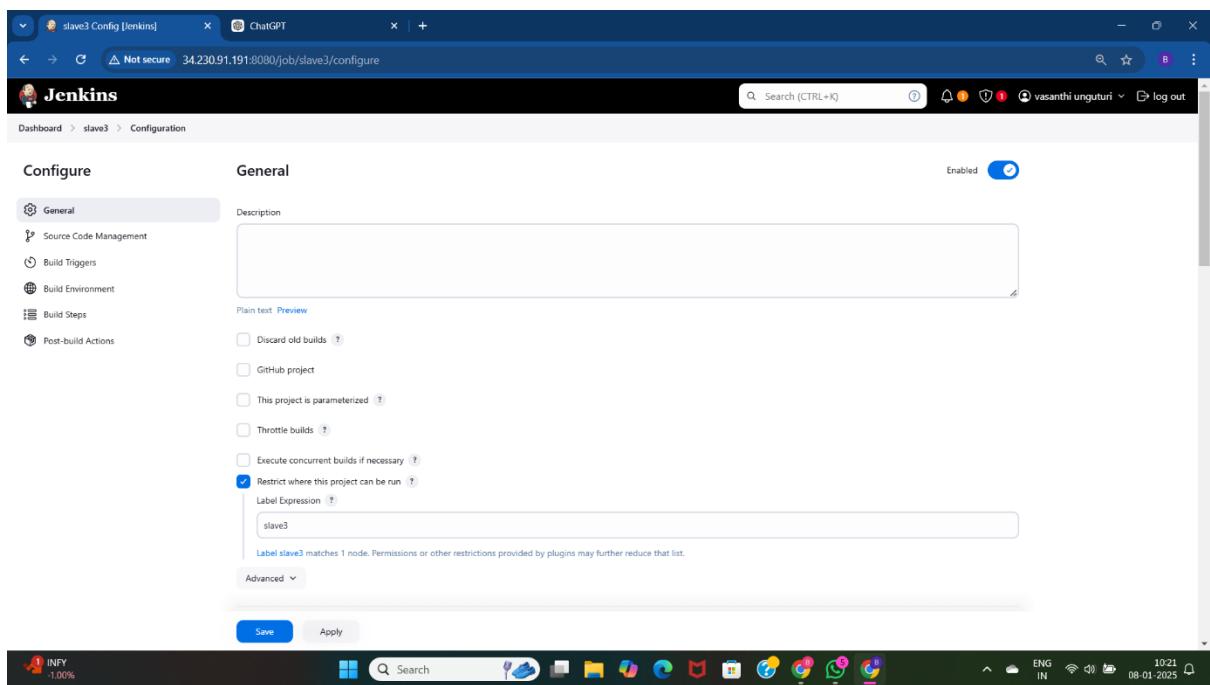
i-012b50baa234c3639 (slave3)
PublicIPs: 54.235.41.173 PrivateIPs: 172.31.29.229
```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 24°C Haze ENG IN 11:57 08-01-2025

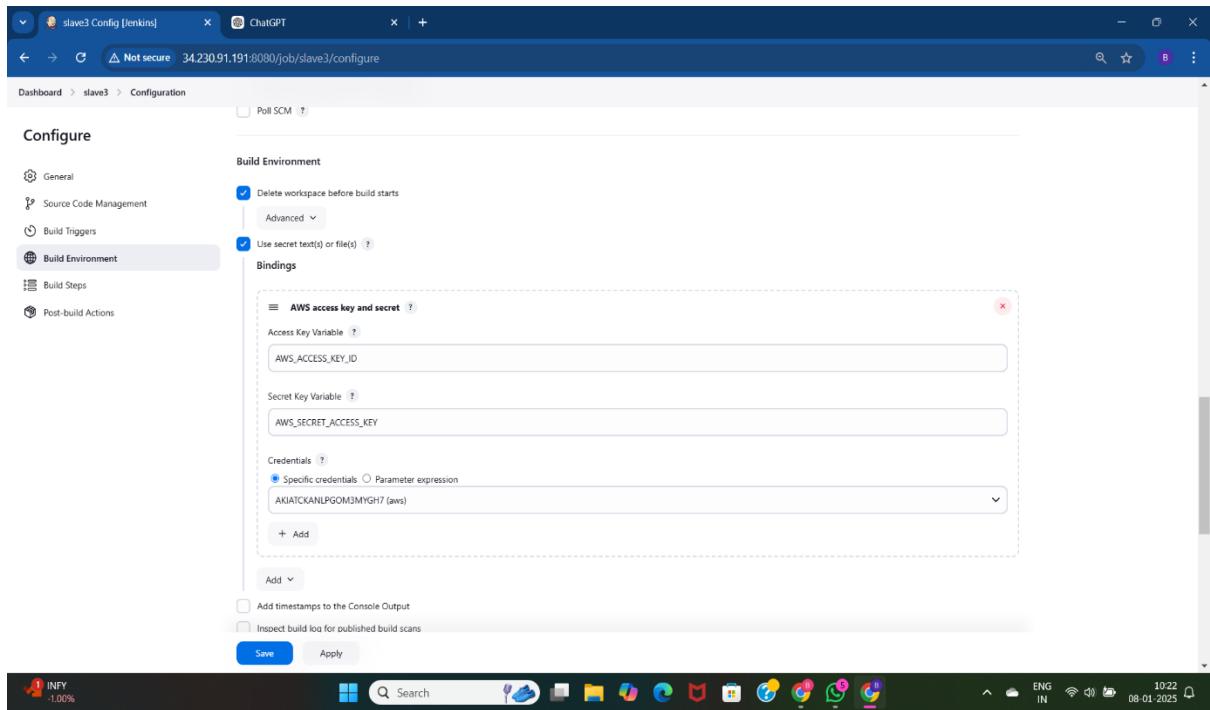
In slave3 server I have installed the java17 and connected it with the master using public key of master.



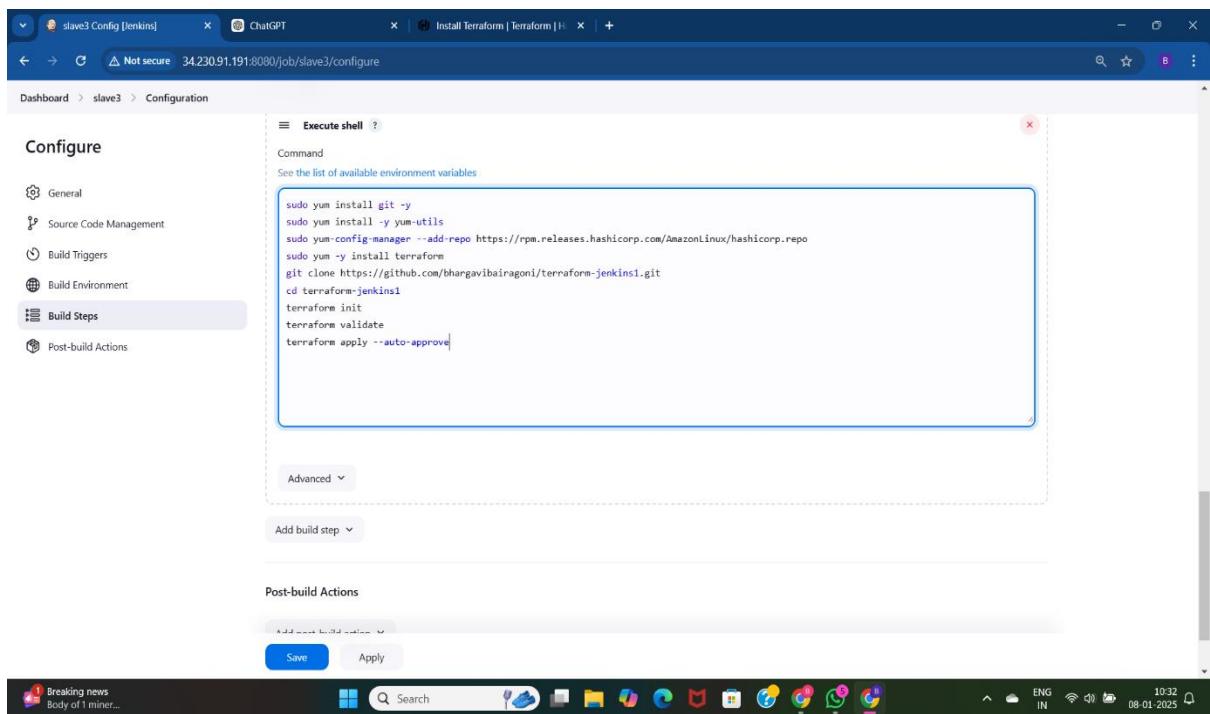
I have created a slave3 free style job for deploying the application using the terraform.



Select the created label here to run the builds.



Now give the aws access keys and secret access keys using the build environment.

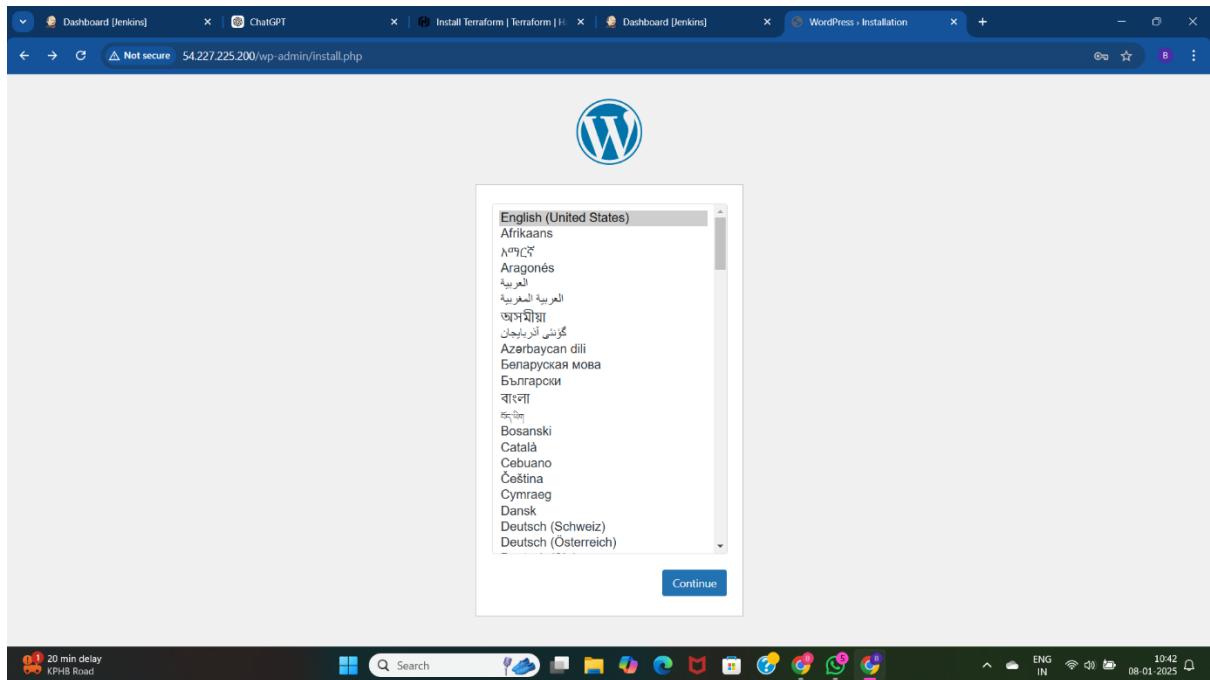


In execute shell write the commands to install git, terraform. Provide the github repository url which is having the terraform files to create the aws resources. Select the branch where we are having the code/files. Give the terraform commands to see and create the resources.



These are the files cloned using the execute shell. As the build is success the files are moved to this slave3 job.

A new ec2 instance is created when I have build the project. Now use this instance and access the application.



I have taken the newly created instance public IP address and accessed the wordpress application.

SLAVE-4:

```
Last login: Wed Jan  8 05:27:49 2025 from ec2-18-206-107-27.compute-1.amazonaws.com
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-25-231 ~]$ history
1 cd .ssh/
2 sudo vi authorized_keys
3 cd
4 sudo yum install java-17-amazon-corretto -y
5 history
[ec2-user@ip-172-31-25-231 ~]$ sudo yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn-core
Package git-2.40.1-1.amzn2.0.3.x86_64 already installed and latest version
Nothing to do
[ec2-user@ip-172-31-25-231 ~]$
```

i-09a426798b2ac3c89 (slave4)
PublicIPs: 107.22.138.60 PrivateIPs: 172.31.25.231



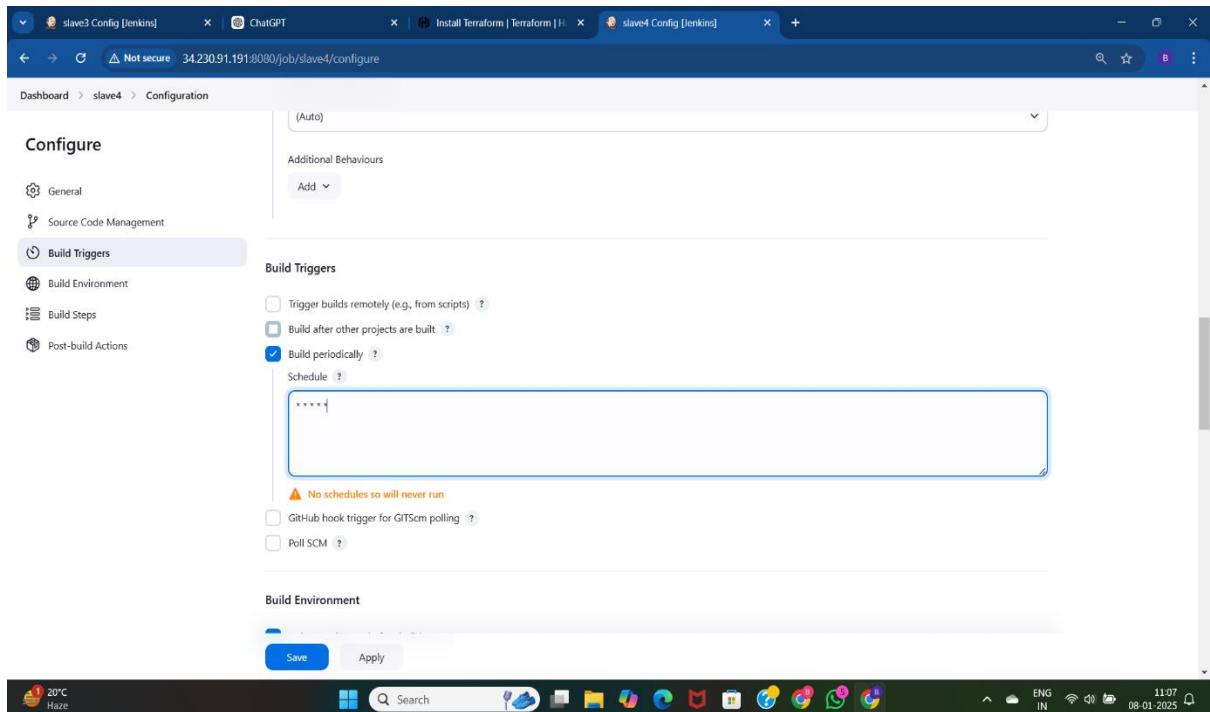
In slave 4 I have installed java17 and git. Connected it with the master.

The screenshot shows the Jenkins job configuration page for 'slave4'. In the 'General' section, under 'Label Expression', the value 'slave4' is entered. A note below states: 'Label slave4 matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.' At the bottom, there are 'Save' and 'Apply' buttons.

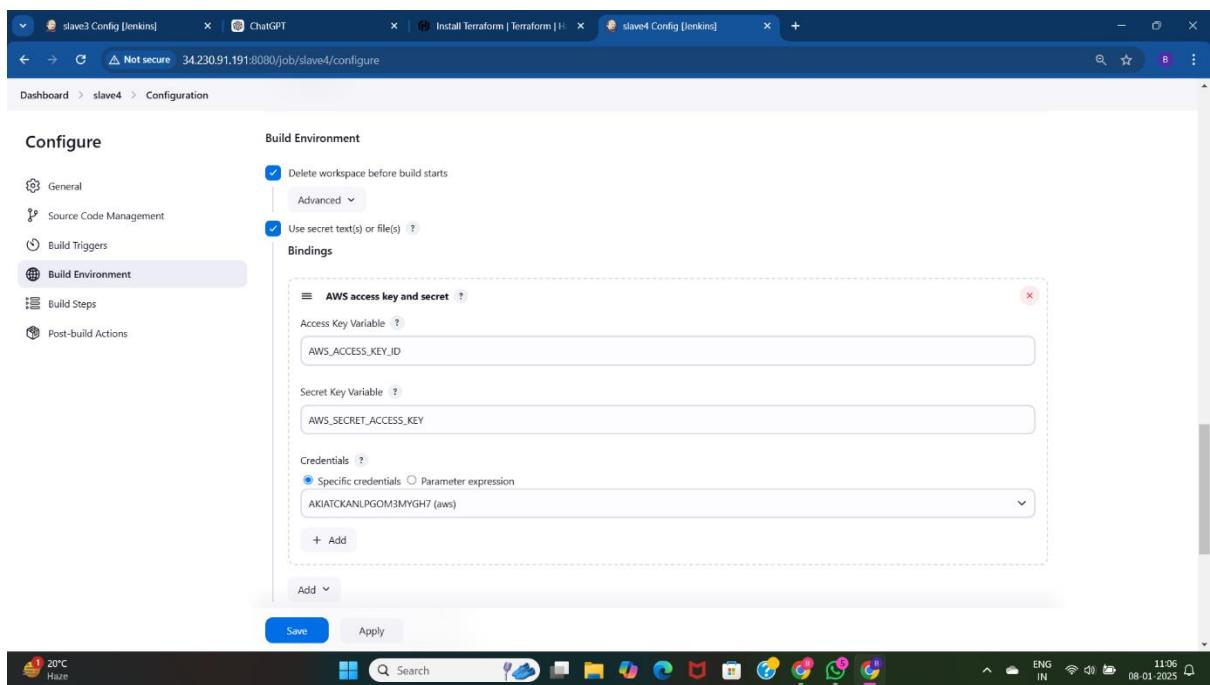
In Jenkins job select the created label for slave4.

The screenshot shows the Jenkins job configuration page for 'slave4'. In the 'Source Code Management' section, 'Git' is selected as the provider. Under 'Repositories', a single repository is configured with the URL 'https://github.com/bhargavibairagi/terraform-jenkins1.git'. The 'Branches to build' field contains the branch specifier '/main'. At the bottom, there are 'Save' and 'Apply' buttons.

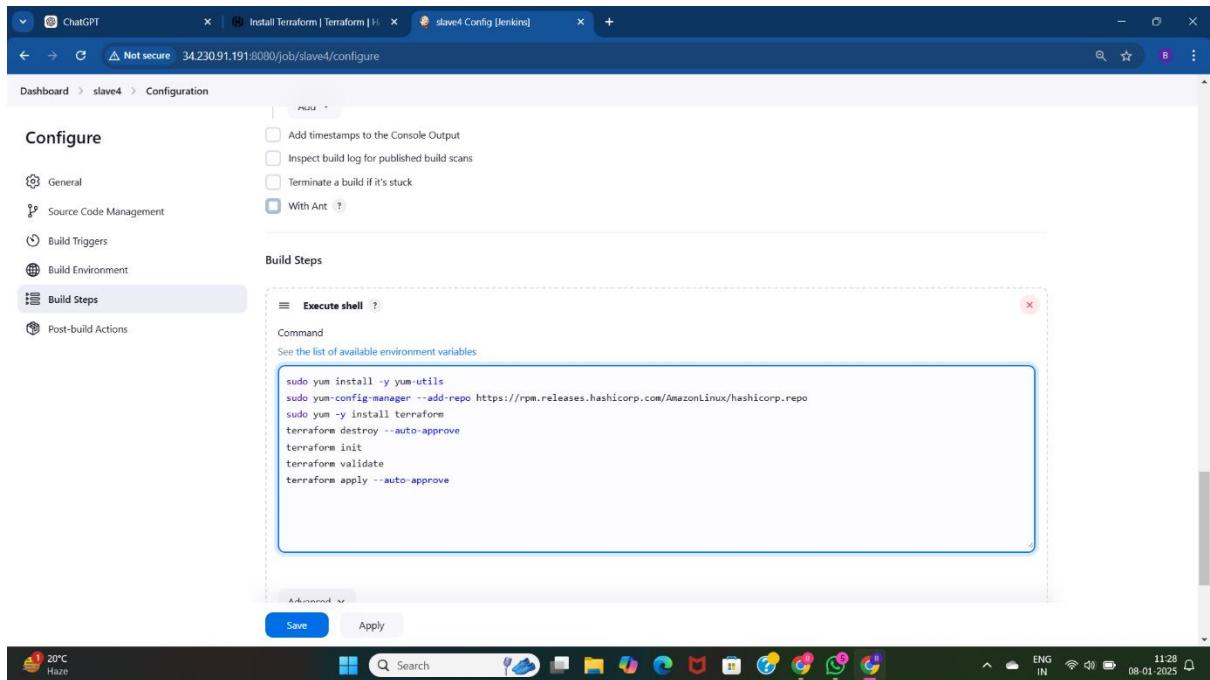
Select the github repository and the branch where we are having the terraform files.



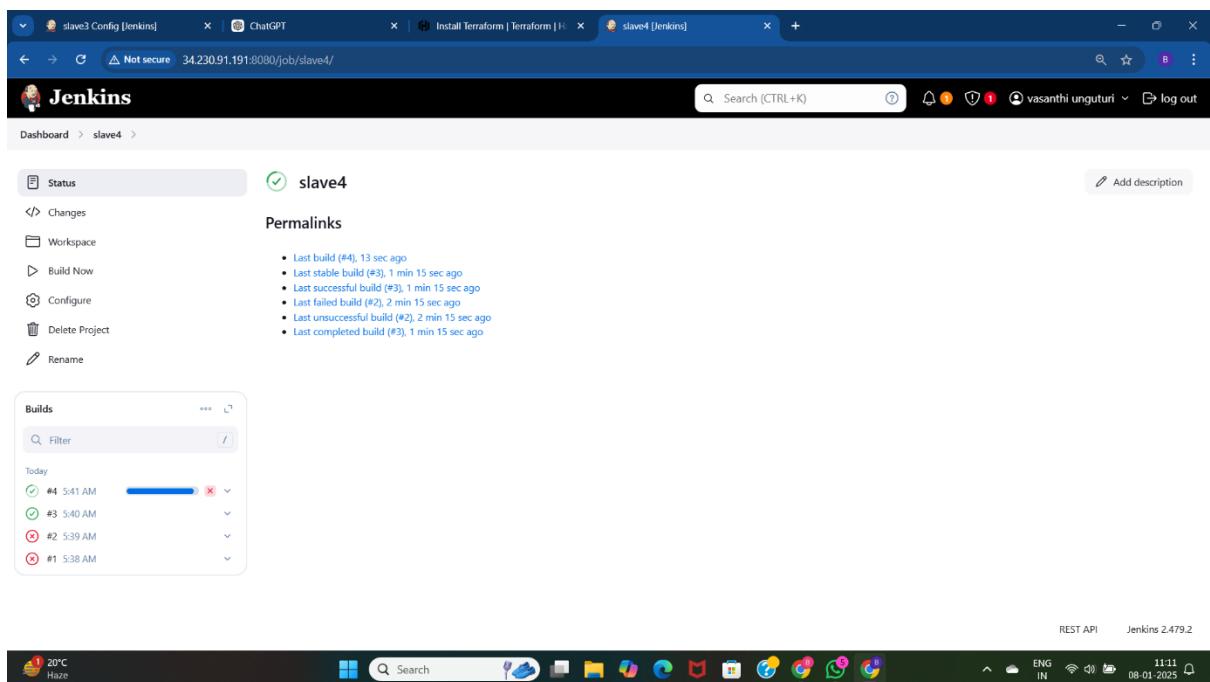
To deploy the application automatically select the build periodically option, give the syntax there. (minute, hour, day of the month, month, day of the week).



Select the created credentials using the build environment> use secret text or files.



In execute shell provide the commands to install terraform and check the terraform plan and create the resources using “terraform apply –auto-approve”.



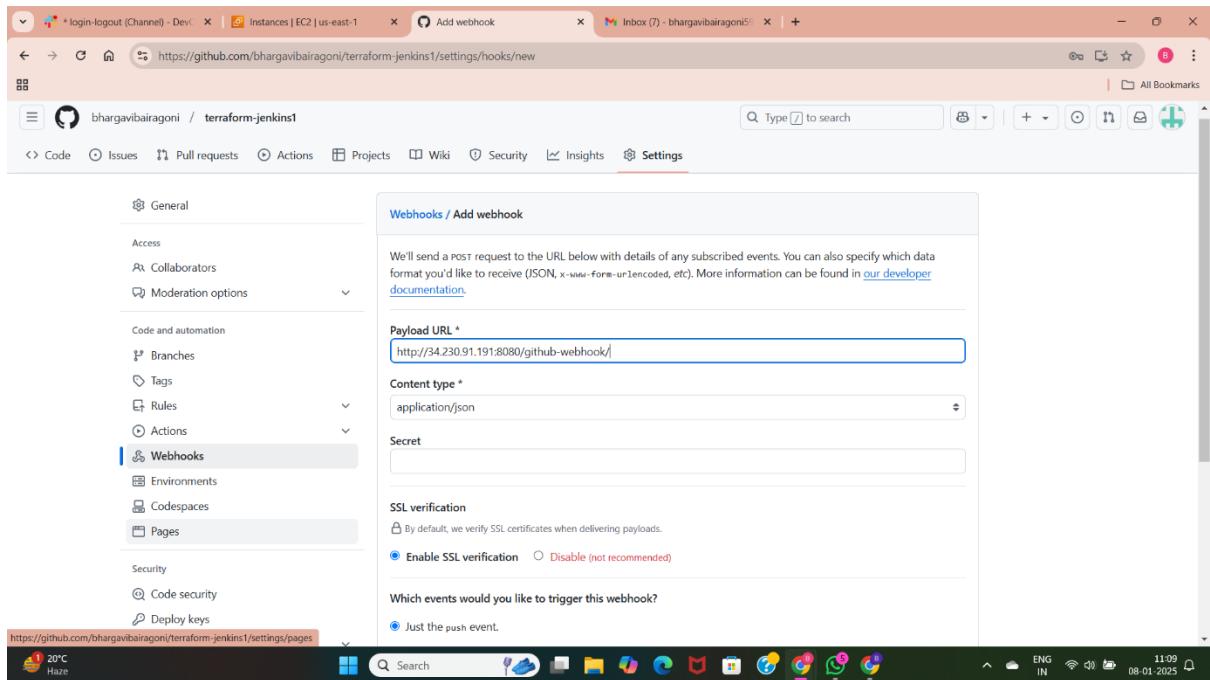
We can see the build is started automatically. We can also see the time gap between every build is one minute which matches which the syntax.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like 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 CloudShell. The main area displays a table of instances. One instance, 'My Terraform' (ID: i-0edd5850e294a36ba), is selected and highlighted with a blue border. The table columns include Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone. Below the table, a detailed view for 'My Terraform' shows its instance ID, public IPv4 address (52.90.5.226), private IPv4 addresses (172.31.93.91), and public IPv4 DNS (ec2-52-90-5-226.compute-1.amazonaws.com). The status summary indicates it's running.

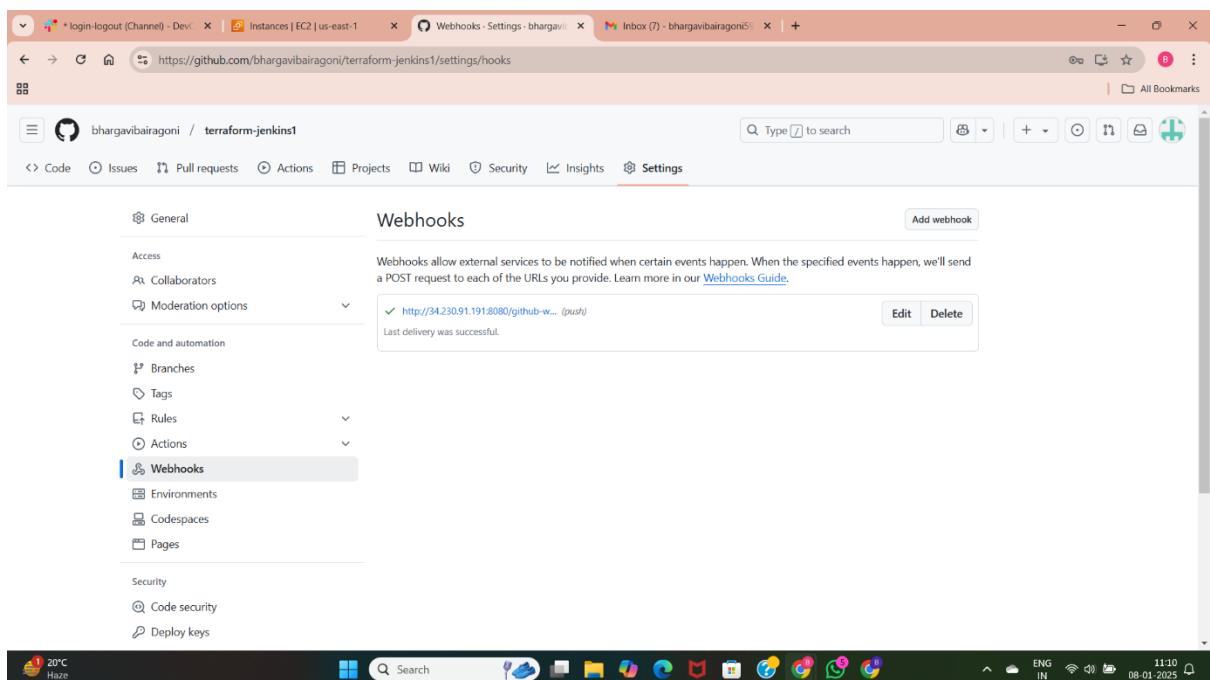
As the builds are running automatically new resources are creating in the aws console. Now use this public ip to check with the application.

The screenshot shows a web browser window with several tabs open: slave3 Config [Jenkins], ChatGPT, Install Terraform | Terraform, slave4 [Jenkins], and WordPress - Installation. The main content area shows the WordPress installation screen. A language selection dropdown is open, displaying a list of languages: English (United States), Afrikaans, অসমীয়া, Aragonés, العربية, العربية المعاصرة, অসমীয়া, گونئی اردو بولن, Azərbaycan dil, Беларуская мова, Български, বাংলা, ດັວໂນ, Bosanski, Català, Cebuano, Čeština, Cymraeg, Dansk, Deutsch (Schweiz), and Deutsch (Österreich). A 'Continue' button is visible at the bottom of the dropdown.

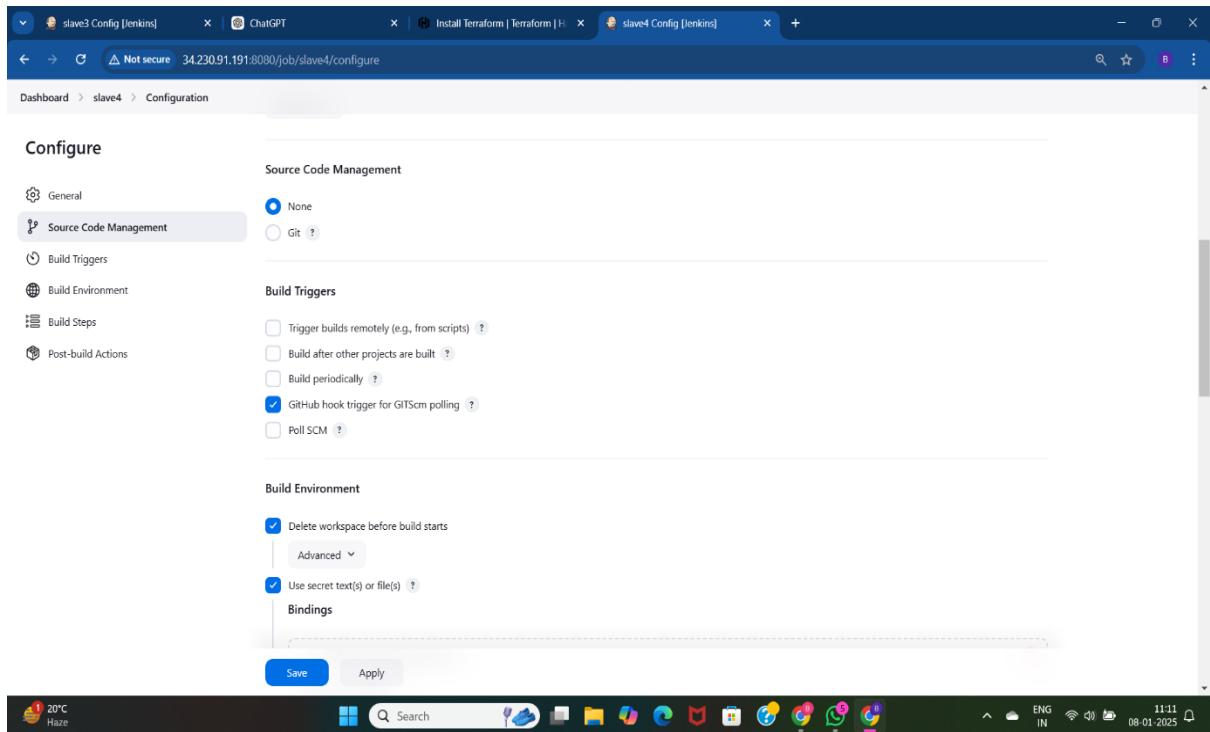
The application is deployed using the build periodically option.



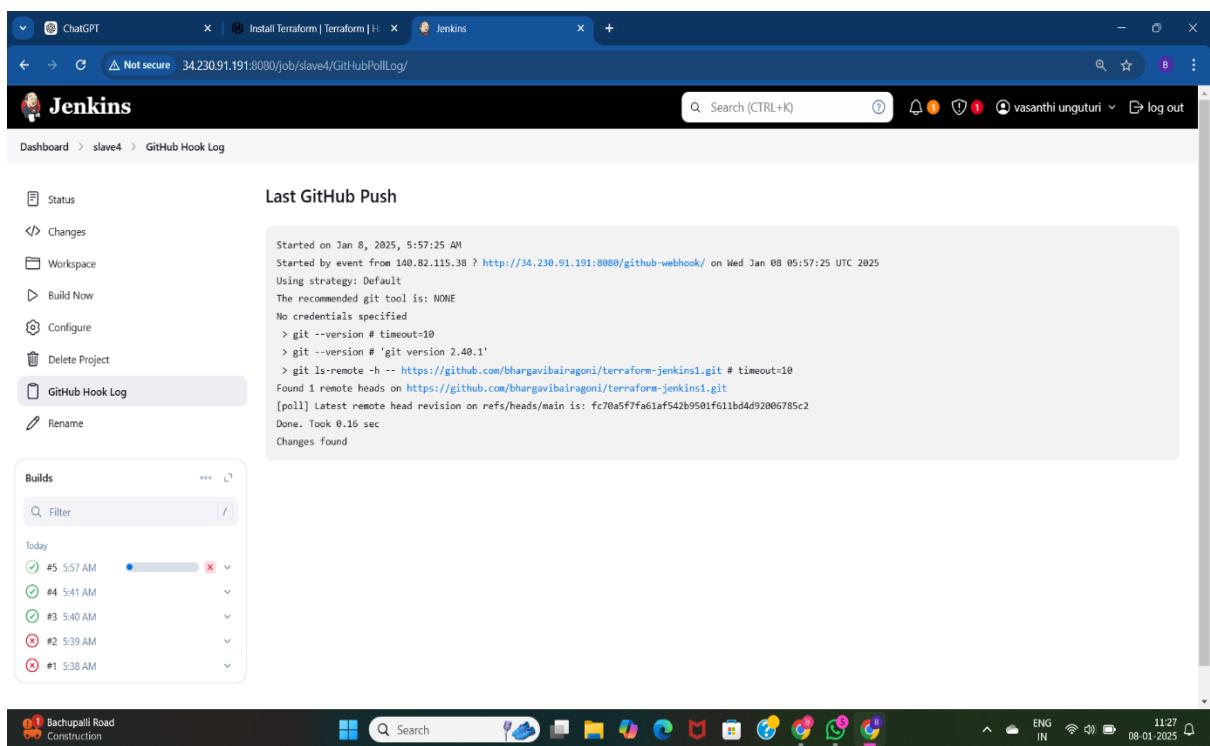
Now create the github webhooks in the repository where we are having the terraform files. Give the Jenkins url in the payload url. Select the content type as application/json.



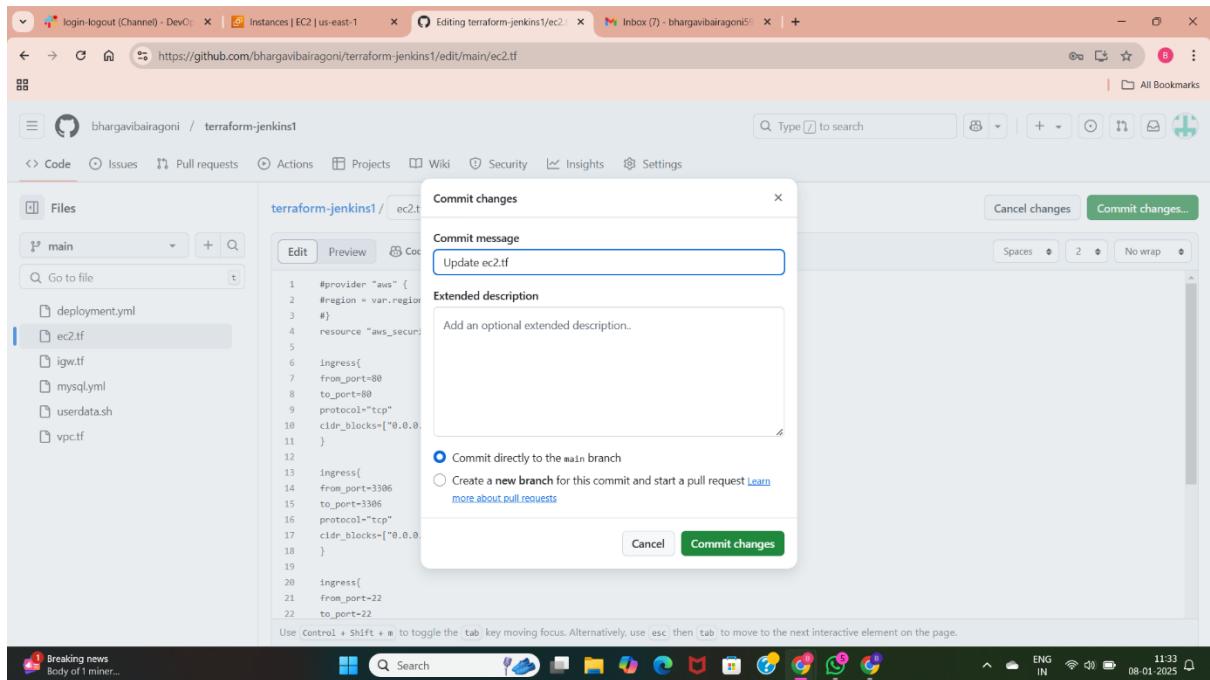
The webhook is created successfully. From now whenever we made changes we can automate the builds.



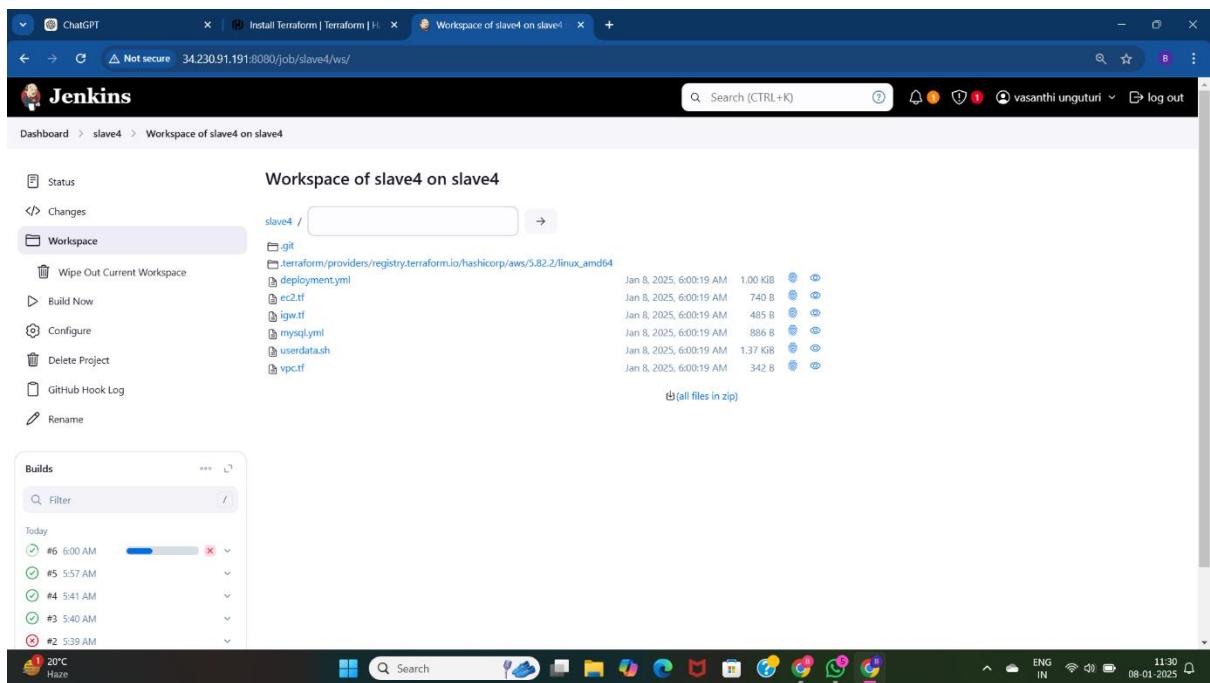
Now select the github hook trigger for gitscm polling and save the Jenkins job, Change code in the github repository to trigger the builds.



Here we can see the new option “github hook log” which will trigger from the github repository.



Here am changing the code in the terraform repository and committing the changes to trigger the builds.



Here we can see the builds are happening automatically when we do changes in the code.

The screenshot shows the AWS EC2 Instances page. The left sidebar navigation includes 'Instances' (selected), 'Images', 'Elastic Block Store', and 'Network & Security'. The main content area displays a table of instances with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone. One instance, 'Webhook' (Instance ID: i-08e12a38af73656c7), is selected. The details pane for this instance shows its summary, including its Public IPv4 address (44.203.178.52), Instance state (Running), and Private IP DNS name (ec2-44-203-178-32.compute-1.amazonaws.com).

Here a new instance is created with the terraform code.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled 'Install WordPress' and shows the first step of the WordPress installation wizard, which is selecting a language. A dropdown menu is open, listing various languages. The 'English (United States)' option is selected. Other options include Afrikaans, Aragonés, Arabic, العربية, অসমীয়া, গুৰাঙ୍ଗ অৰিয়া, Azərbaycan dil, Беларуская мова, Български, বাংলা, ດັວໂນ, Bosanski, Català, Cebuano, Čeština, Cymraeg, Dansk, Deutsch (Schweiz), and Deutsch (Österreich). At the bottom of the form, there is a 'Continue' button.

With the created new instance public Ip address a able to access the application.

The screenshot shows the Jenkins 'Configure' screen for a job named 'slave4'. Under the 'Build Triggers' section, the 'Poll SCM' option is selected, and the schedule is set to */2 * * * *. A warning message at the bottom of this section states: '⚠ Spread load evenly by using 'H/2 * * * *' rather than '/2 * * * *'. Would last have run at Wednesday, January 8, 2025 at 6:14:19 AM Coordinated Universal Time; would next run at Wednesday, January 8, 2025 at 6:16:19 AM Coordinated Universal Time.' Below the schedule, there is an unchecked checkbox for 'Ignore post-commit hooks'. The 'Build Environment' section contains a checked checkbox for 'Delete workspace before build starts'. At the bottom of the configuration page, there are 'Save' and 'Apply' buttons.

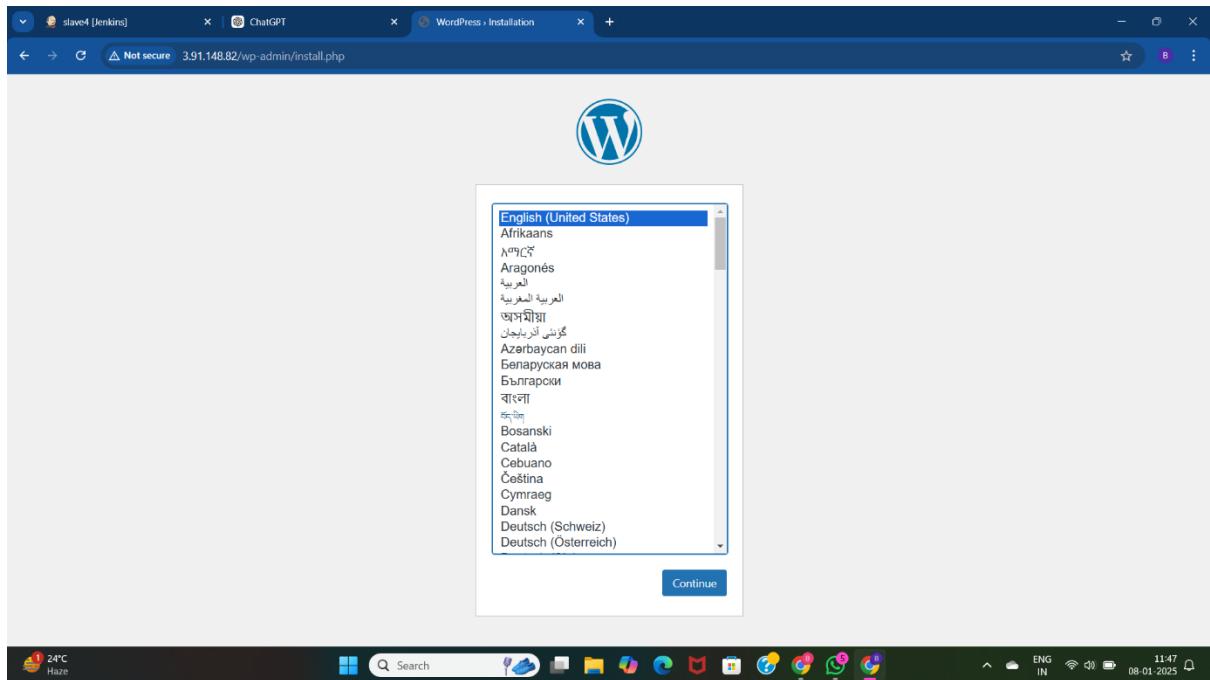
Now select the poll scm for checking the automatic builds in the Jenkins using webhook.

The screenshot shows a GitHub commit dialog for a file named 'ec2.tf' in a repository called 'terraform-jenkins1'. The commit message field contains the text 'Update ec2.tf'. Below the message, there is an 'Extended description' field with the placeholder 'Add an optional extended description..'. At the bottom of the dialog, there are two radio button options: 'Commit directly to the main branch' (selected) and 'Create a new branch for this commit and start a pull request'. There are also 'Cancel' and 'Commit changes' buttons. The background shows the GitHub interface with other files like 'main', 'deployment.yml', 'igw.tf', 'mysql.yml', 'userdata.sh', and 'vpc.tf' listed.

Change the code in the terraform files and commit the changes.

New builds are triggered using the poll scm for every two minutes.

New instances are created using poll scm and terraform.



SLAVE-5:

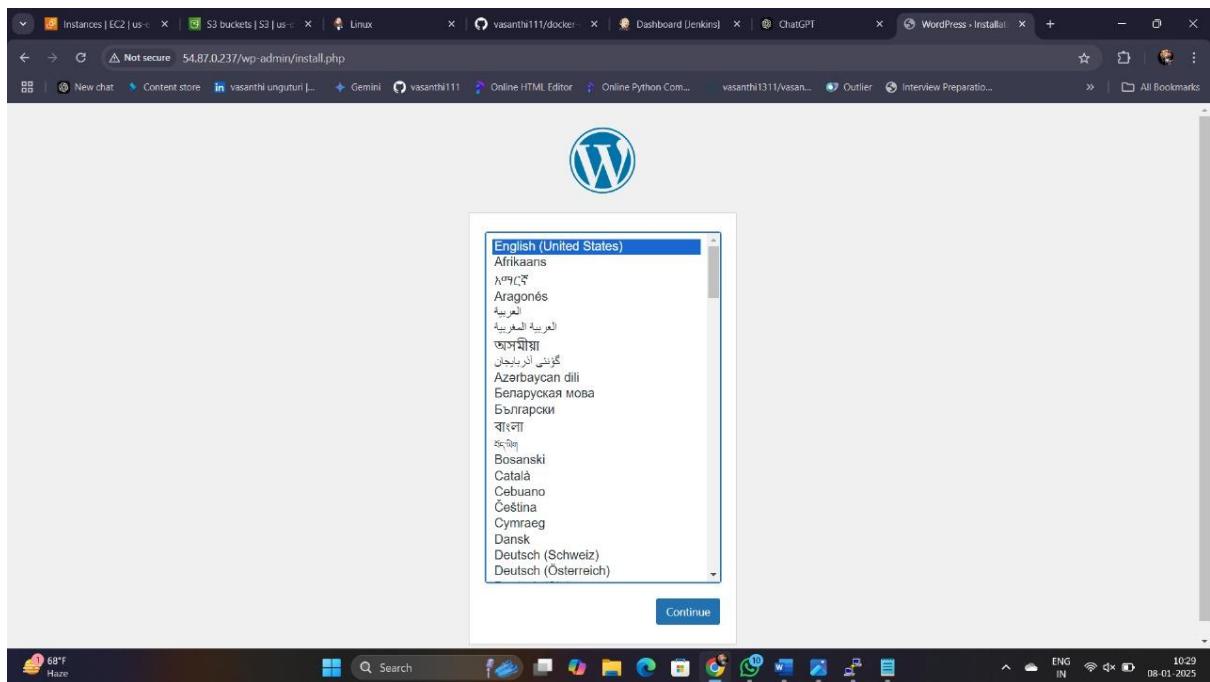
Now deploying the application using the bash script with docker, docker-compose. For that install and start docker. Install docker compose and give permissions.

The screenshot shows a browser window with multiple tabs open. The active tab is 'slave5' on the Jenkins dashboard. The build history for slave5 shows several builds, with build #7 being the most recent and successful. The Jenkins interface includes a sidebar with options like Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. Below the sidebar is a 'Builds' section listing recent builds with their status and number.

Now save and click on the build now option. If the build is success we can able to access the application.

The screenshot shows the Jenkins console output for build #7. The output details the steps taken during the build process, including the installation of Docker and Docker Compose. The terminal session shows commands like 'curl' to download Docker Compose, 'tar' to extract it, and 'sudo mv' to move the binary to /usr/local/bin. It also shows the creation of a Docker socket and the start of the Docker service. The Jenkins UI at the top indicates the build was successful.

In console output we can see every step related to installations of docker, docker compose.



Now use the slave5 public IP address and access the application.