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.

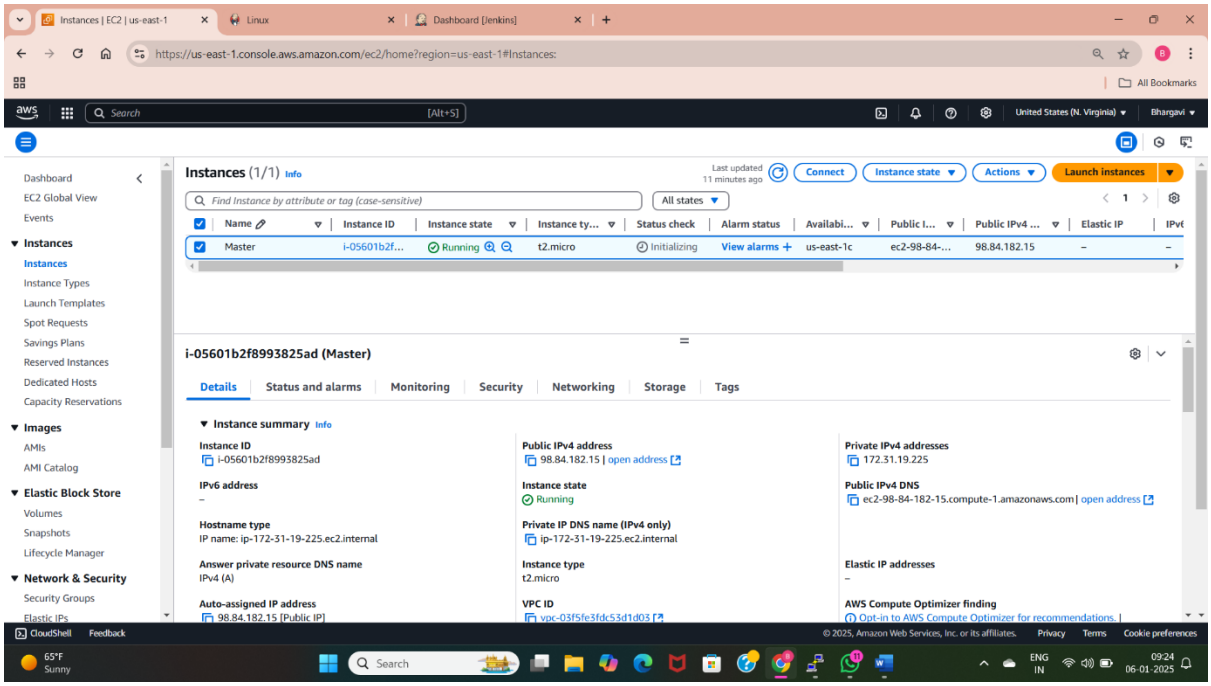
METHOD-3:

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

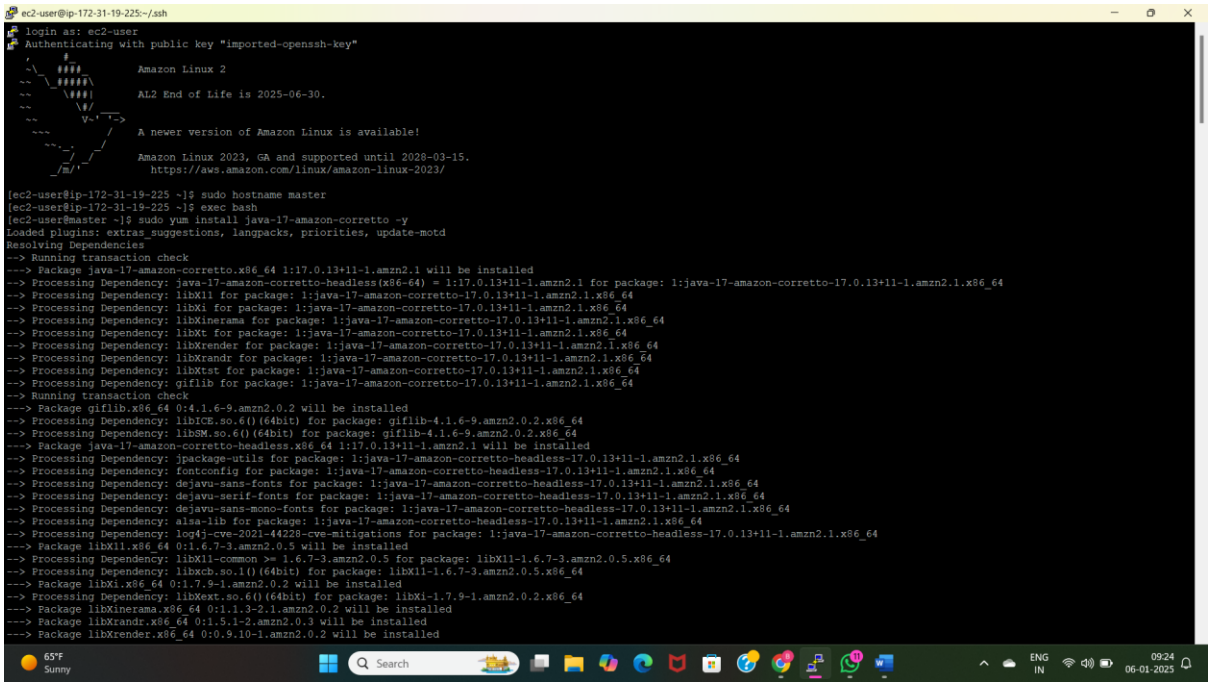
- Master, Slave-5: Bash Script (Bhargavi)
- Slave-1: Kubernetes (Vamsi)
- Slave-2: Docker Compose (Vasanthi)
- Slave-3: Terraform (Balaji)
- Slave-4: Jenkins (Poll SCM, Build Periodically, Webhooks by Balaji)

MASTER INFRASTRUCTURE SETUP:

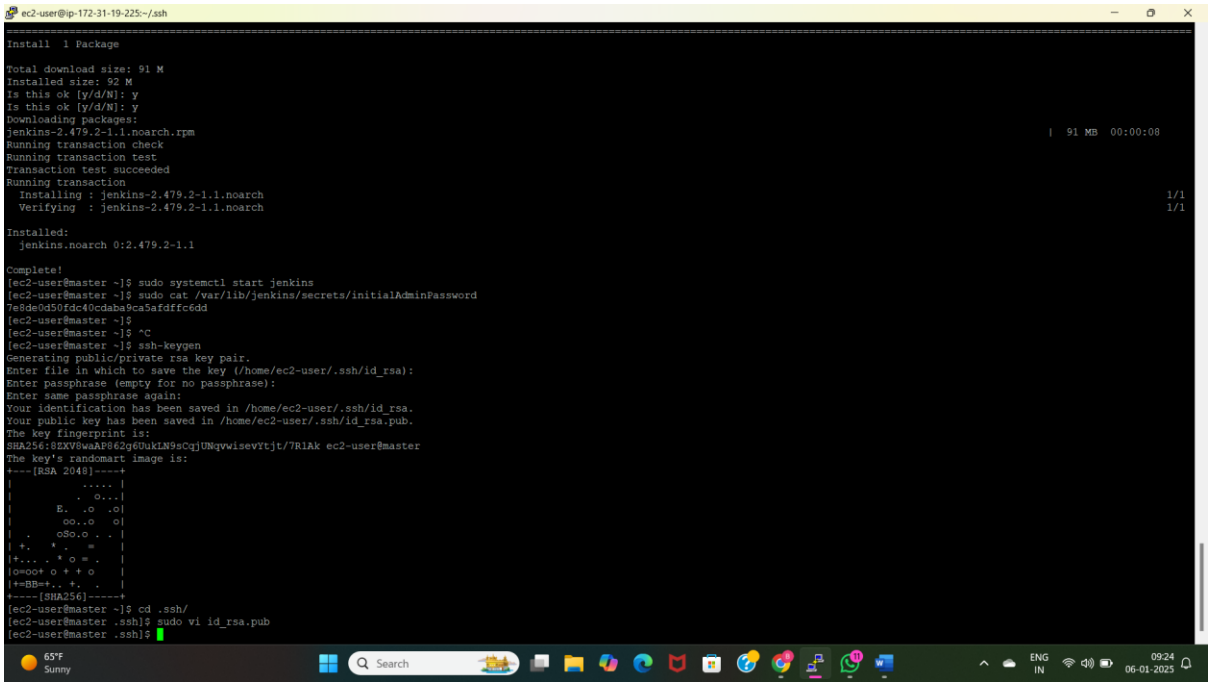
First we(first person) need to create a master, slave 5. For that first launch a master server with amazon-linux machine image with t2.micro, and allow required security group ports in inbound.



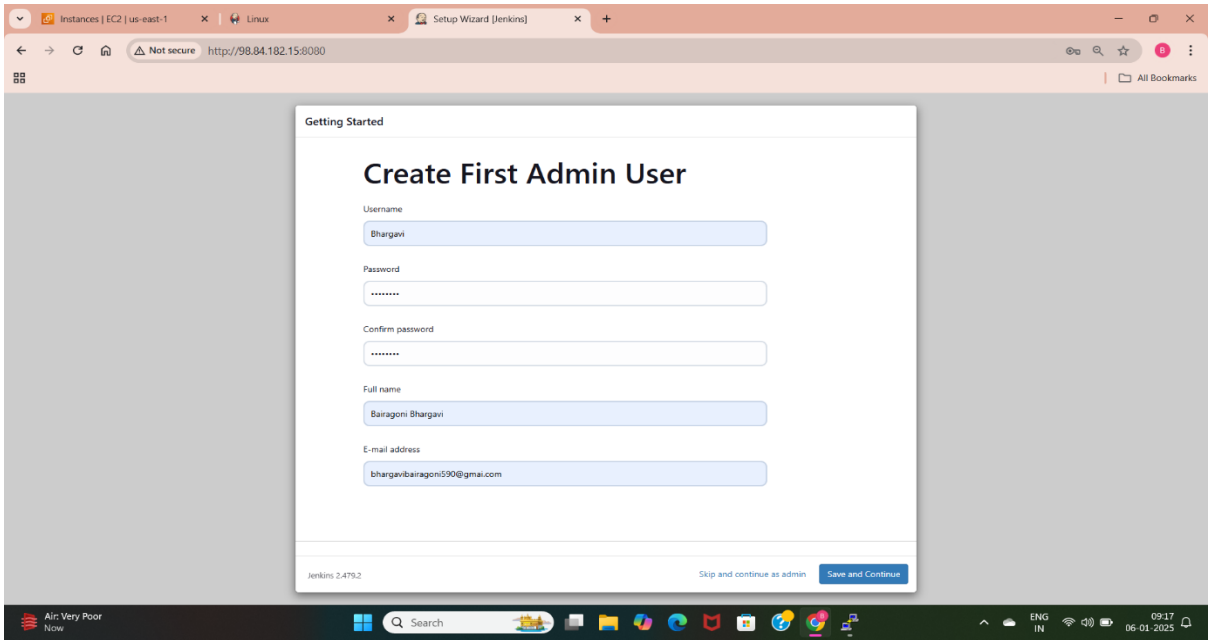
Master is created using above configurations. Now setup Jenkins in this master server. For that first we need to connect this master with terminal using the public Ip address of instance and private key(i.e key pair).



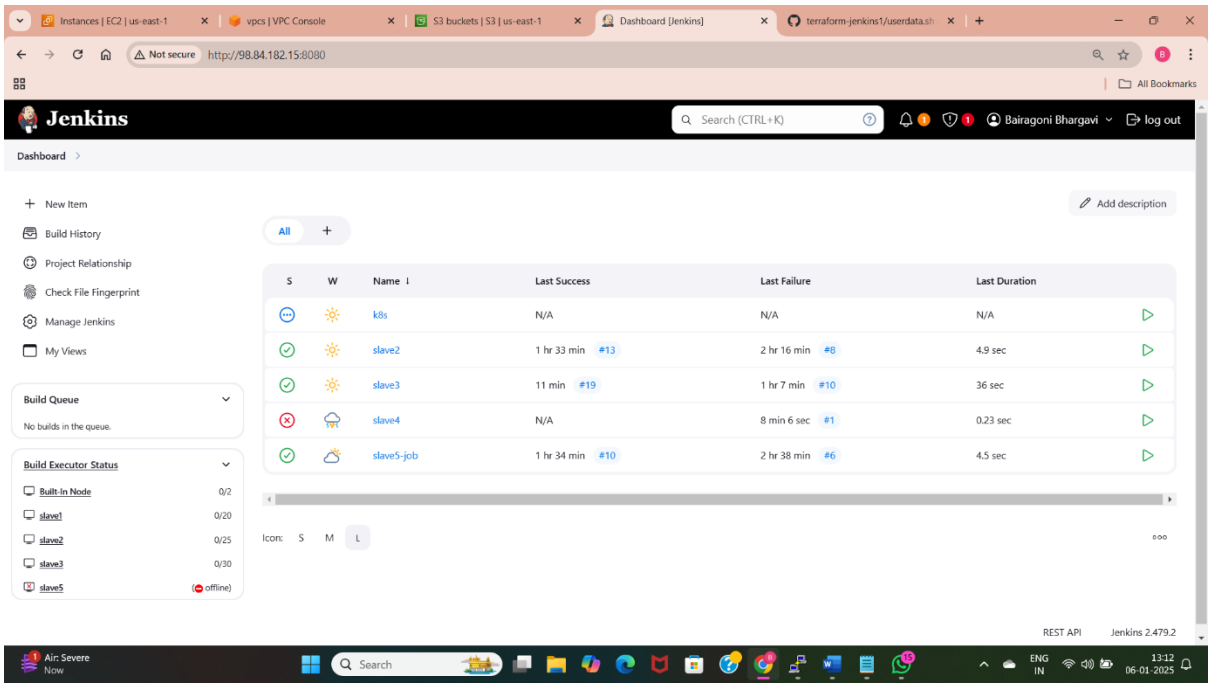
- Here I have changed the hostname to master using “sudo hostname master” and apply the changes using “exec bash”.
- Install java-17 in master using “sudo yum install java-17-amazon-corretto -y”.
- Install Jenkins repository and install token to login to the Jenkins.



- 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.

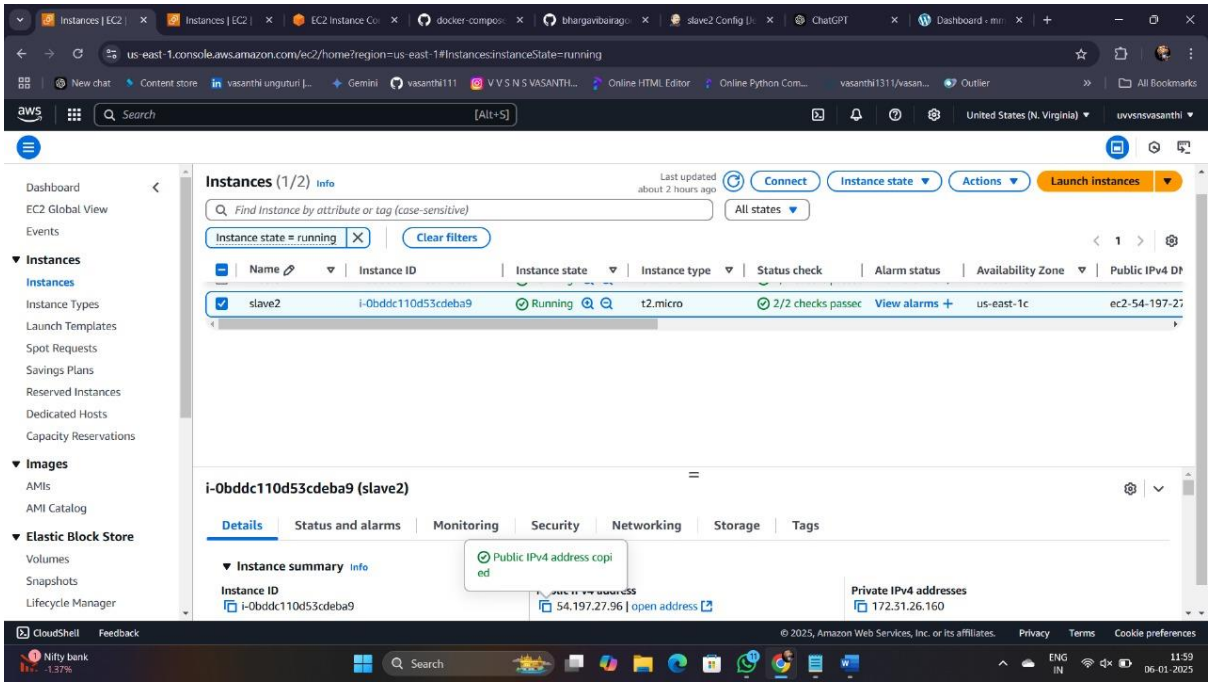


- Now create Jenkins user with password and provide full name and email address, save and create.
- Next we will see the Jenkins dashboard which shows new items, manage Jenkins and other options we can create jobs and nodes and install plugins.
- Next I (bhargavi) have shared the Jenkins url with my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

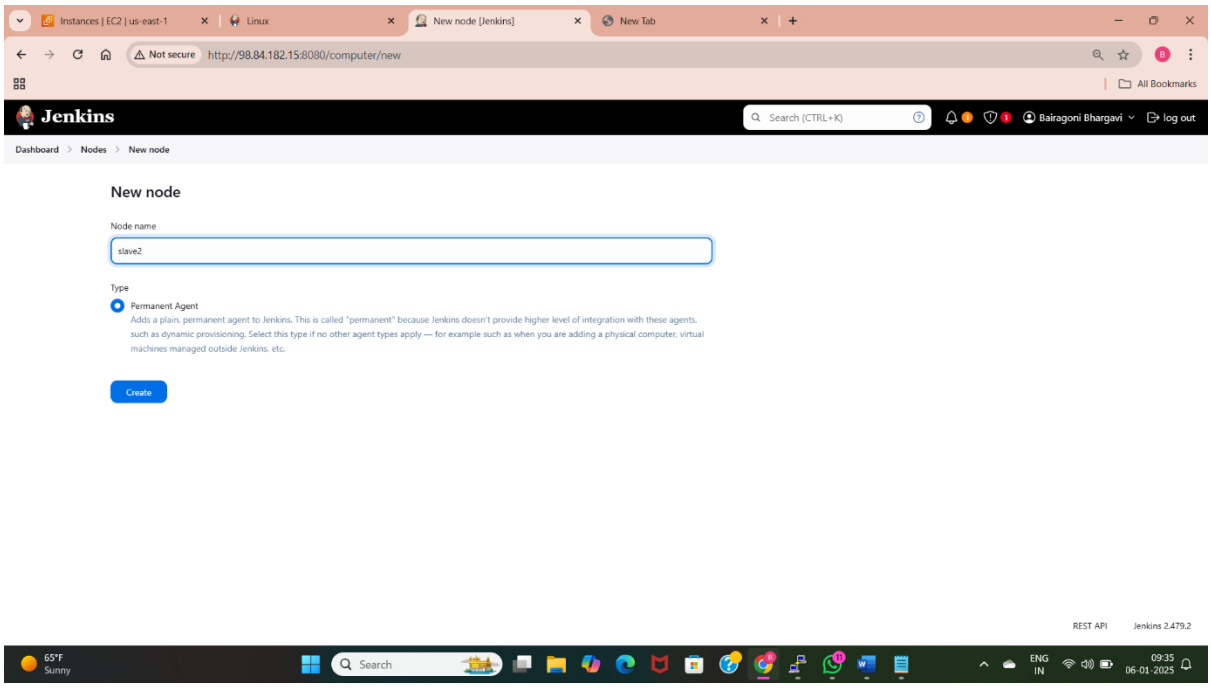


These are the jobs created in the Jenkins master and we are going to deploy the applications using these jobs.

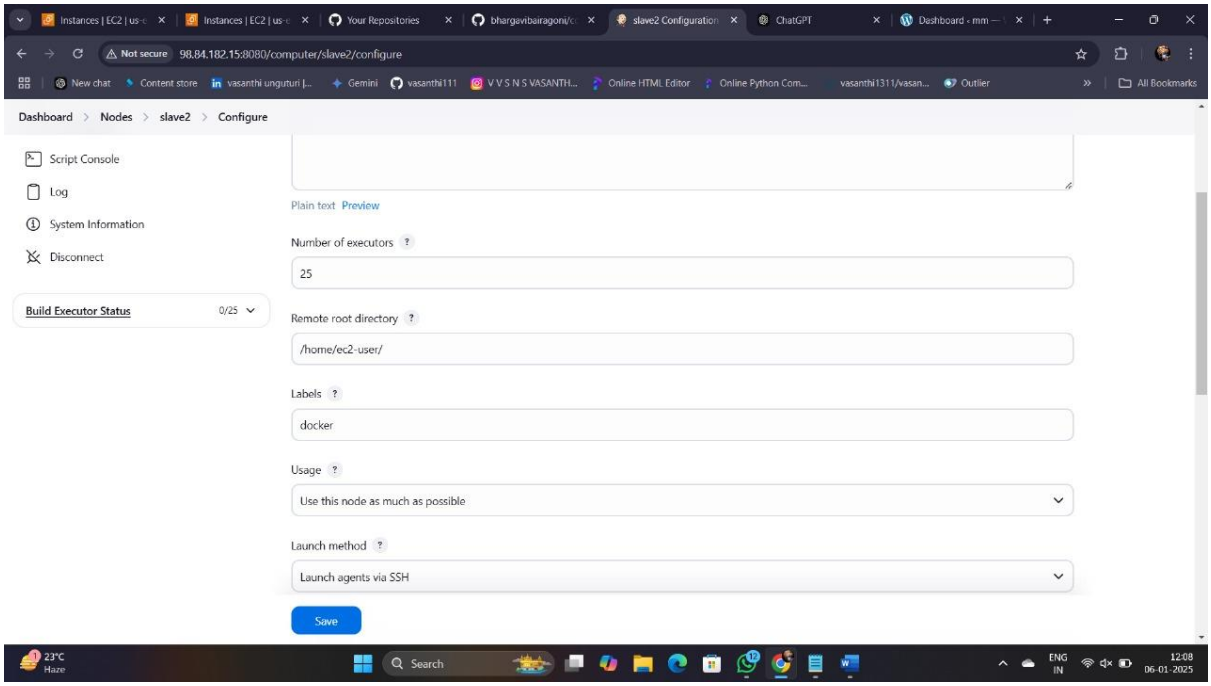
SLAVE-2 :



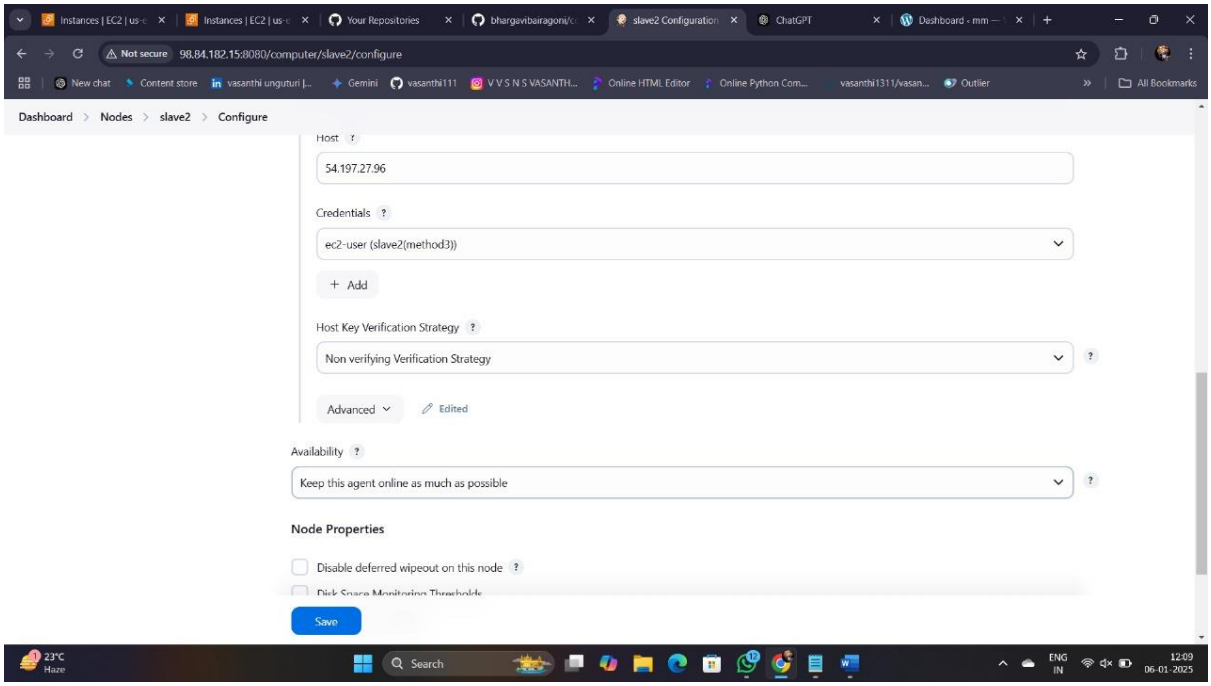
- Here vasanthi is creating a slave2 server in her aws account.
- Now she is able to access the Jenkins using the username and password of above created Jenkins credentials.



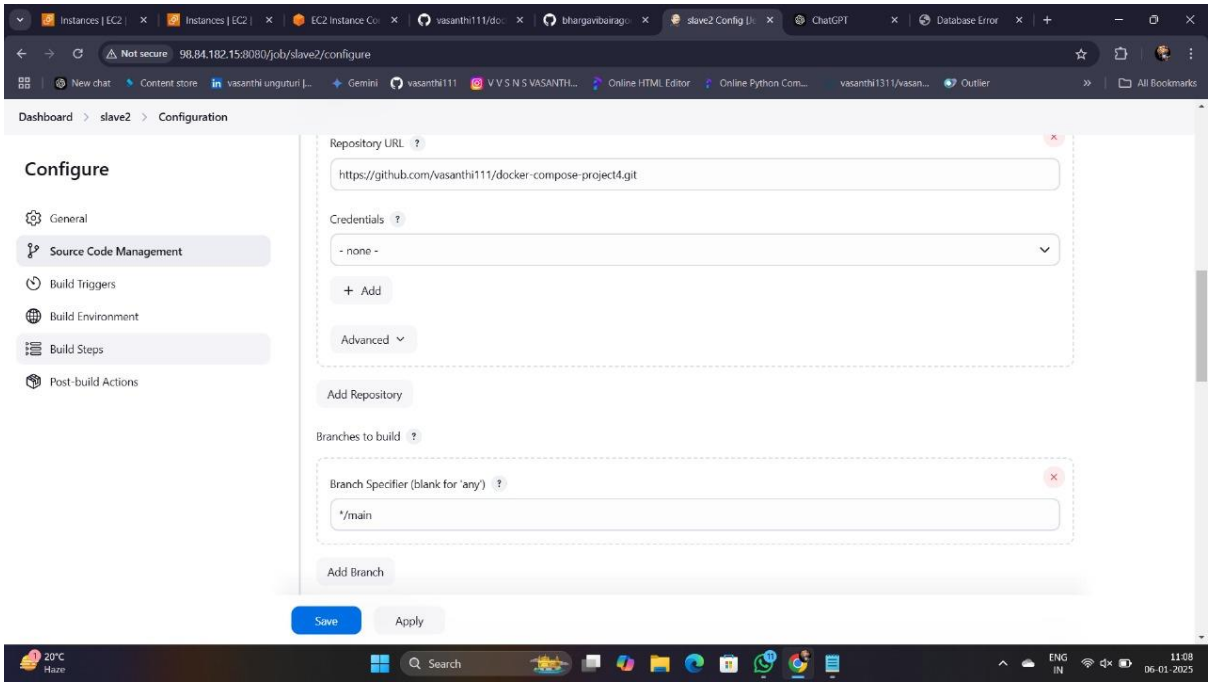
- Now she creating slave2 in the Jenkins. For that go to manage Jenkins select nodes and give name for node.



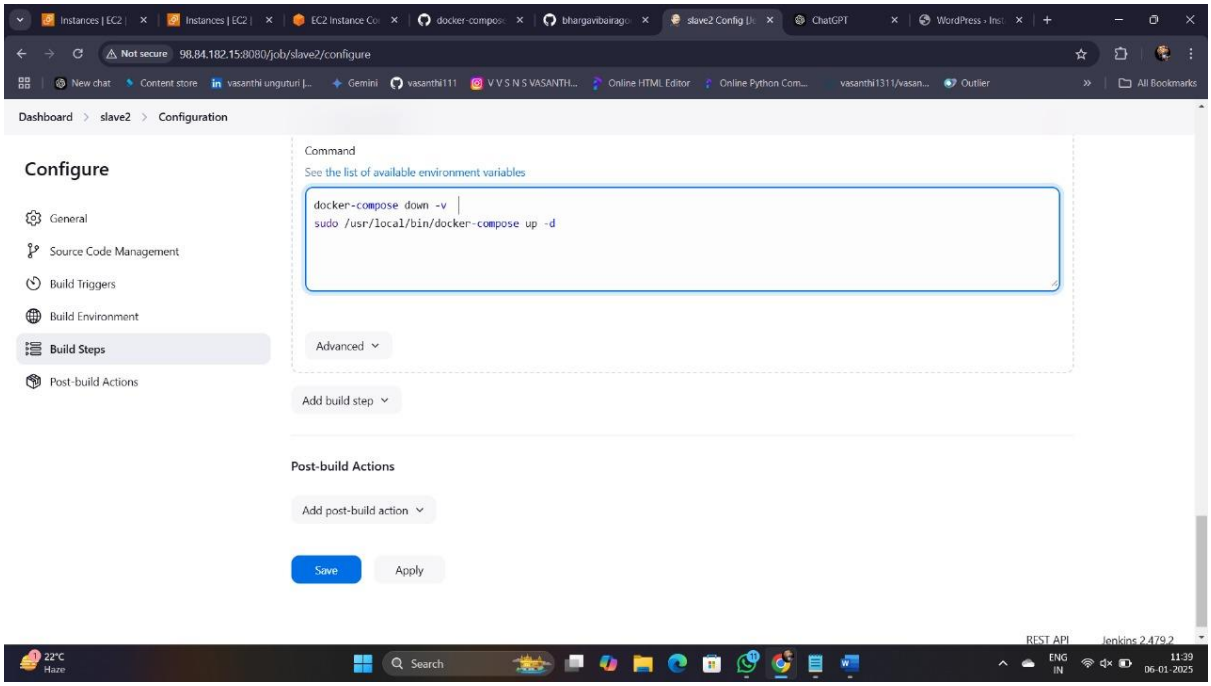
- Now select number of executors for slave 2 node, give remote directory path of slave 2 node and name for lable, select usage and launch method as launch via ssh.



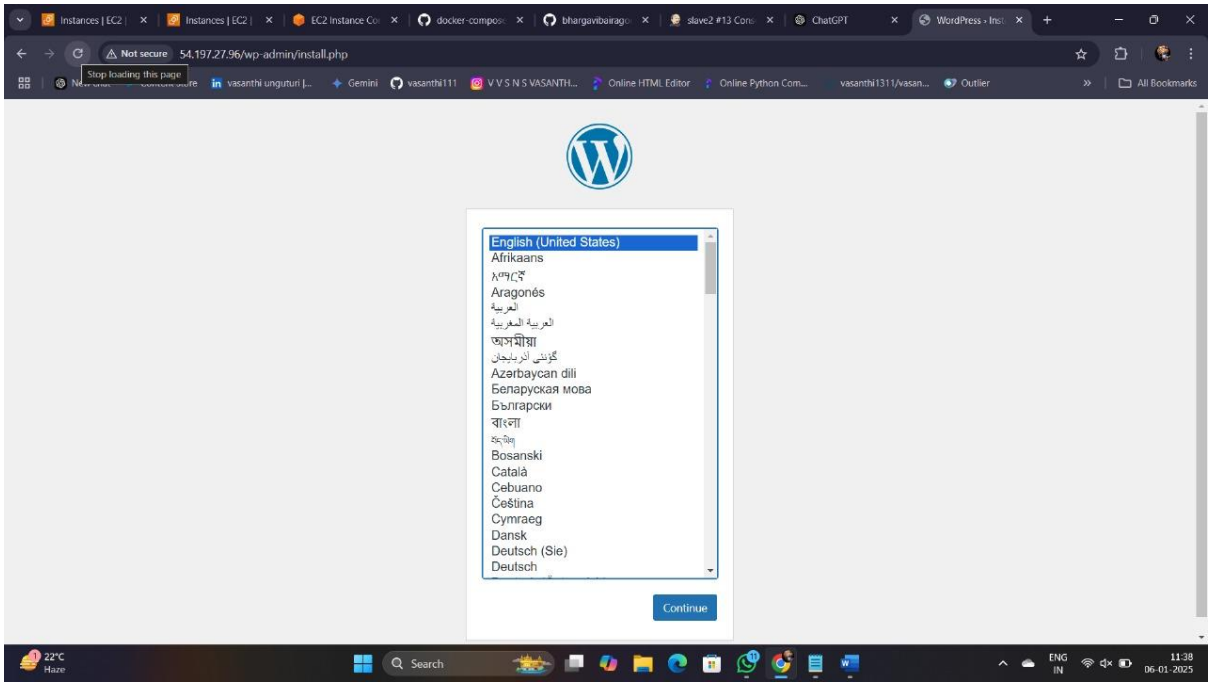
- Here selecting the host as slave2 server public ip because we are using two different accounts so if we take private ip it wont get internet from masters VPC and select verification as non verifying strategy because we are using different key pairs for master and slaves.
- Next create a job in Jenkins.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

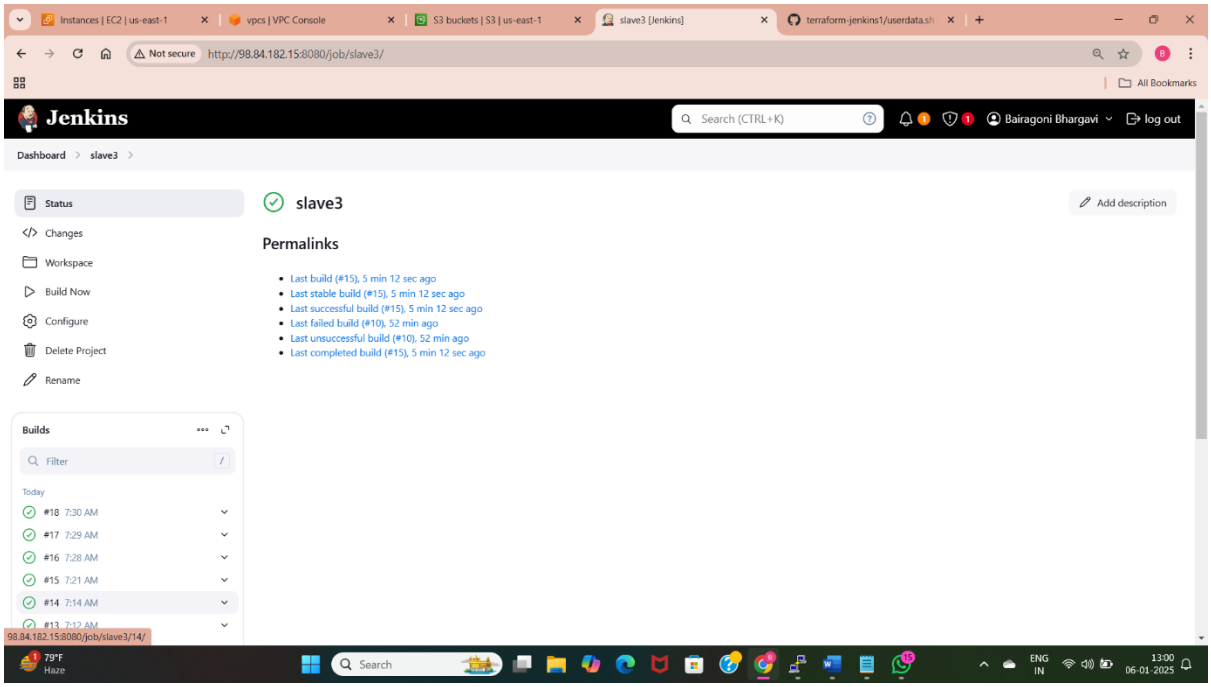


- As the build is success we can able to access the wordpress application which is deployed using slave2 server by using its public IP address.

SLAVE-3:

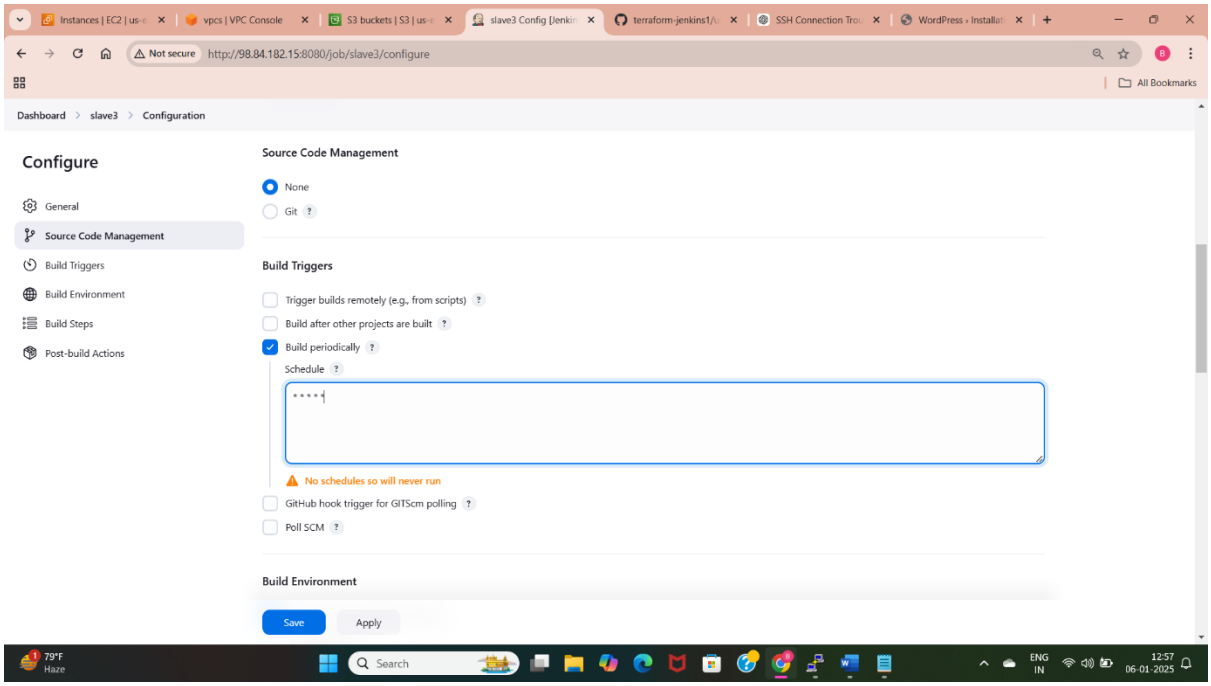
Now connect the slave3 server with the master using the master public key and paste that key in the slave3 authorized keys.

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

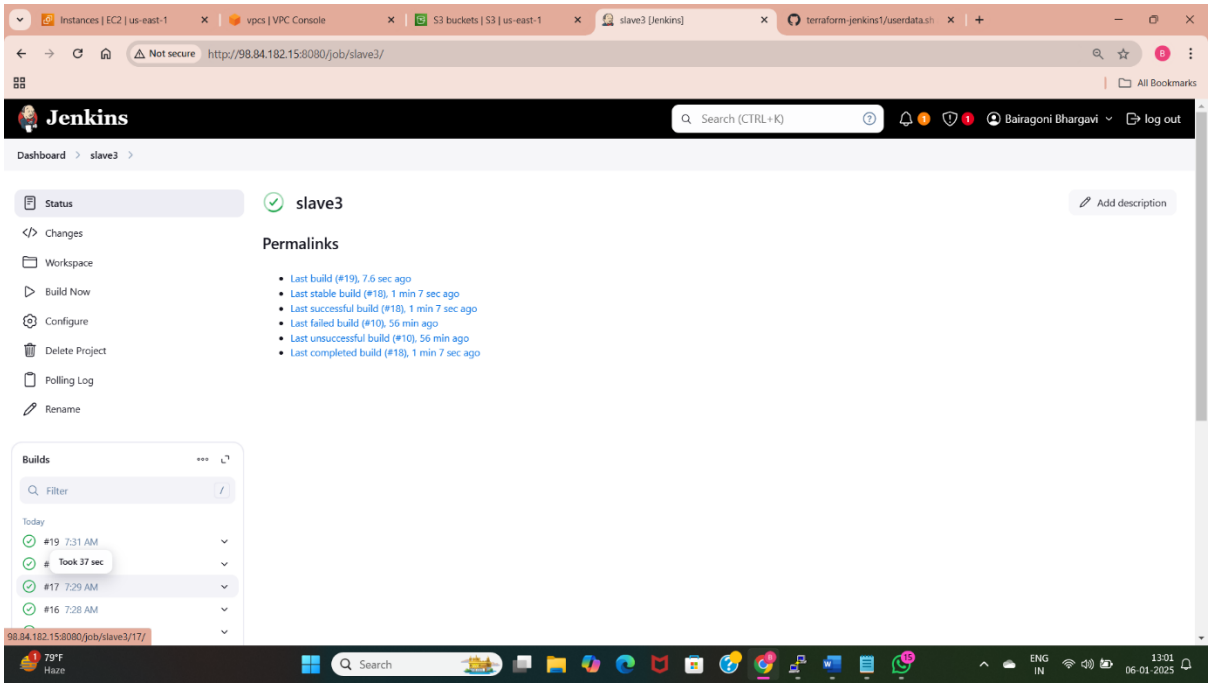


The build is started by the slave3 server and it is success. Now we can able to access the application using this slave server.

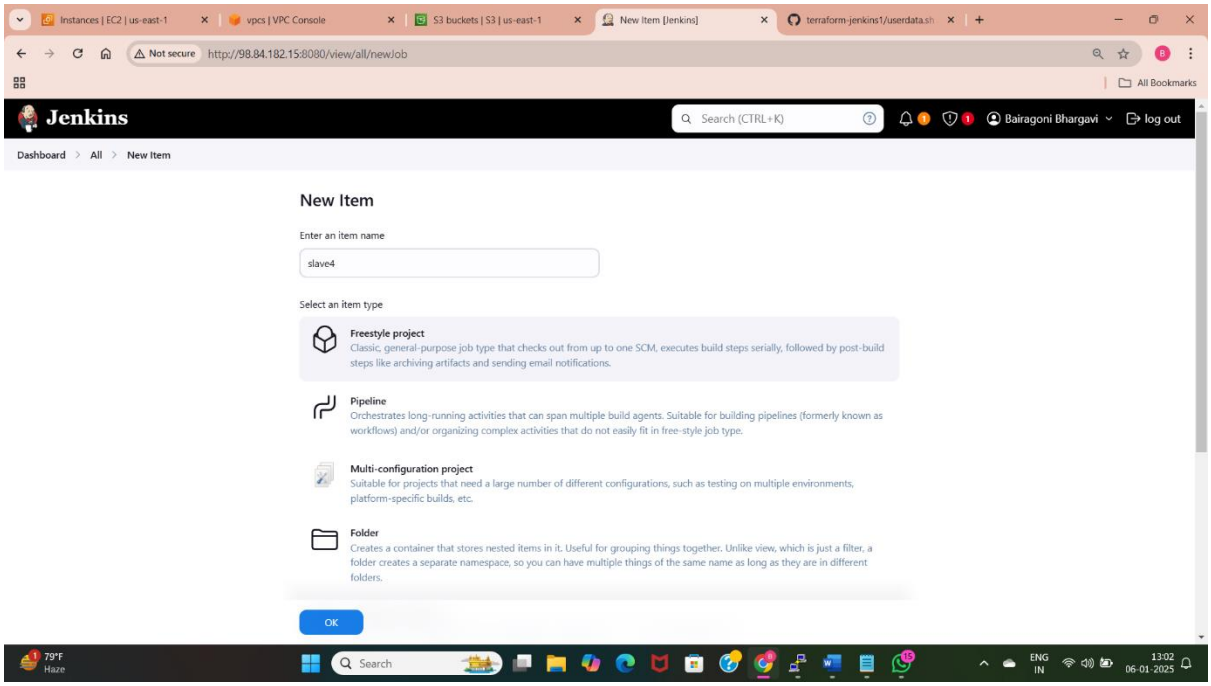
SLAVE-4:



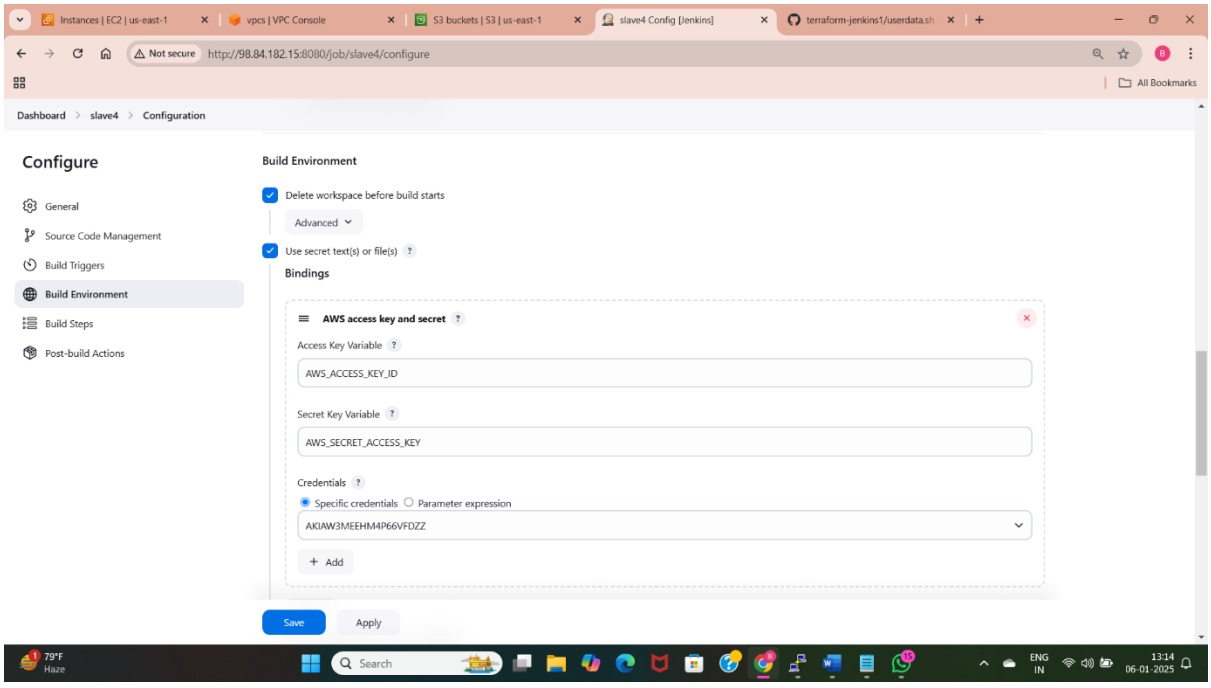
Now we need to deploy the application using automation and terraform. For that we need to select build periodically option and give syntax according to the requirement. Here we need the builds for every minute so (* * * * *) is given.



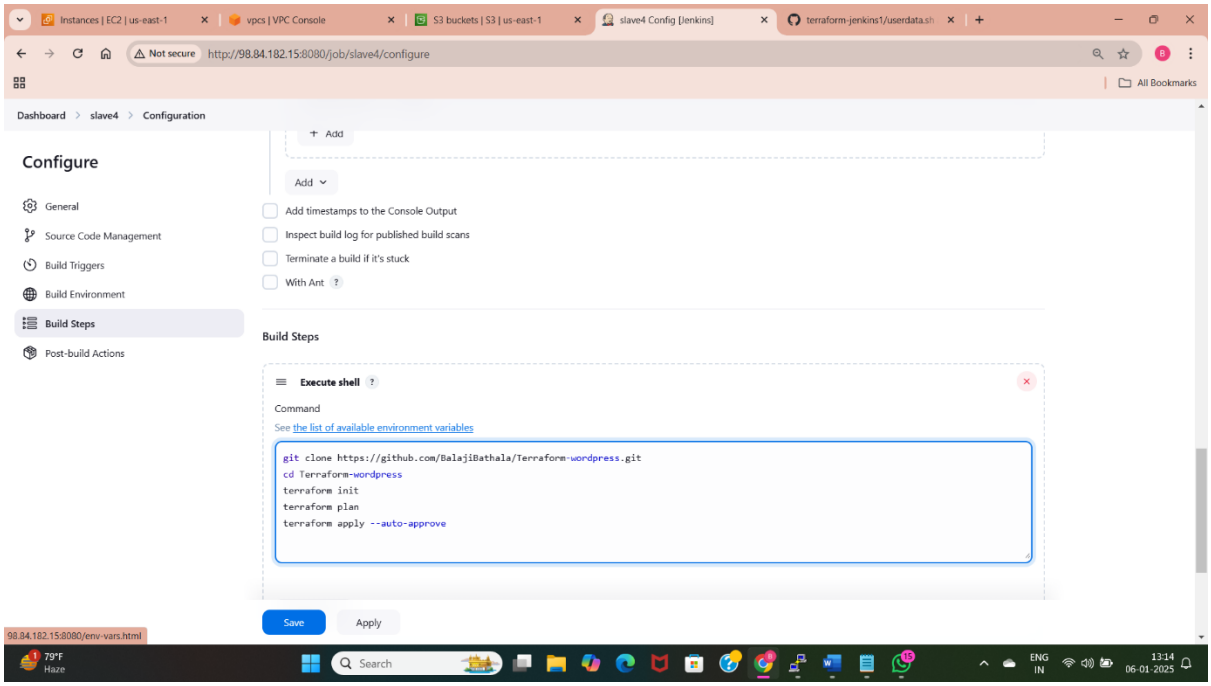
Now save the changes and check for the builds. A new build is started when we save the build periodically option.



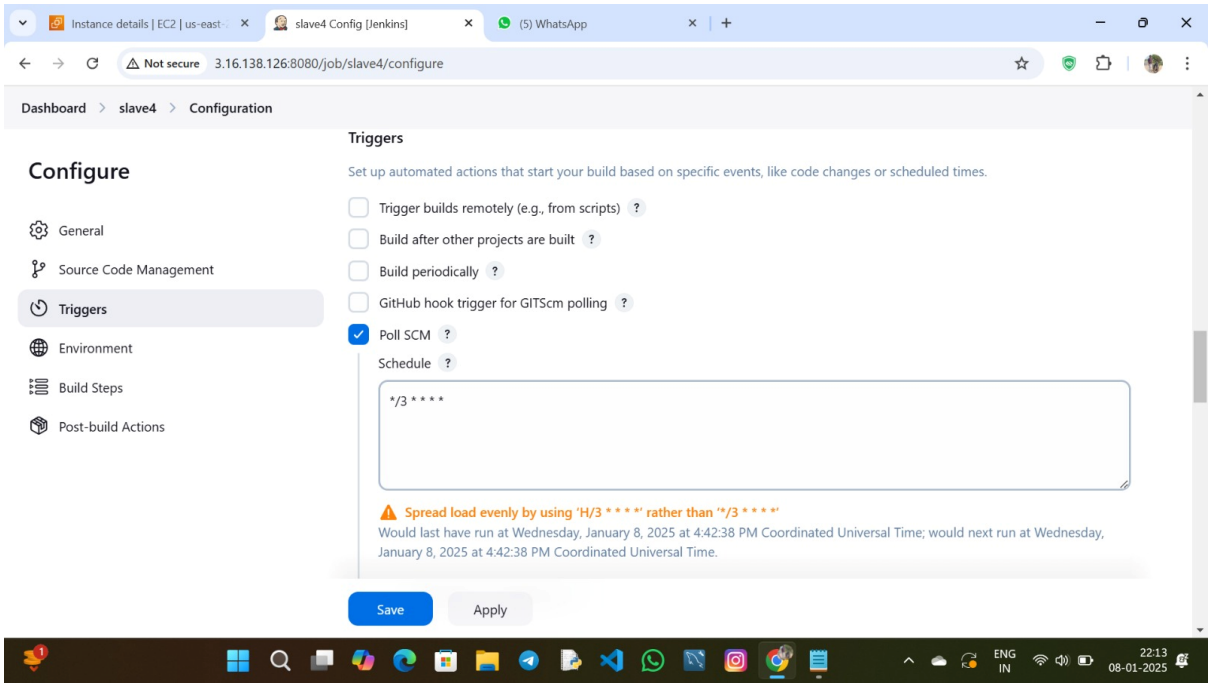
Create a new job for slave 4 for poll scm(automatic deployment).



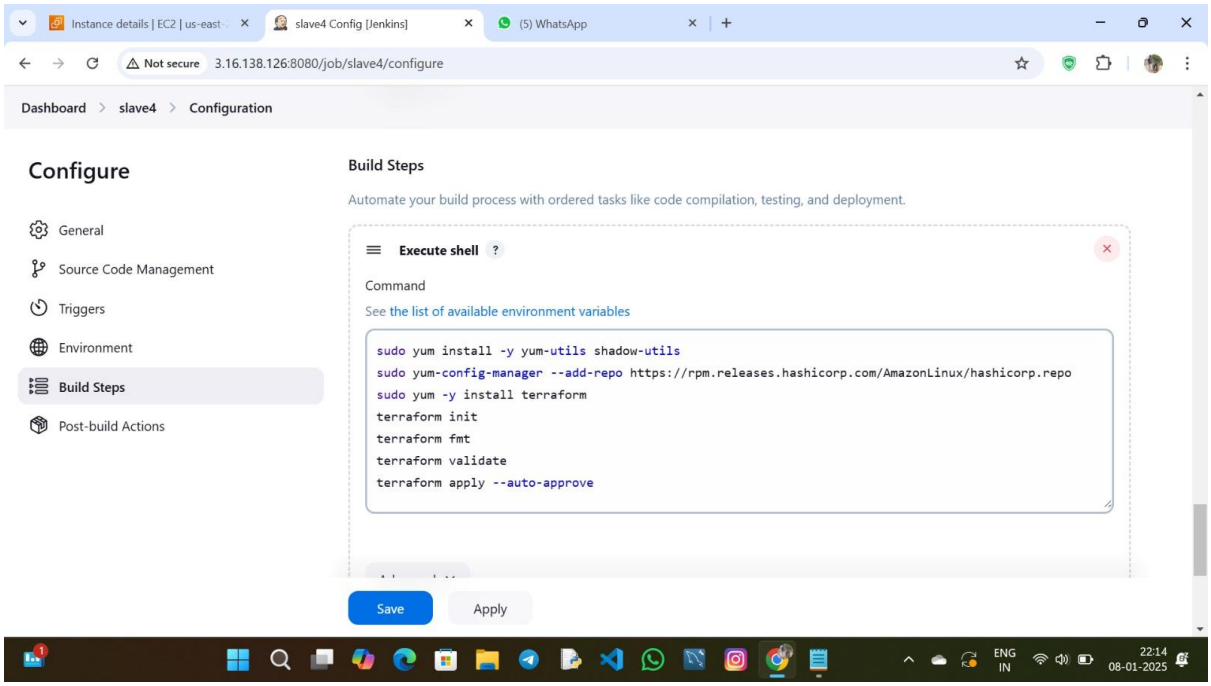
We need to provide the aws credentials to create the aws resources using the terraform automation. For that install the aws credentials plugin and provide credentials.



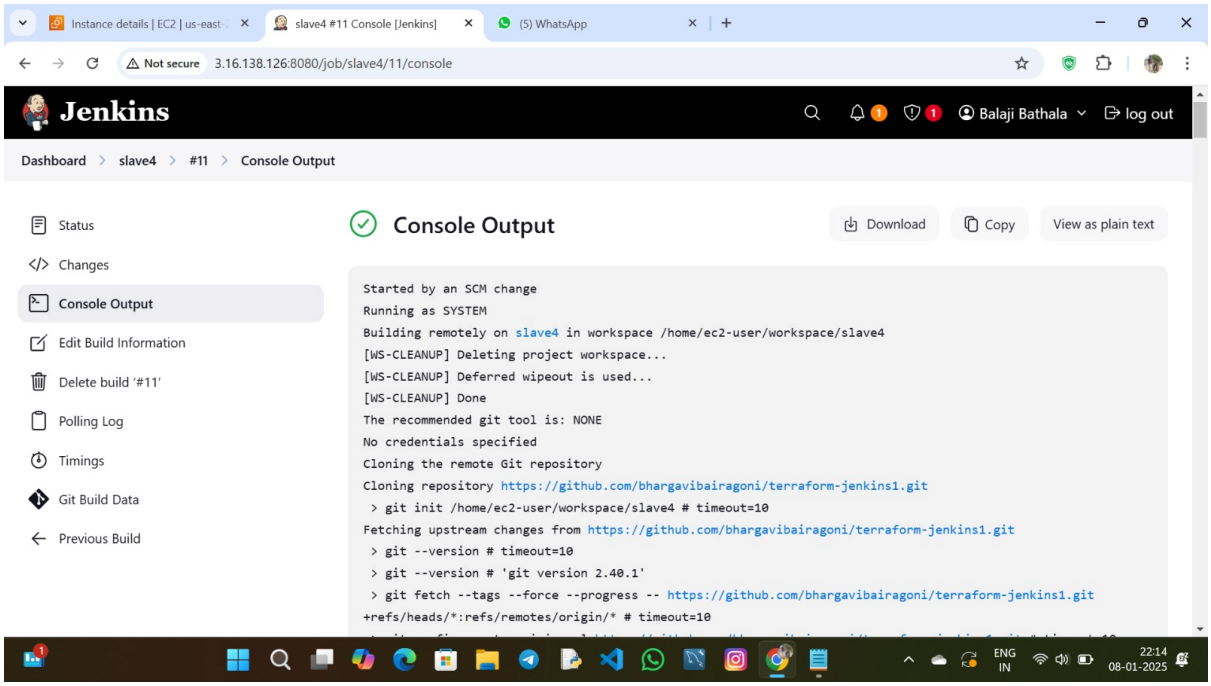
Now select the execute shell under build steps. Take the code from github, initialize the terraform in that files and see the plan what are going to create using that files and apply the changes using the "terraform apply –auto-approve".



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 playload url and application type, create webhook).

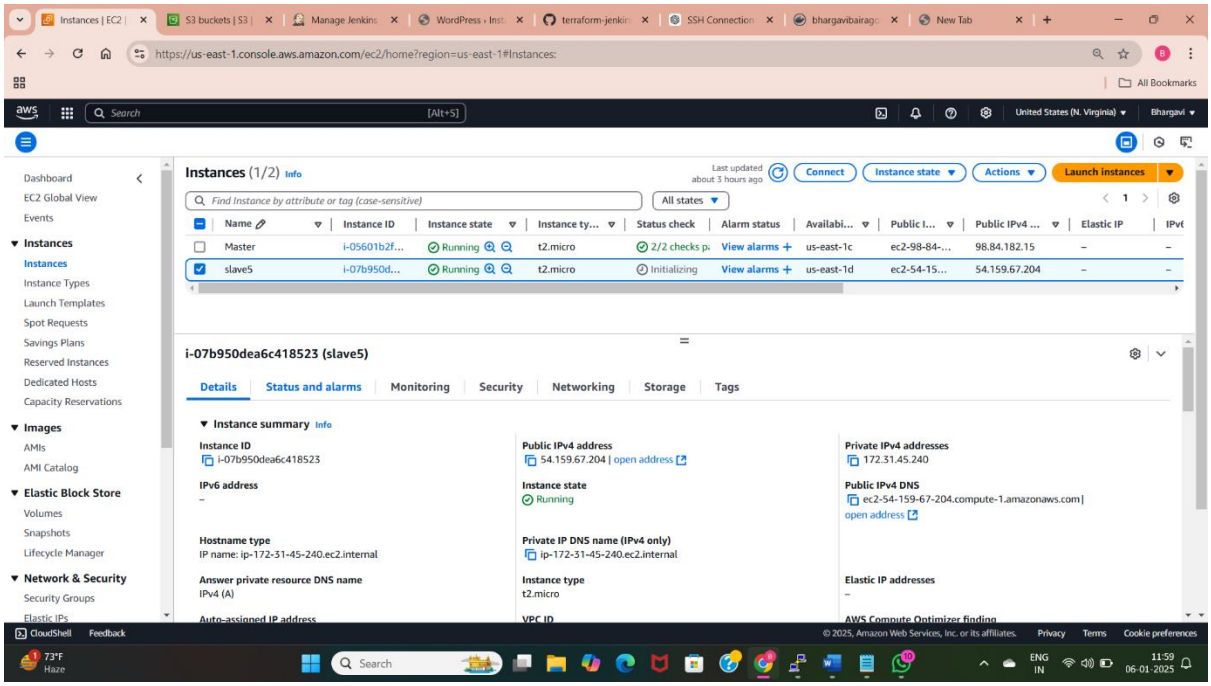


Go to build steps and select execute shell and give commands to install terraform, and initialize the terraform in the files and apply the changes.

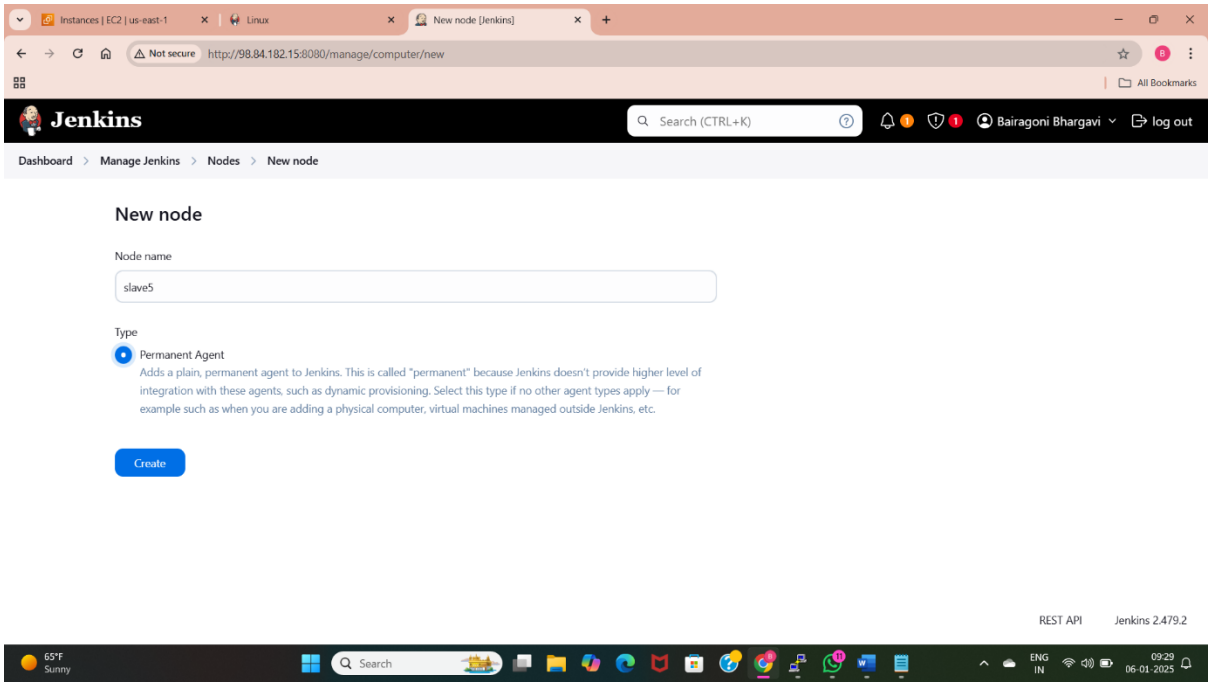


Now check for the builds. The build is started by SCM change(when we change code in the github) and creates the resources.

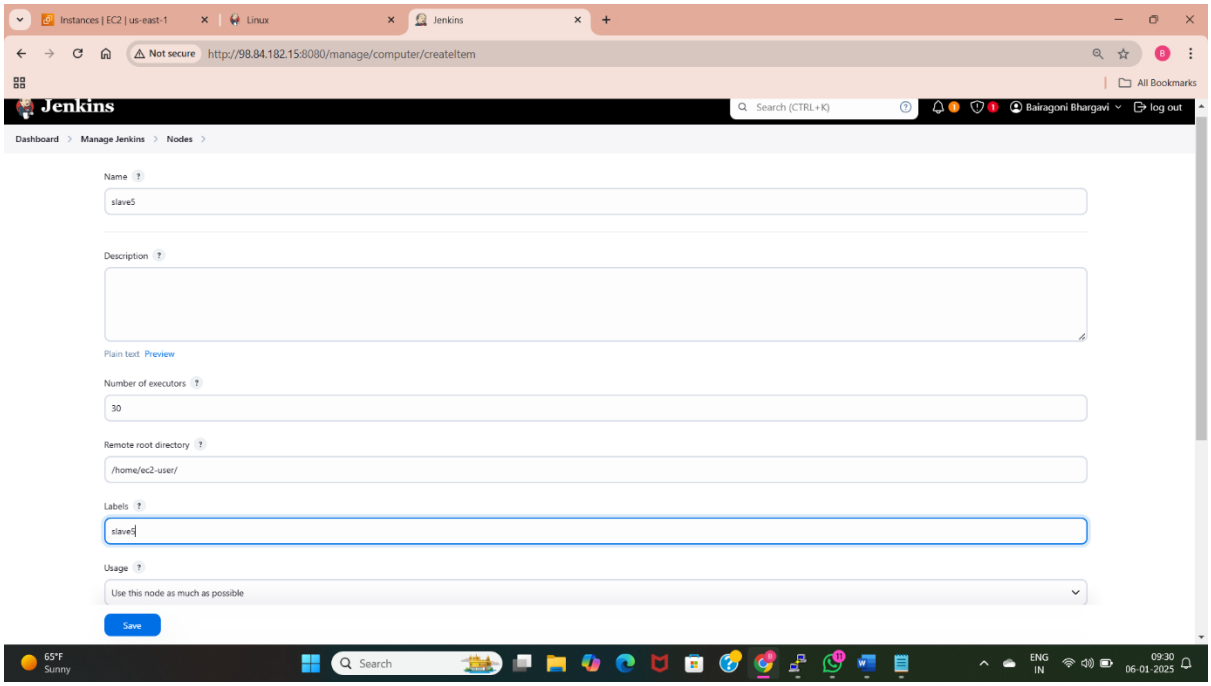
SLAVE-5:



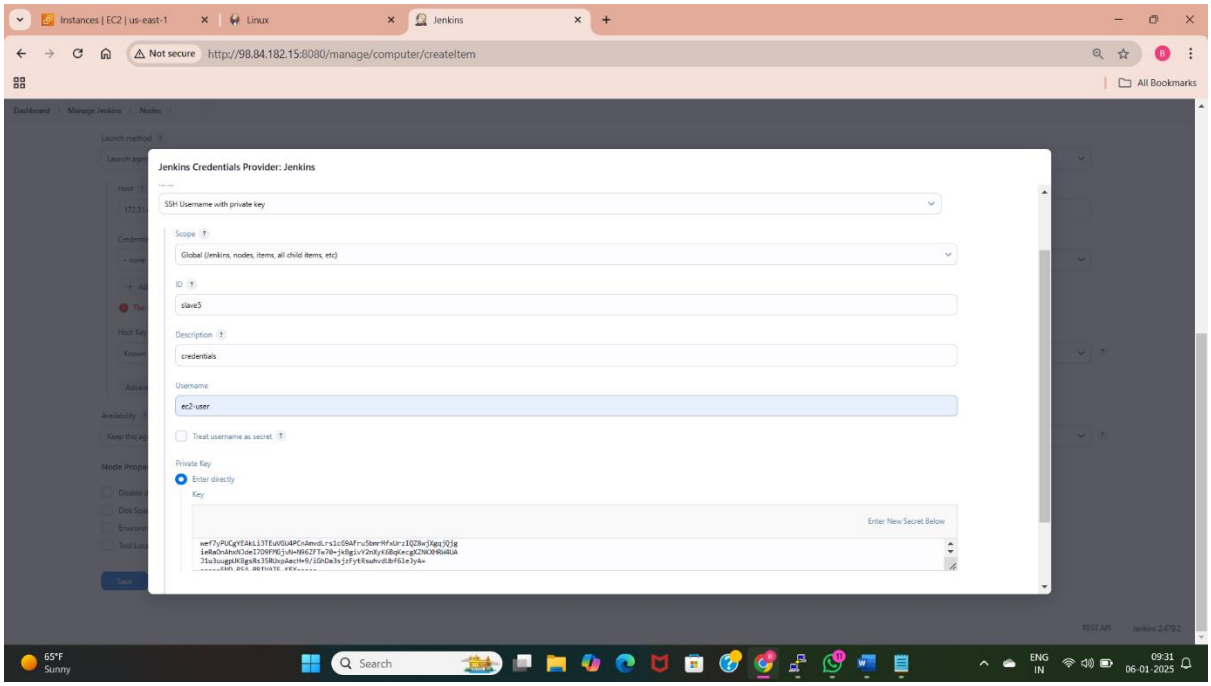
Create a new slave server and connect it with the master.



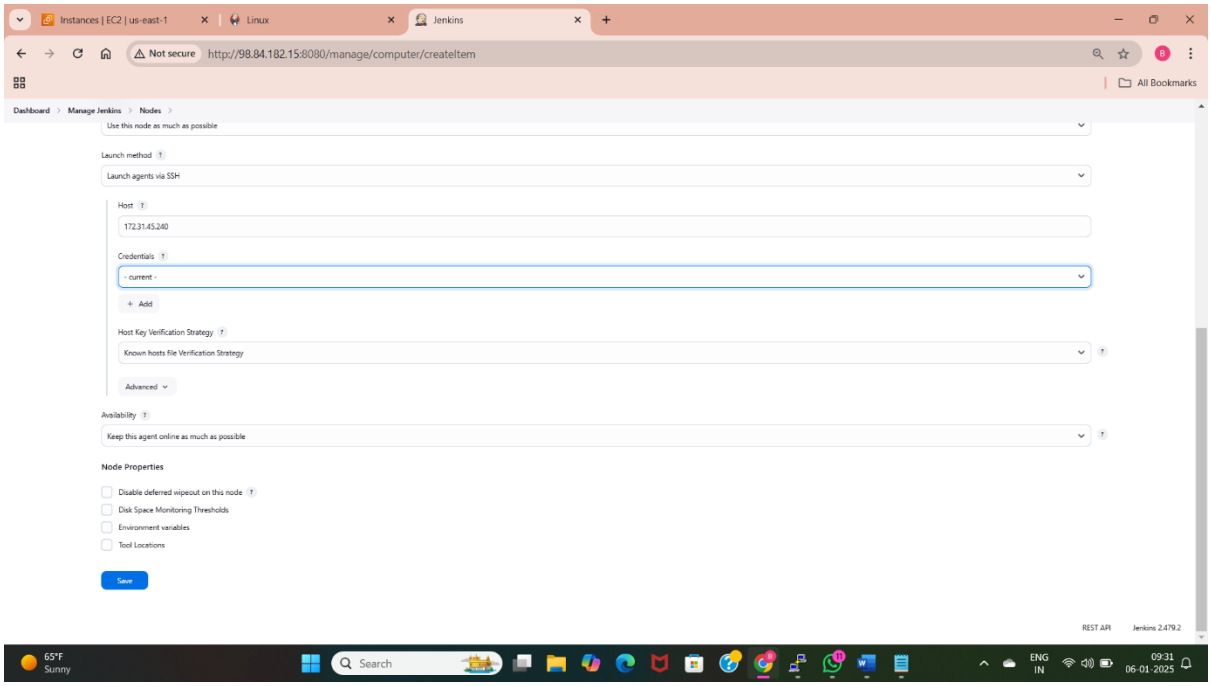
Now create a slave node by moving into manage Jenkins and select nodes and create new node.



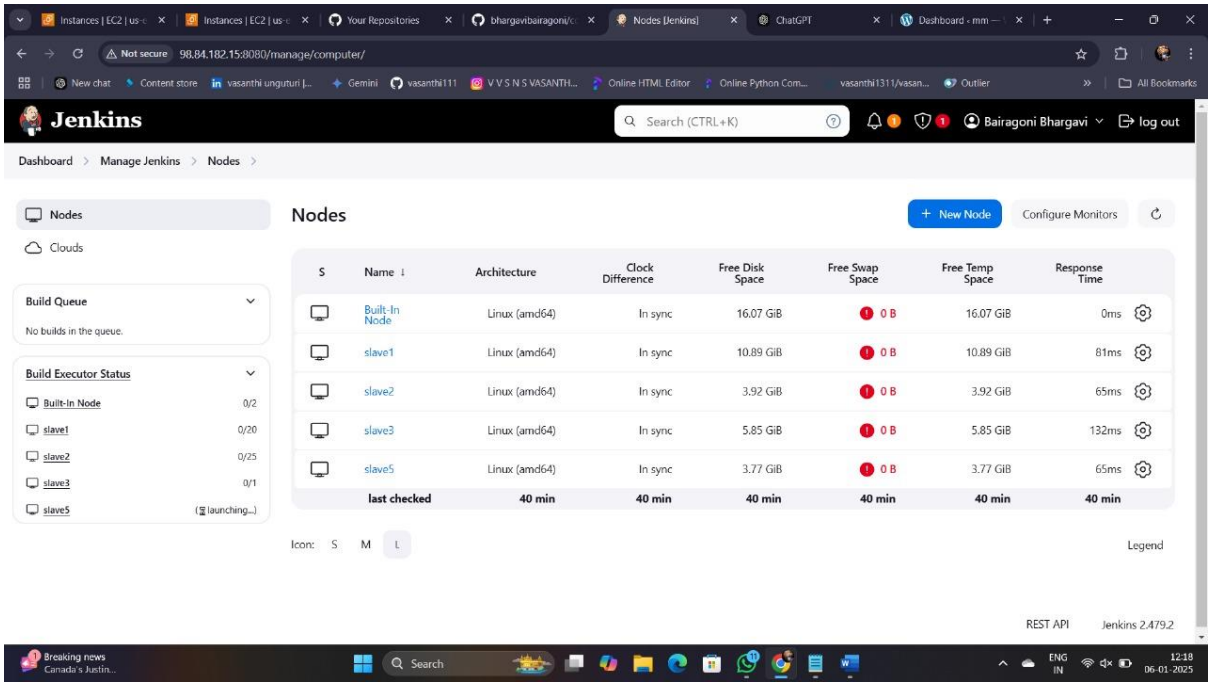
For creating new slave5 node give the node name, select number of executors and give remote root directory of the slave5 node, give label name.



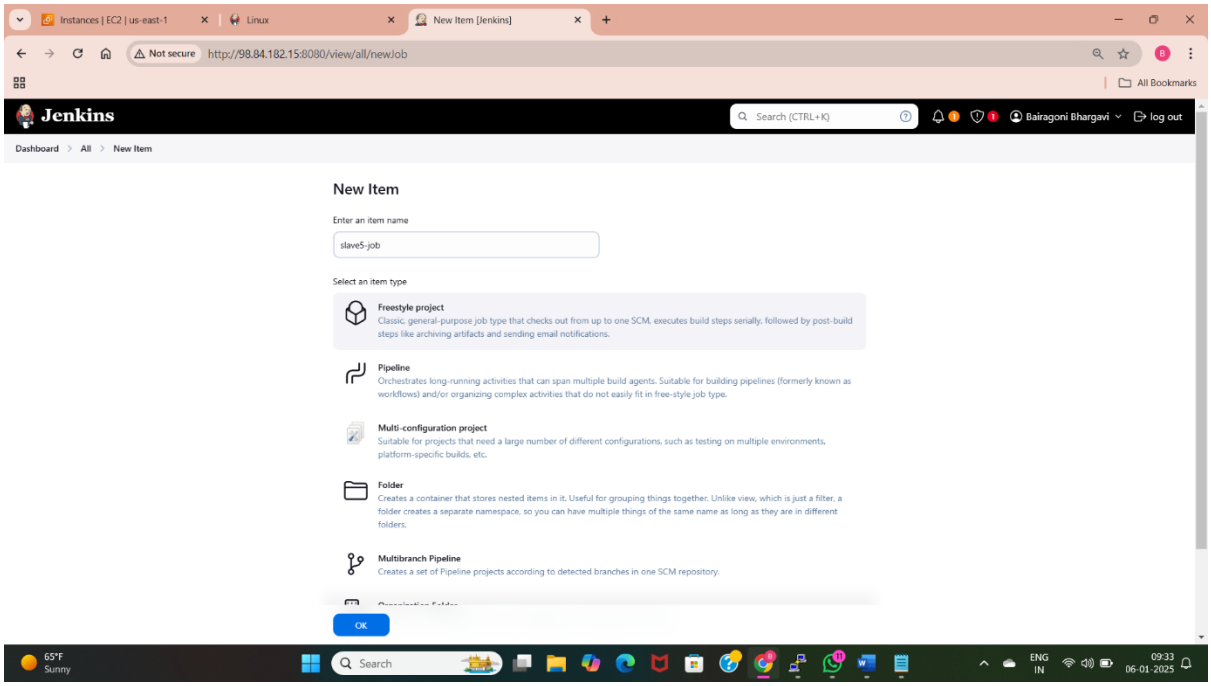
Now provide the credentials using SSH username with private key. Give id as any name, give description for credentials, give the username of the slave5 server, and select the option to enter directly private key



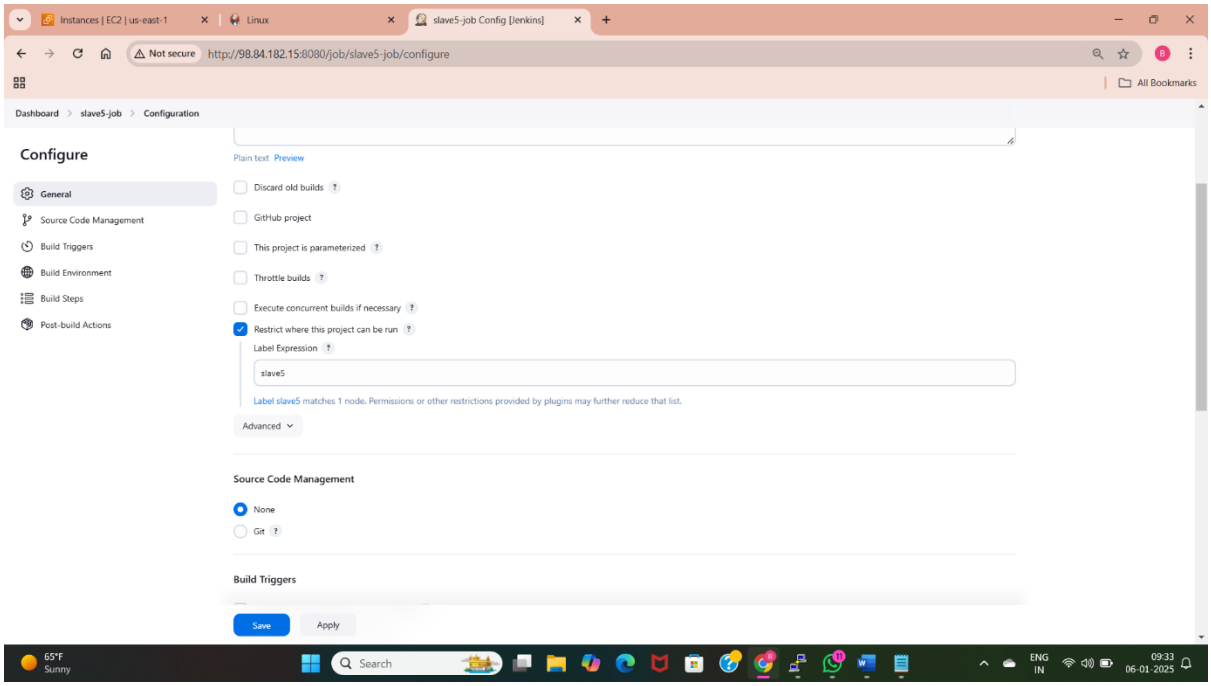
Now provide the public or private ip address of slave server as host. Give the above created credentials and provide host key verification for creating the node.



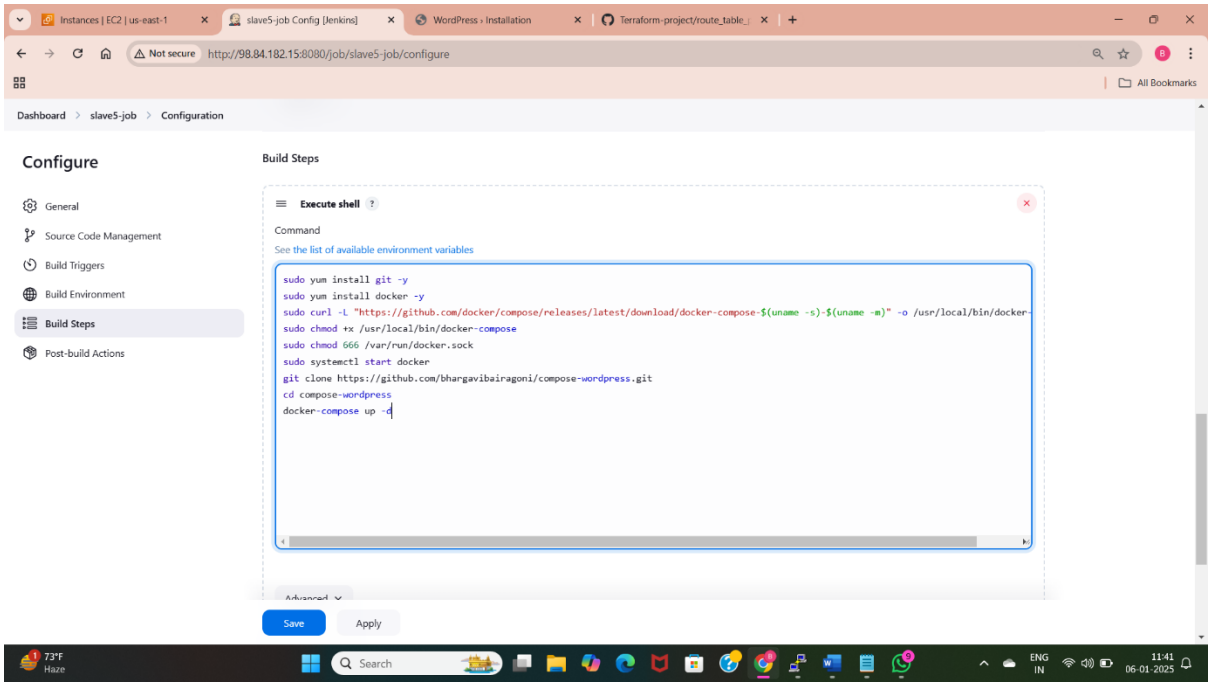
These are the created nodes by all the slave servers and they are launched successfully.



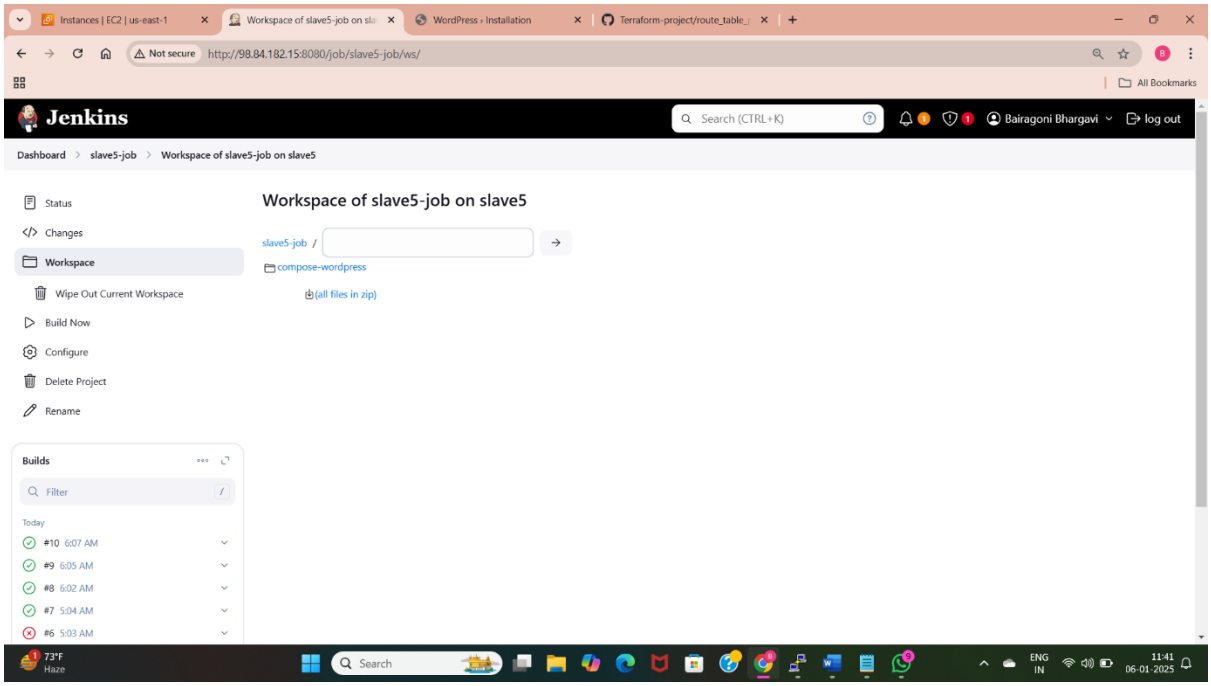
Now create a new freestyle job for the slave5 server to deploy the application.



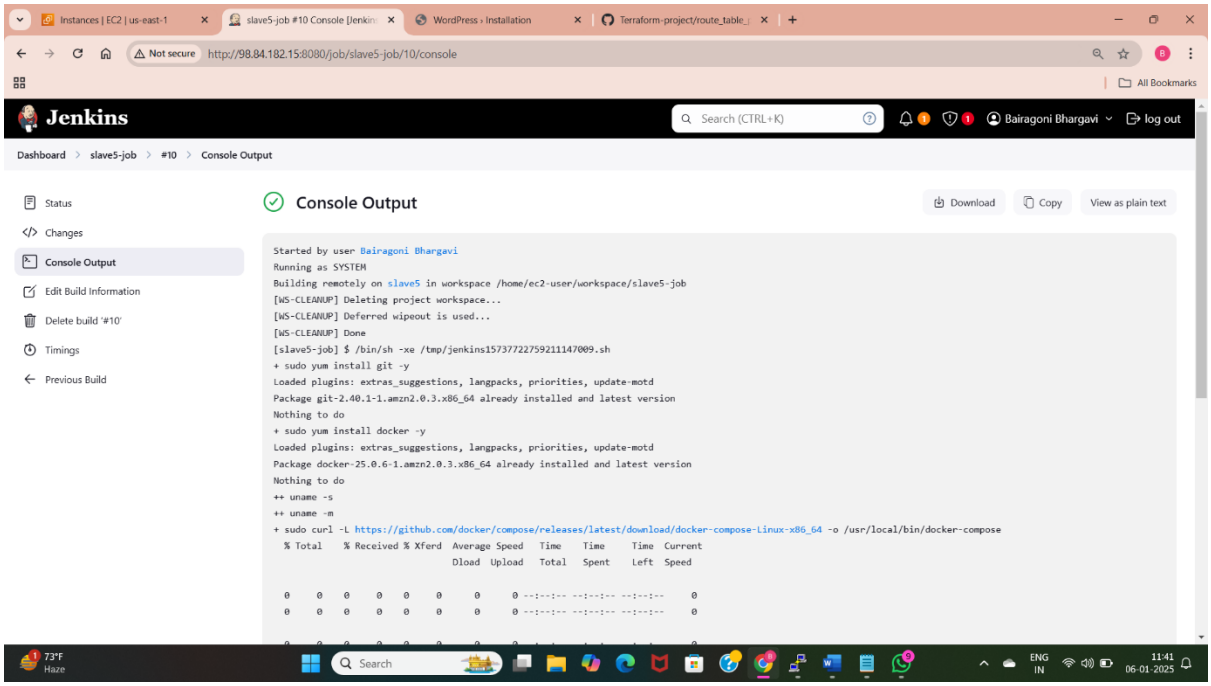
Now select the above created label to build the code(so all the build will be performed by slave 5 server node).



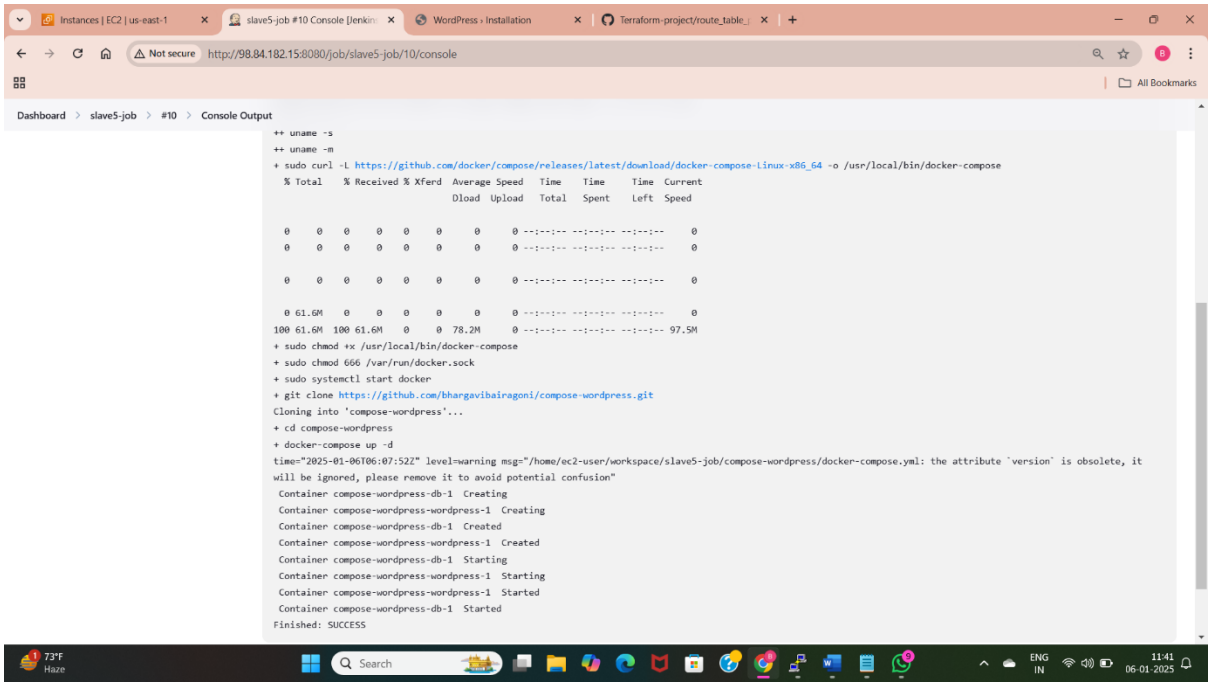
Now under the build steps select execute shell and write the bash script which are used to deploy the wordpress application using the docker.



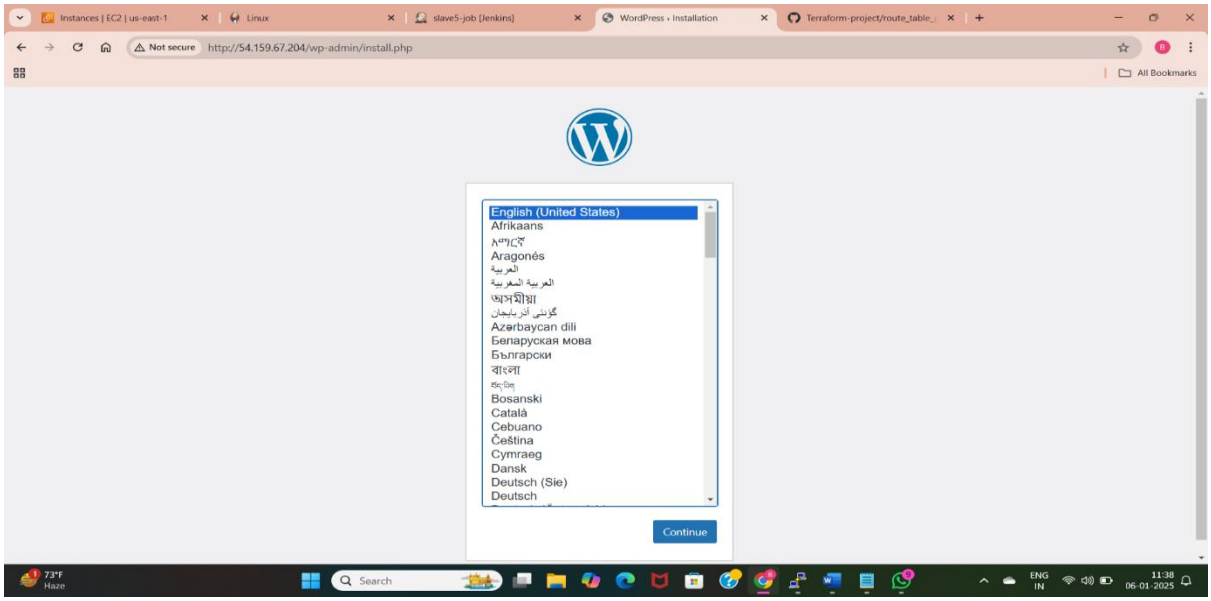
Now save and click on build now option. The build is success and we can see the compose-worpress file which is having the code.



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



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

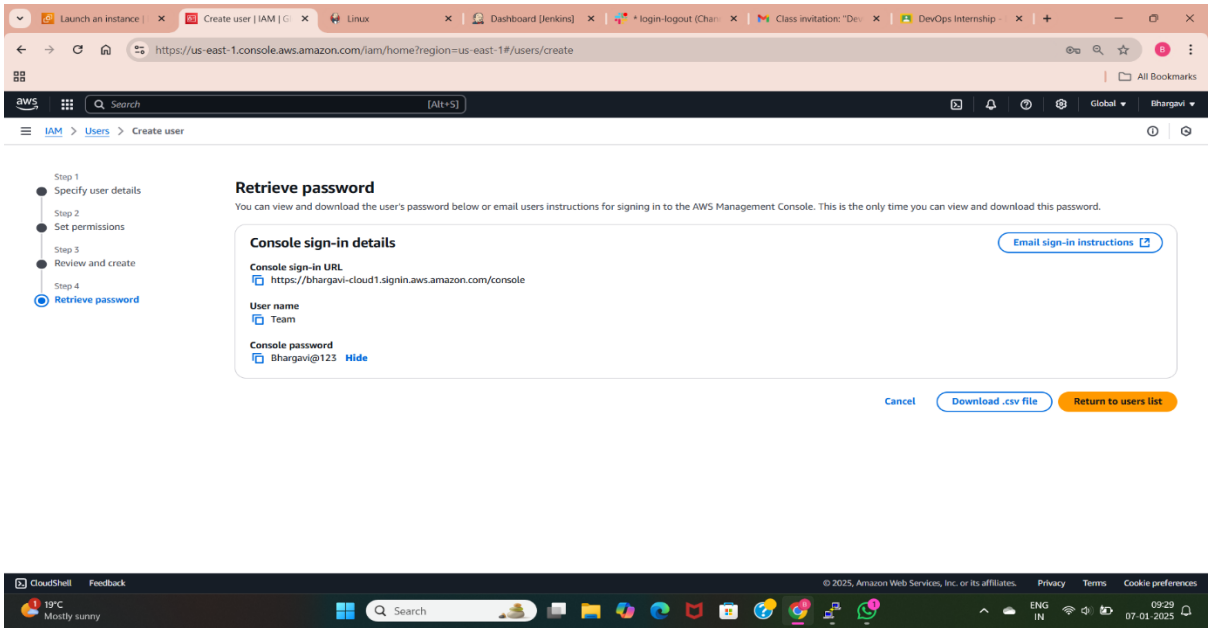


Now copy and paste the public IP address of the instance and paste in browser we can able to access the wordpress application.

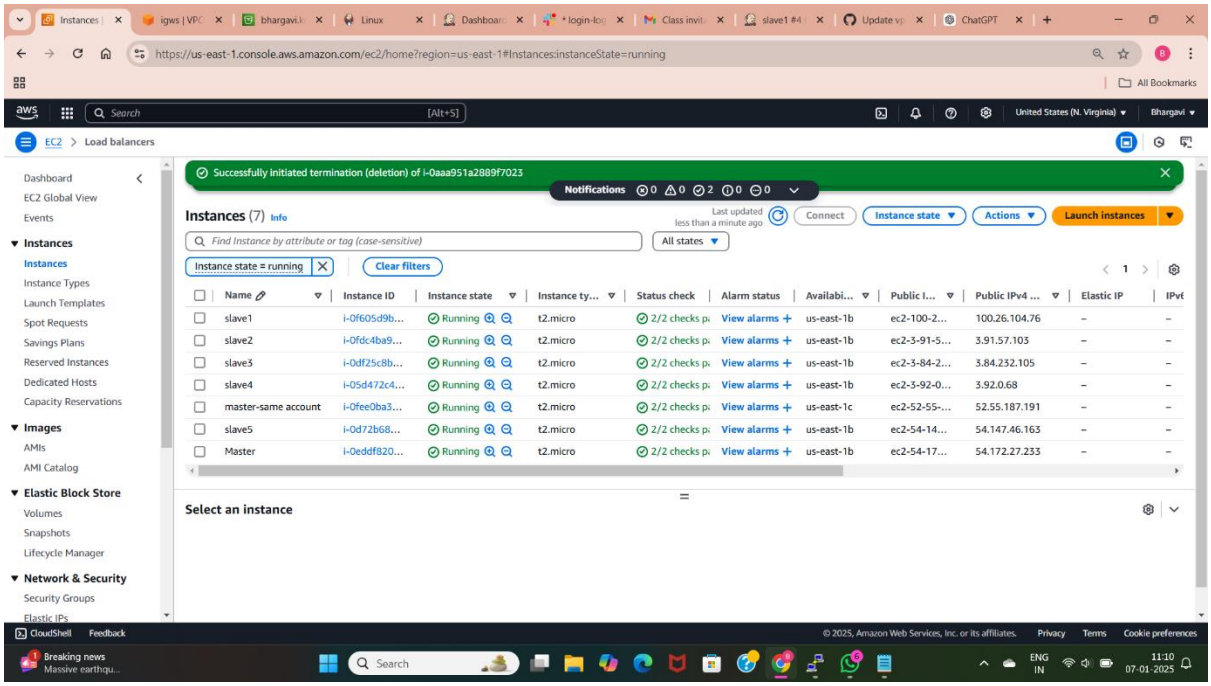
METHOD-4 :

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.

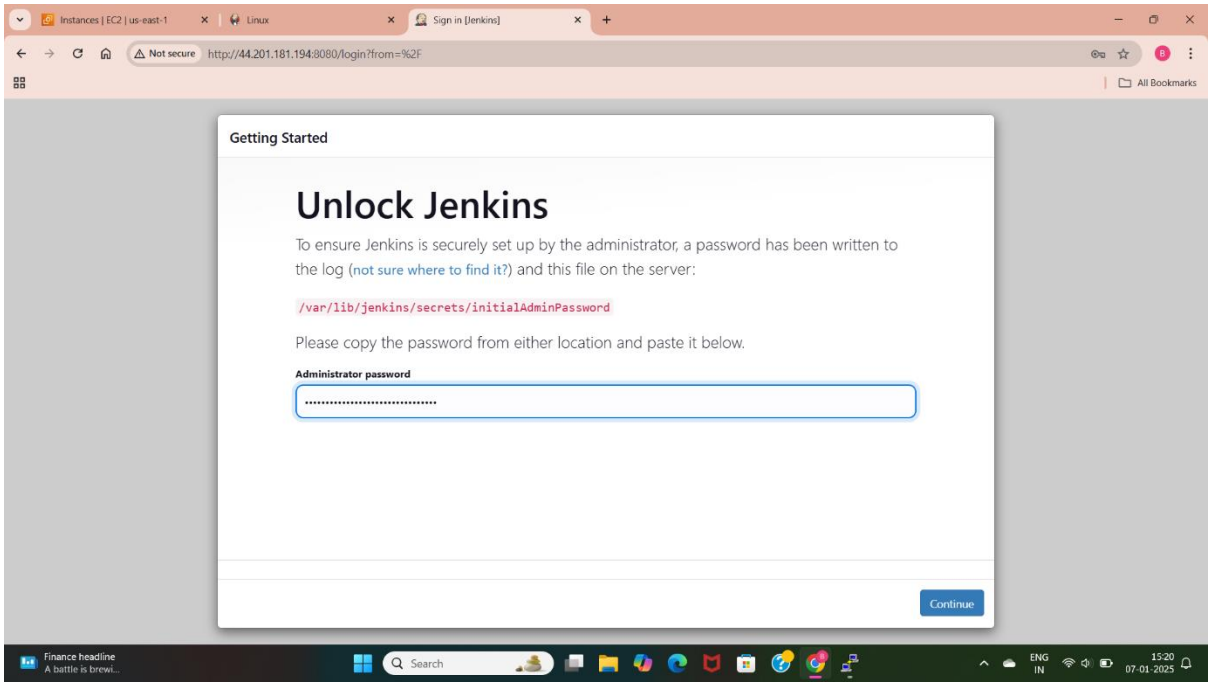
- Master, slave-1: deploying wordpress application using k8s – Bhargavi
- Slave-2: deploying wordpress application using docker and docker-compose- Vamsi
- Slave-3,4: deploying wordpress application using terraform manually and automatically- Vasanthi
- Salve-5: deploying wordpress application using 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.

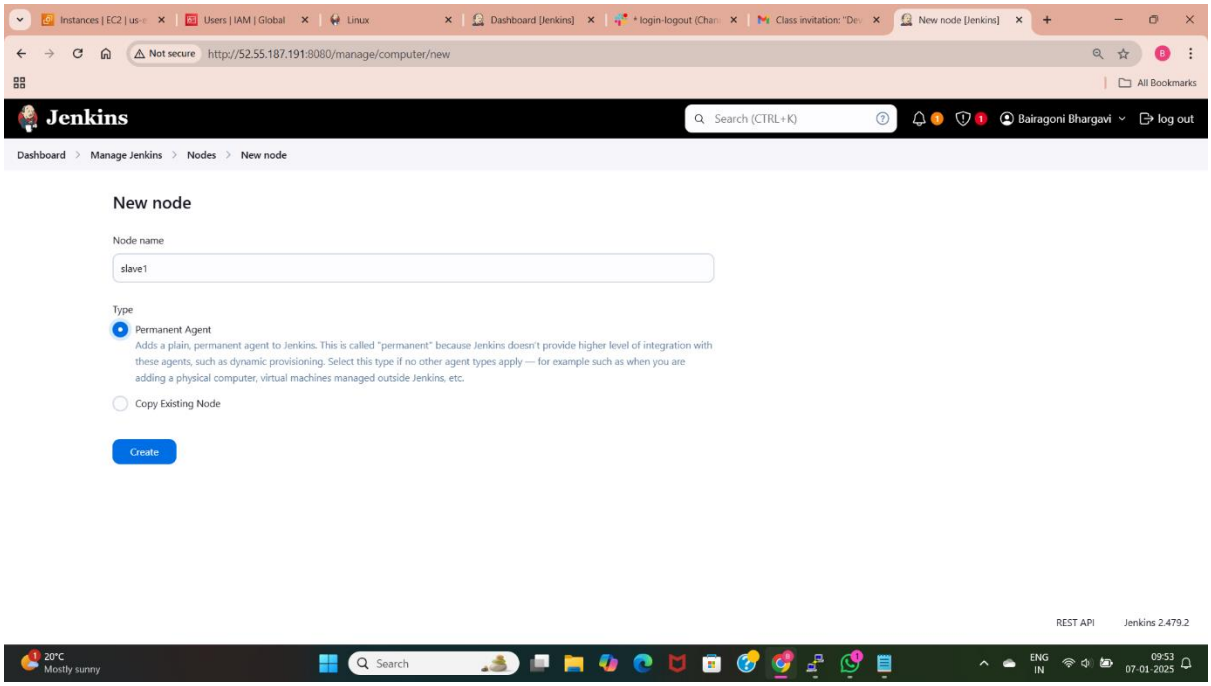


Here five slave servers with one master server is created by the team.

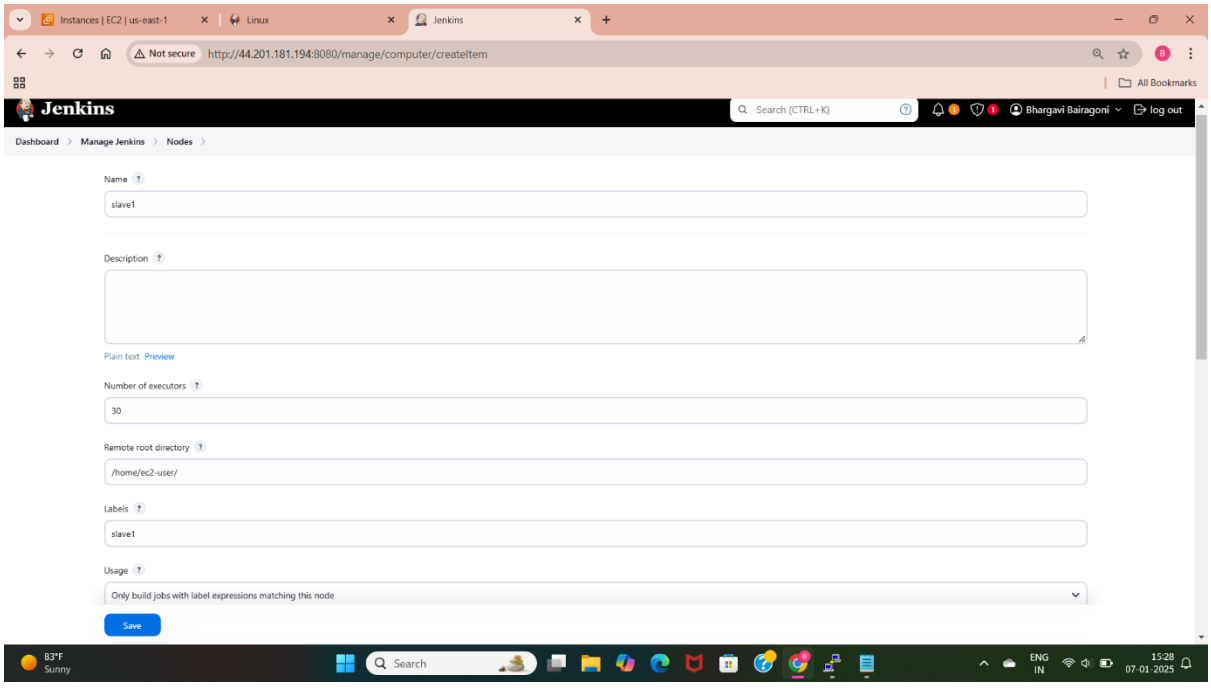


Now install the Jenkins in master server and start the Jenkins, access the Jenkins by giving Jenkins password.

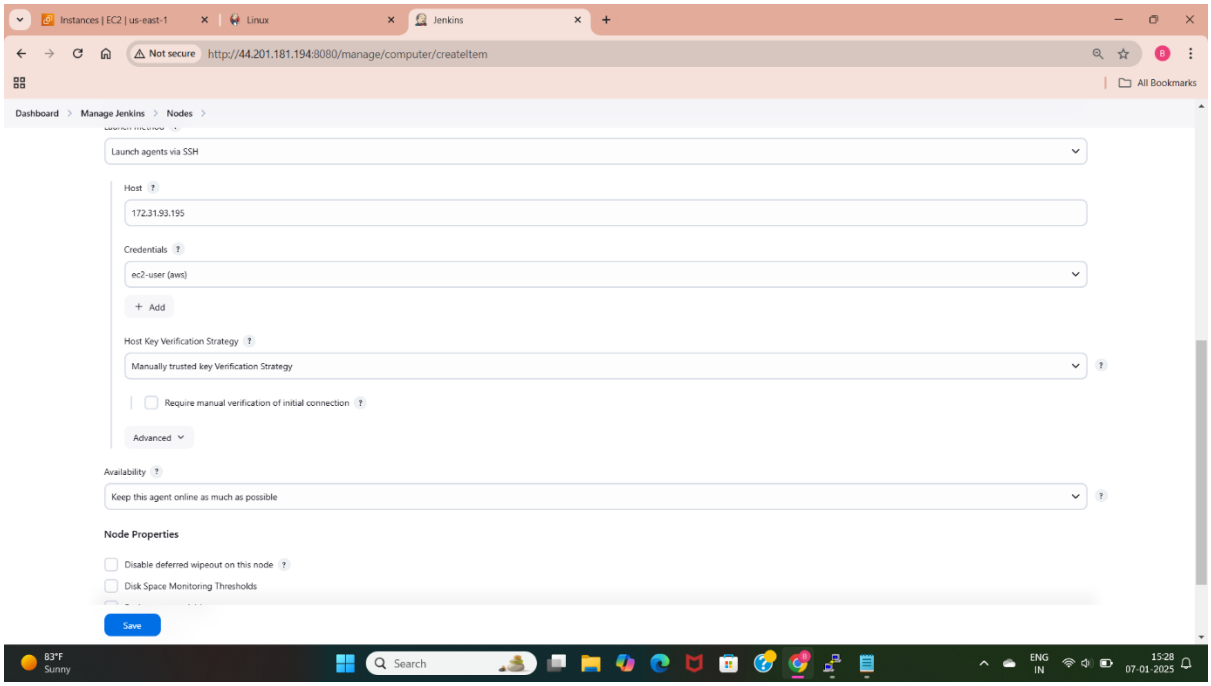
SLAVE-1:



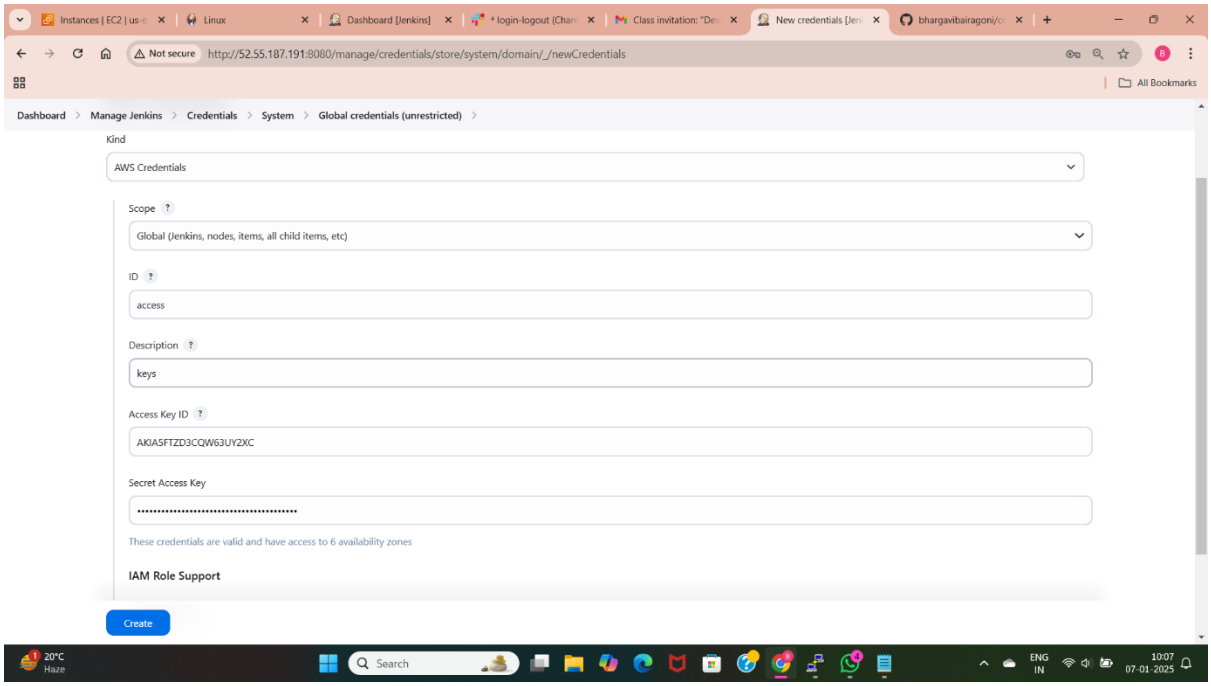
Now create a slave node by selecting manage Jenkins> nodes and give name for the node.



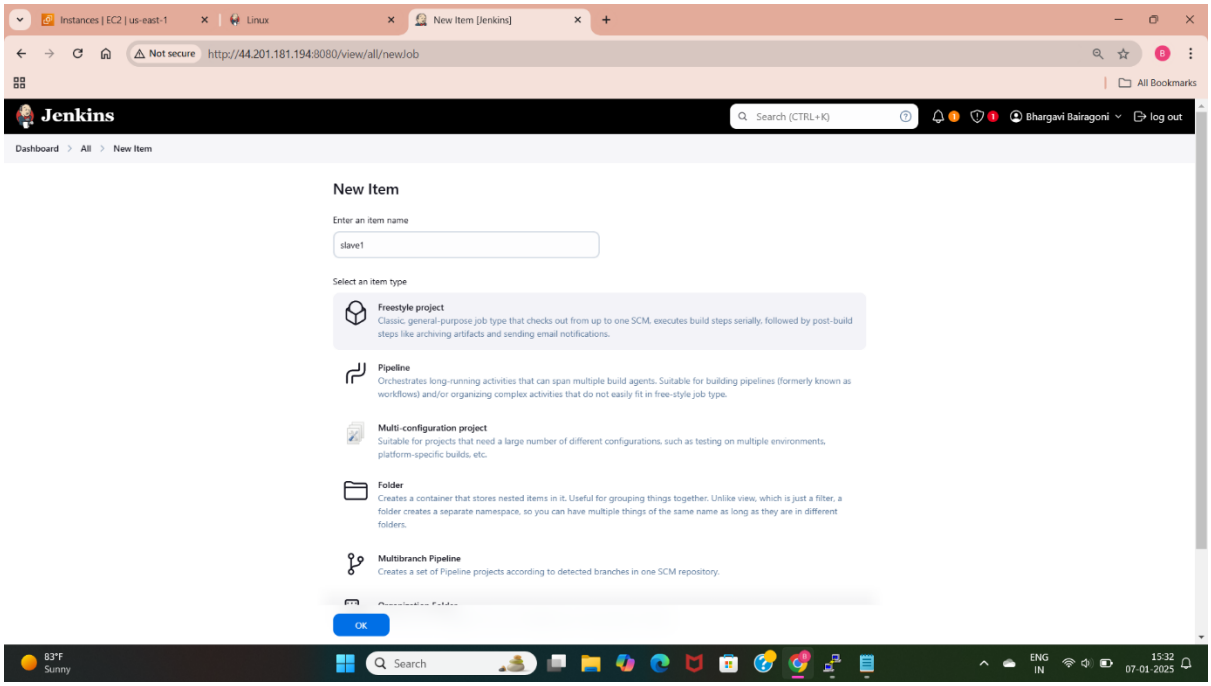
Creating a slave 1 server, giving number of executers, root directory of the slave server, create label to identify.



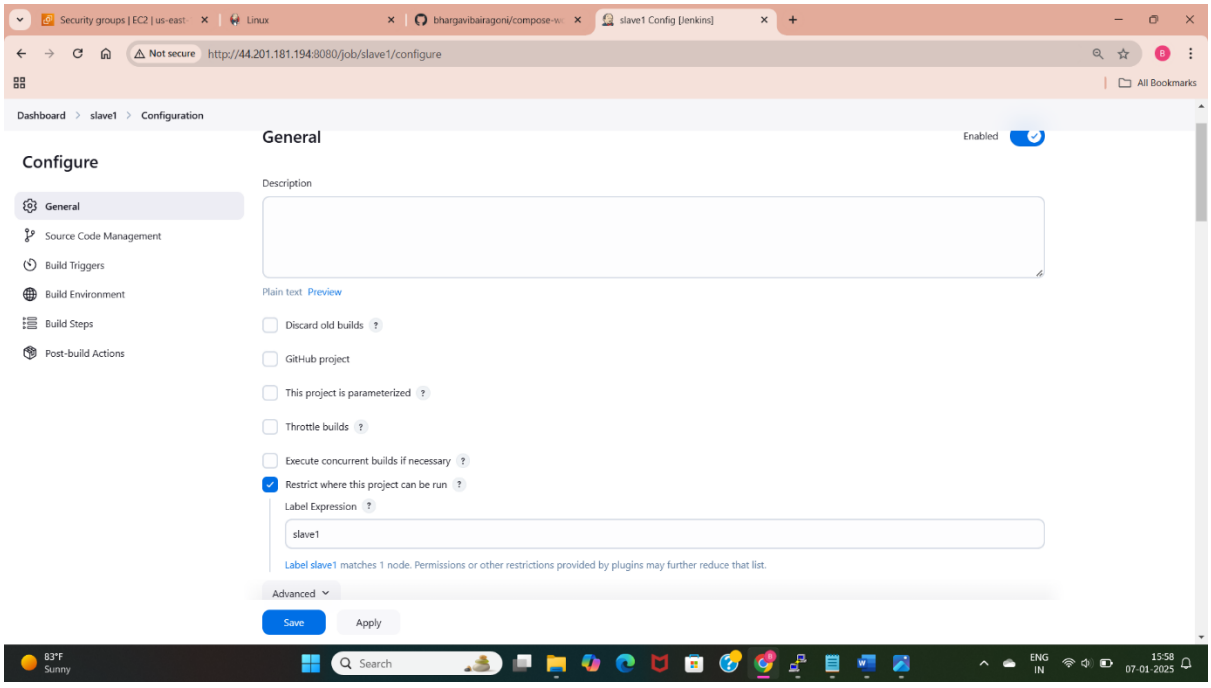
Now select the launch method as agents via SSH, give host, and credentials of the master to connect with the slave and select host key verification strategy.



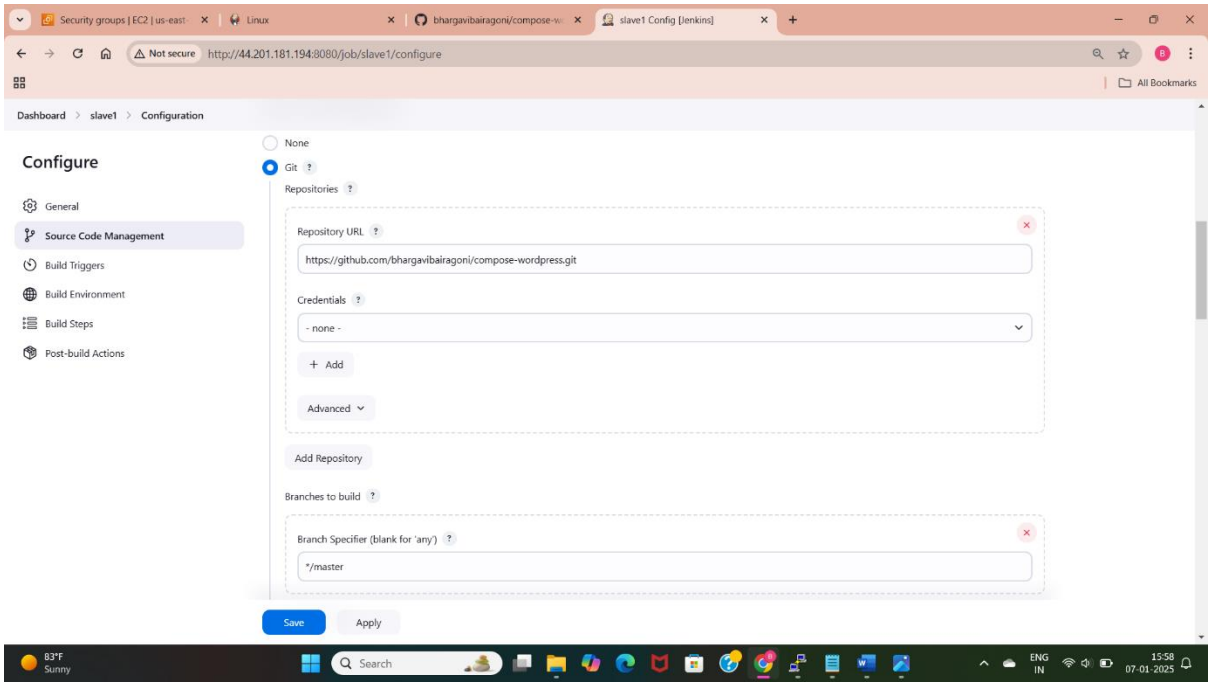
Here am providing the aws credentials by giving access keys and secret access keys by installing aws credentials plugin.



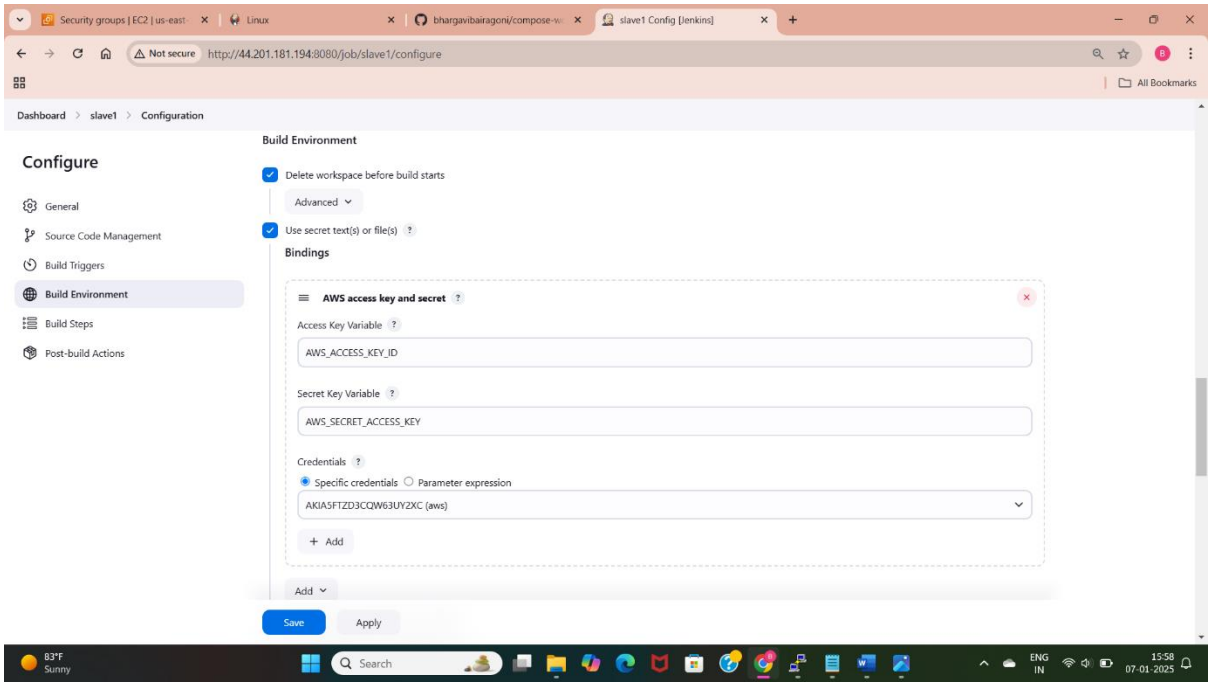
Creating a new free style job in Jenkins for slave1.



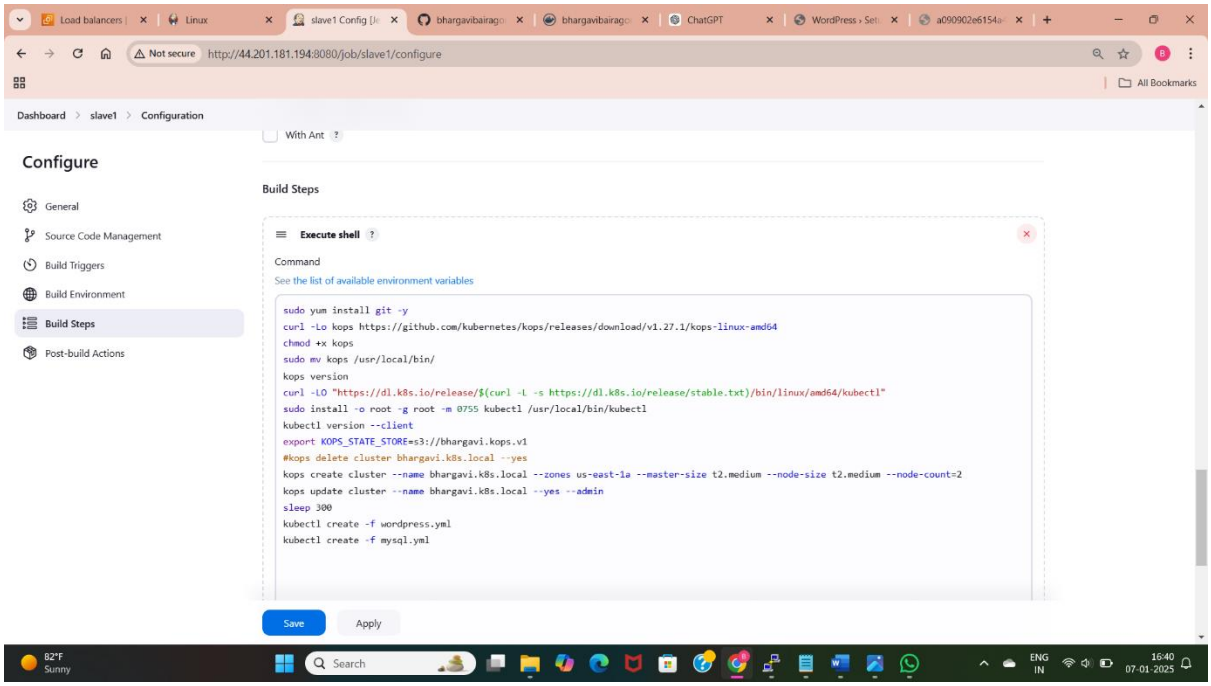
Here am restricting the builds on Jenkins by selecting slave1 label.



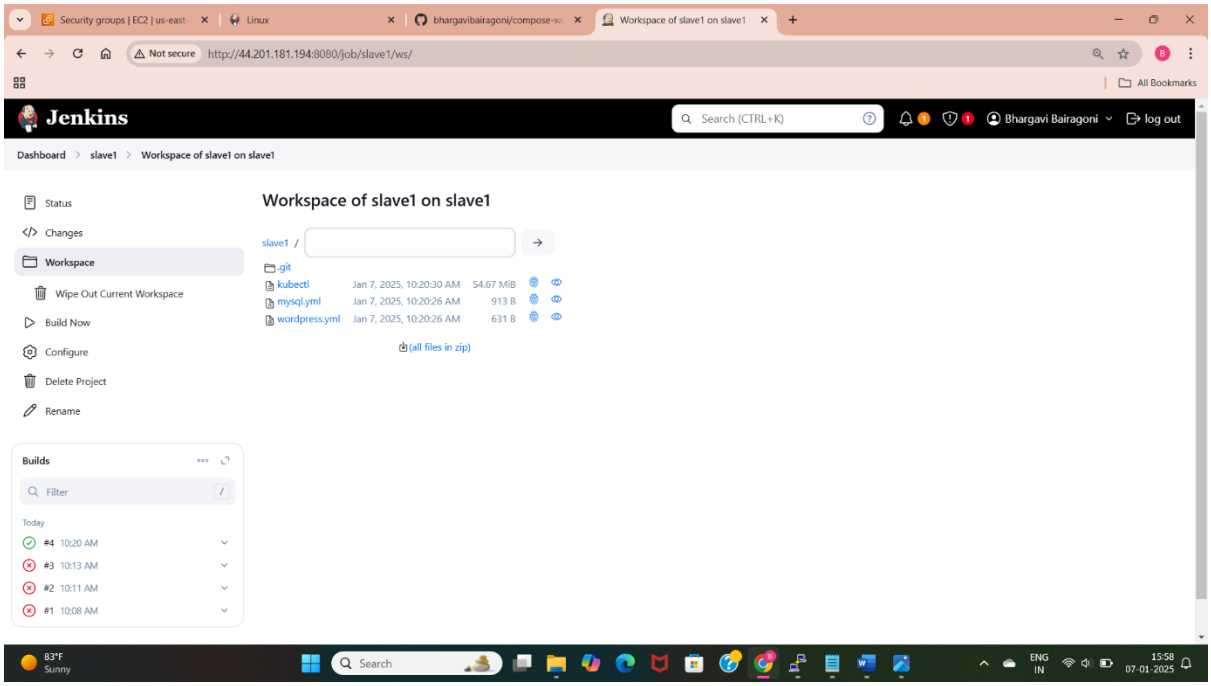
Select git option under source code management and give repository url and branch name where we are having the files.



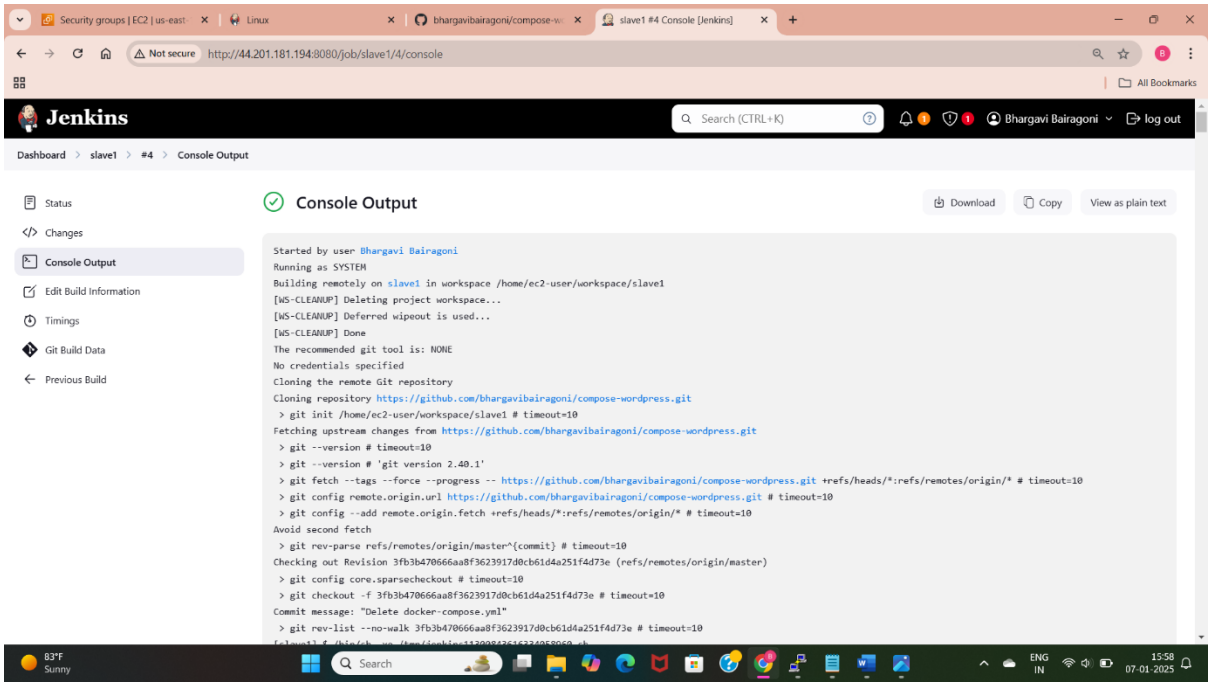
Under build environment select the delete workspace before build starts(it will deletes the before build files and newly downloads the files). Give the access keys and secret access keys by selecting “use secret text or files”.



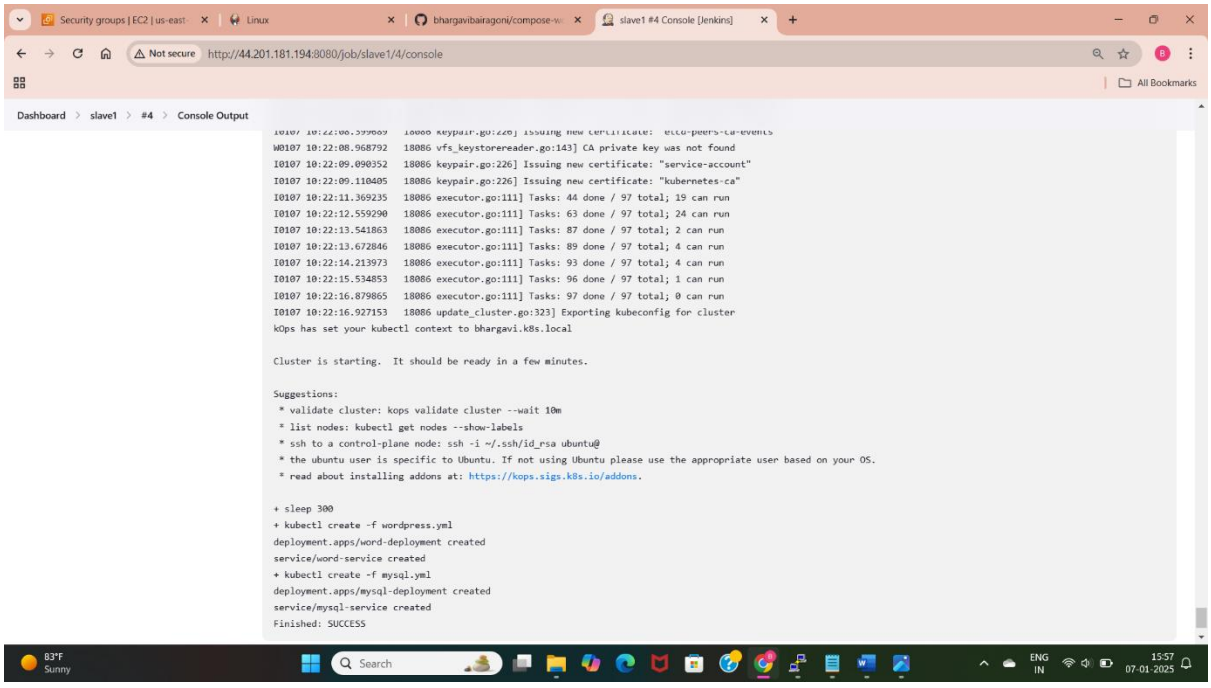
Select execute shell under build steps and give shell commands to install git, kops, kubectl , create a s3 bucket, create clusters(nodes and control plane) using kops create deployment and service files.



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



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



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
```

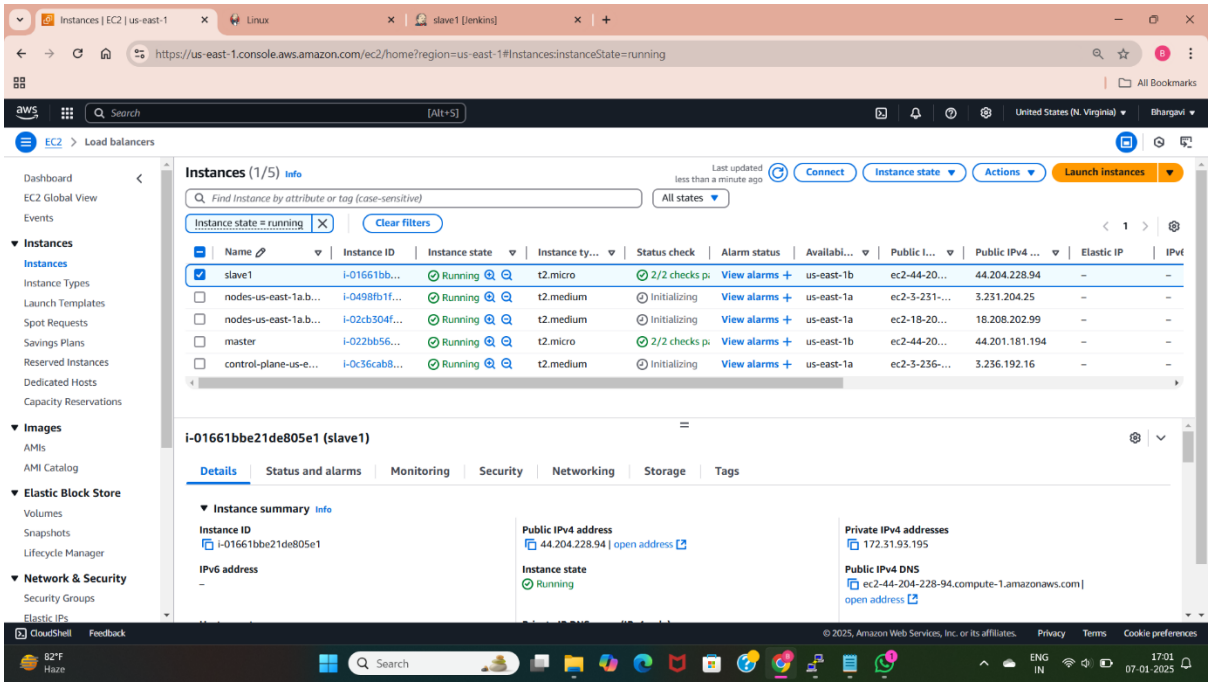
This is the deployment and service file for creating mysql database. We can see 2 replicas of this deployment.

ec2-user@slave1:~/workspace/slave1

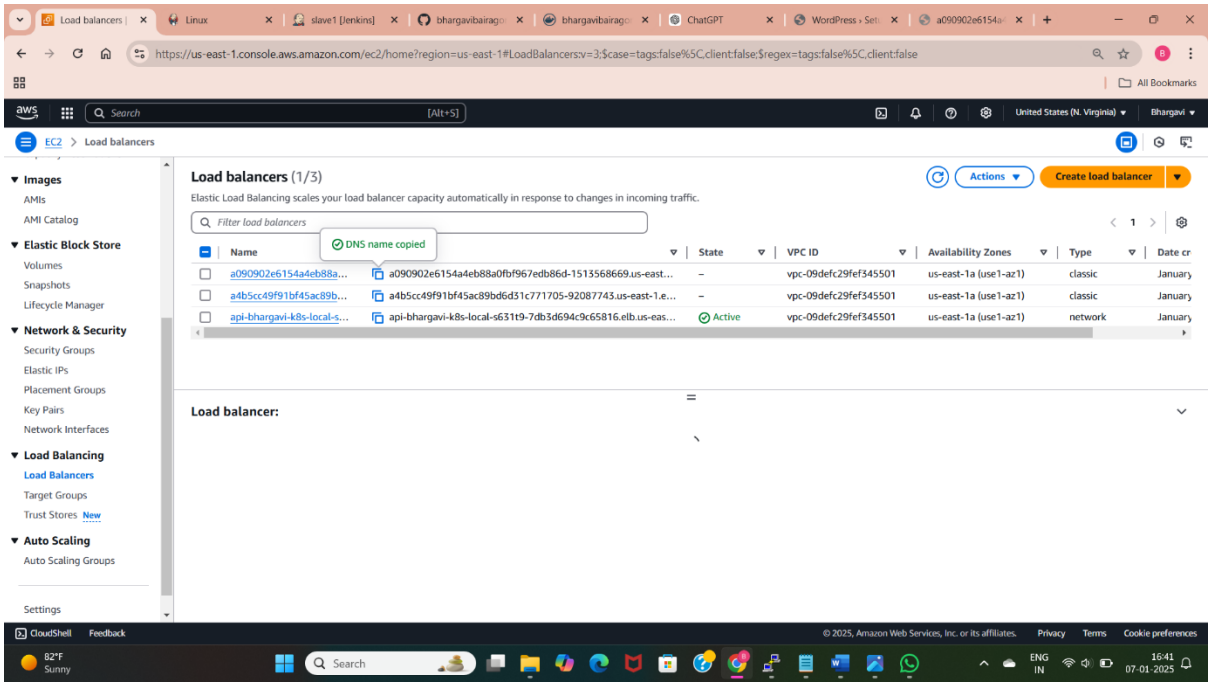
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: word-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: wordpress
  template:
    metadata:
      labels:
        tier: wordpress
    spec:
      containers:
      - name: wordpress-container
        image: bhargavibairagoni/wordpress:wordimage
        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
```

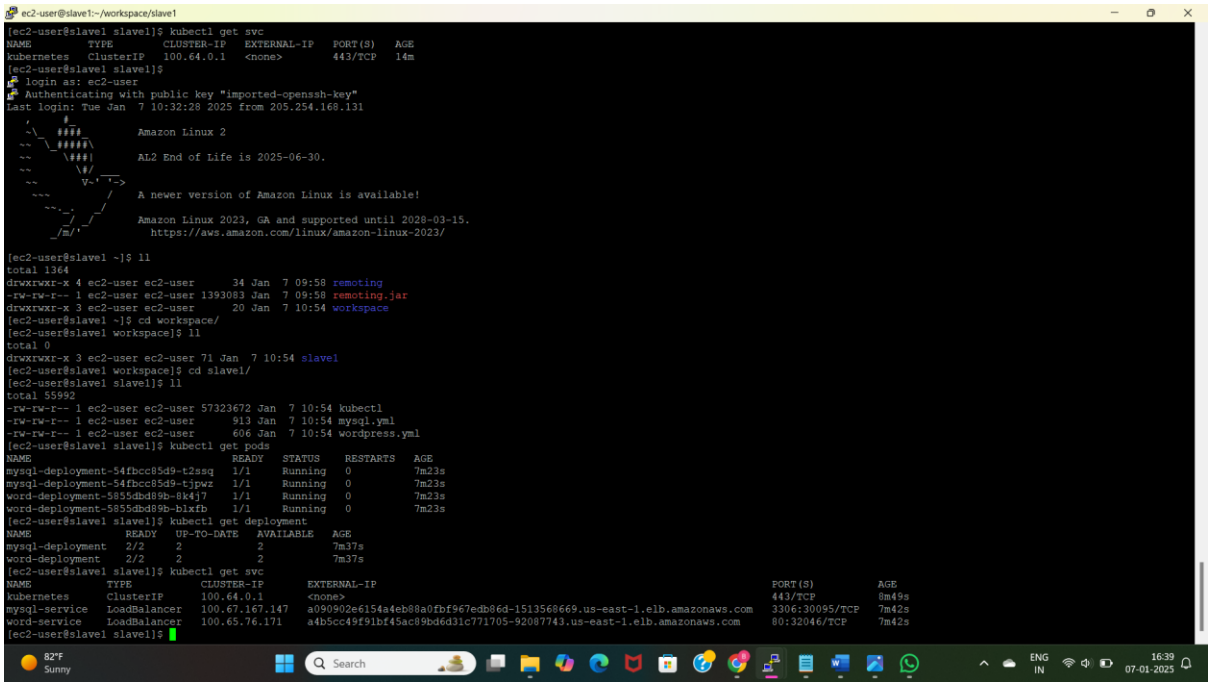
This is the deployment and service file for wordpress. It is running on the port number 80 with two replicas.



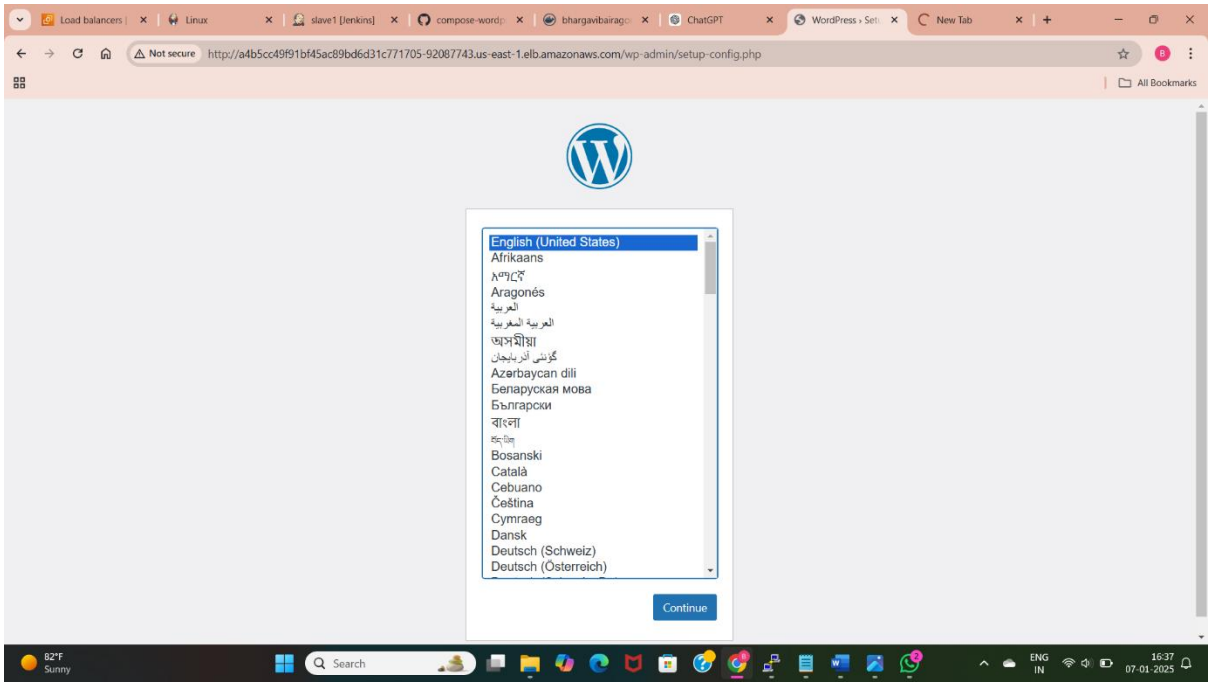
These are the clusters created by the Kubernetes code.



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

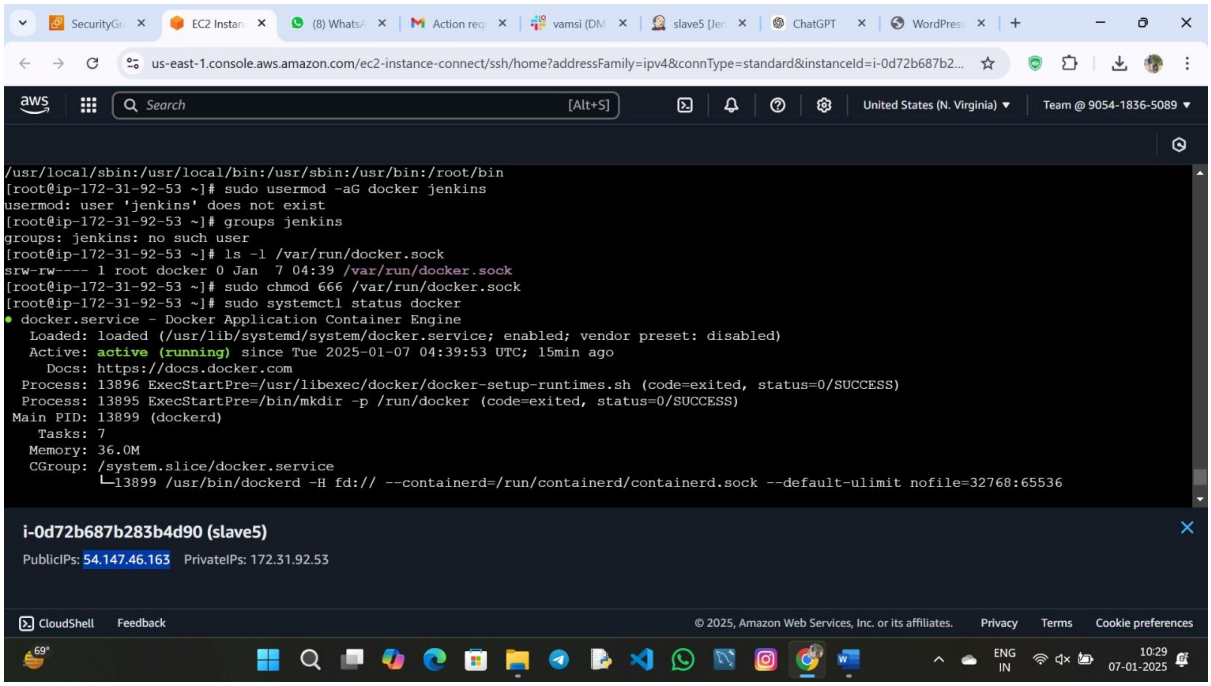


We can also see the external-ip(load balancer urls) in the terminal by checking the service using “kubectl get svc”.

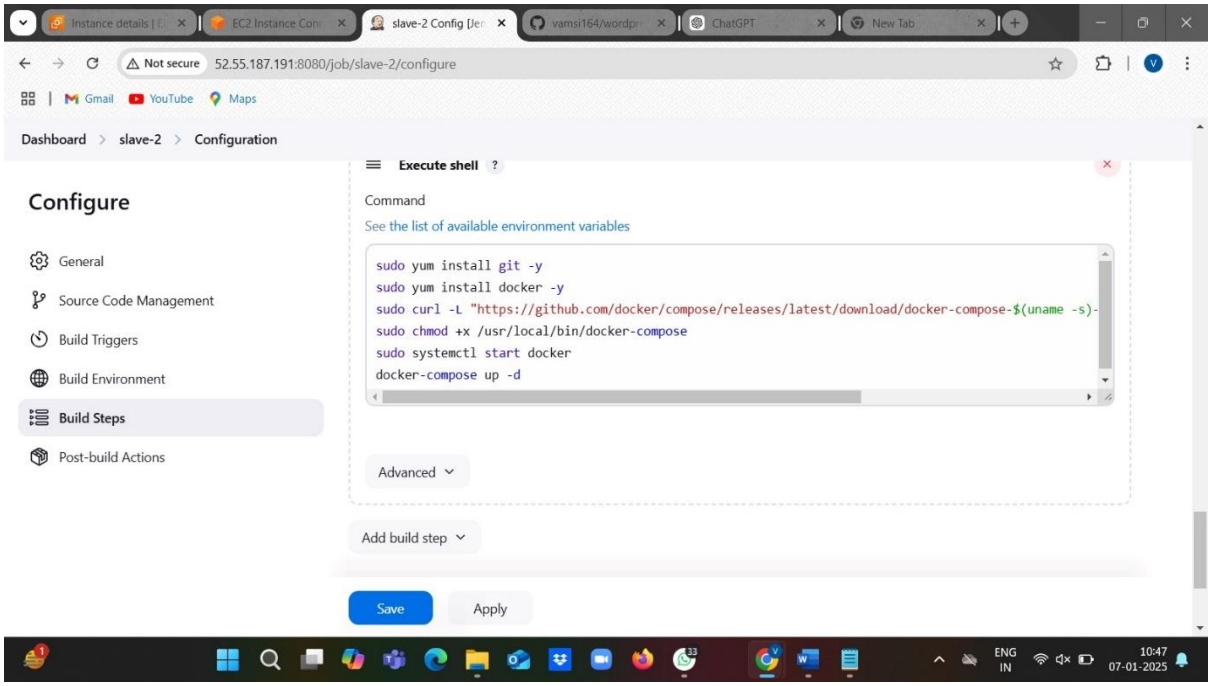


This is the wordpress application deployed using Kubernetes.

SLAVE-2:

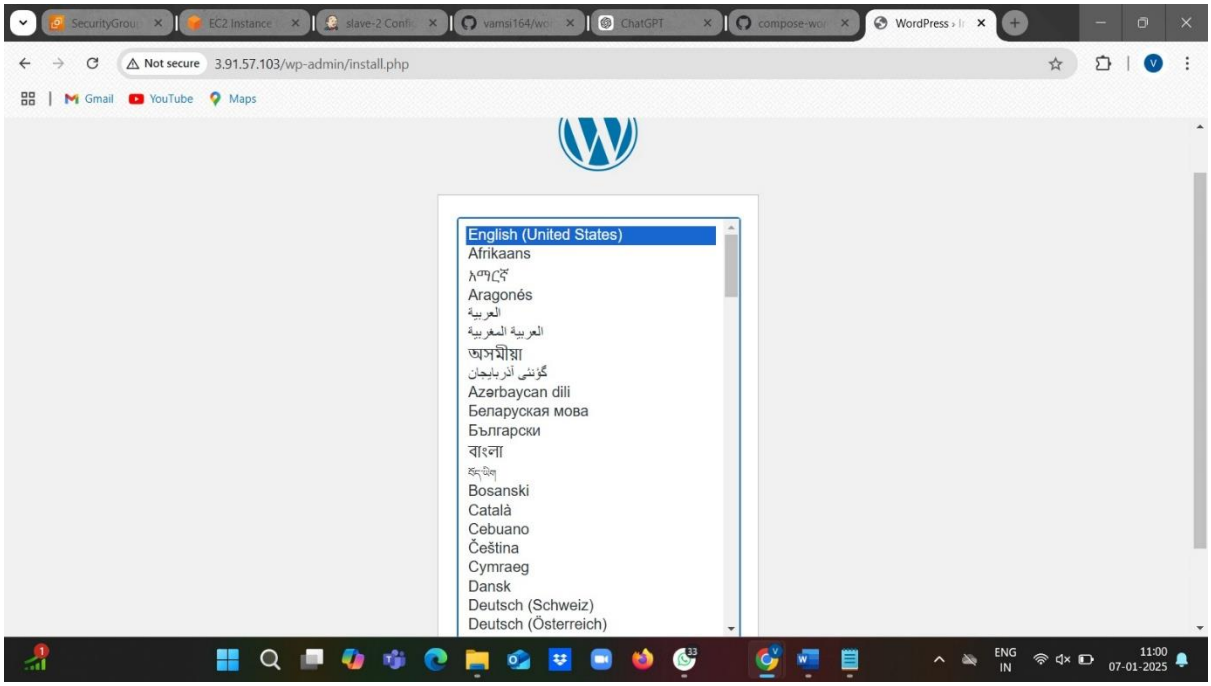


Now the slave2 node is created by Vamsi and he connected the slave with the master and deploying the wordpress application using the docker.



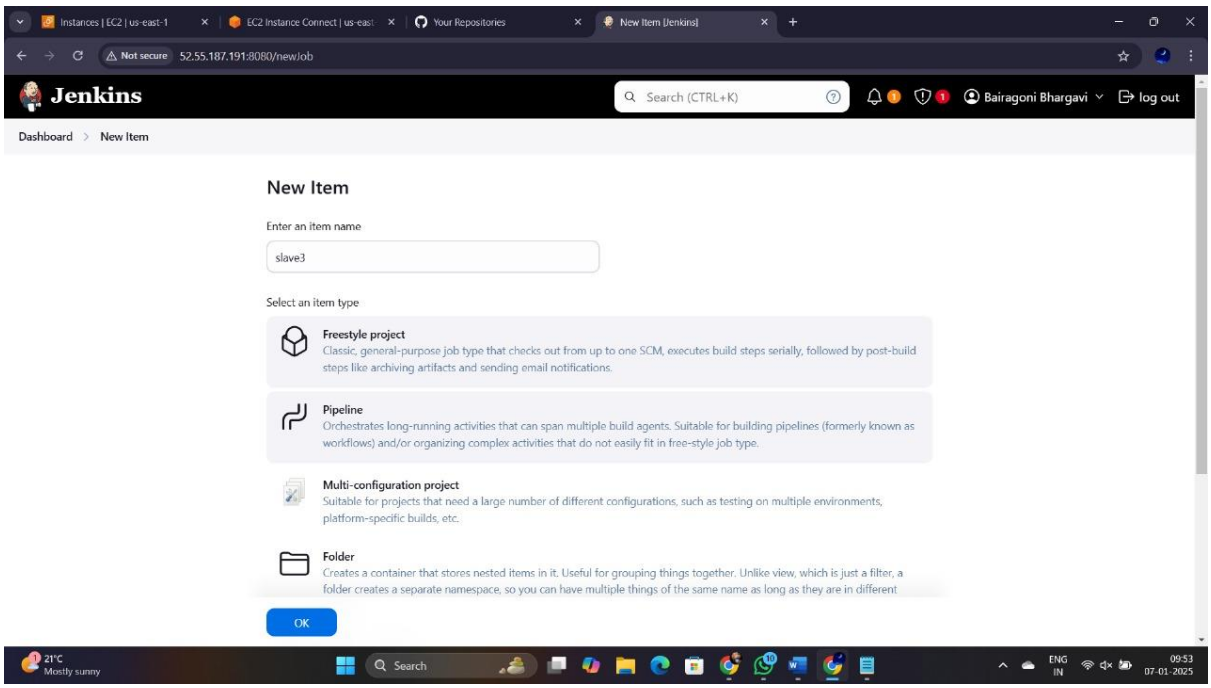
These are the commands used to install git, docker, docker compose and provided the required permissions and started the docker.

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

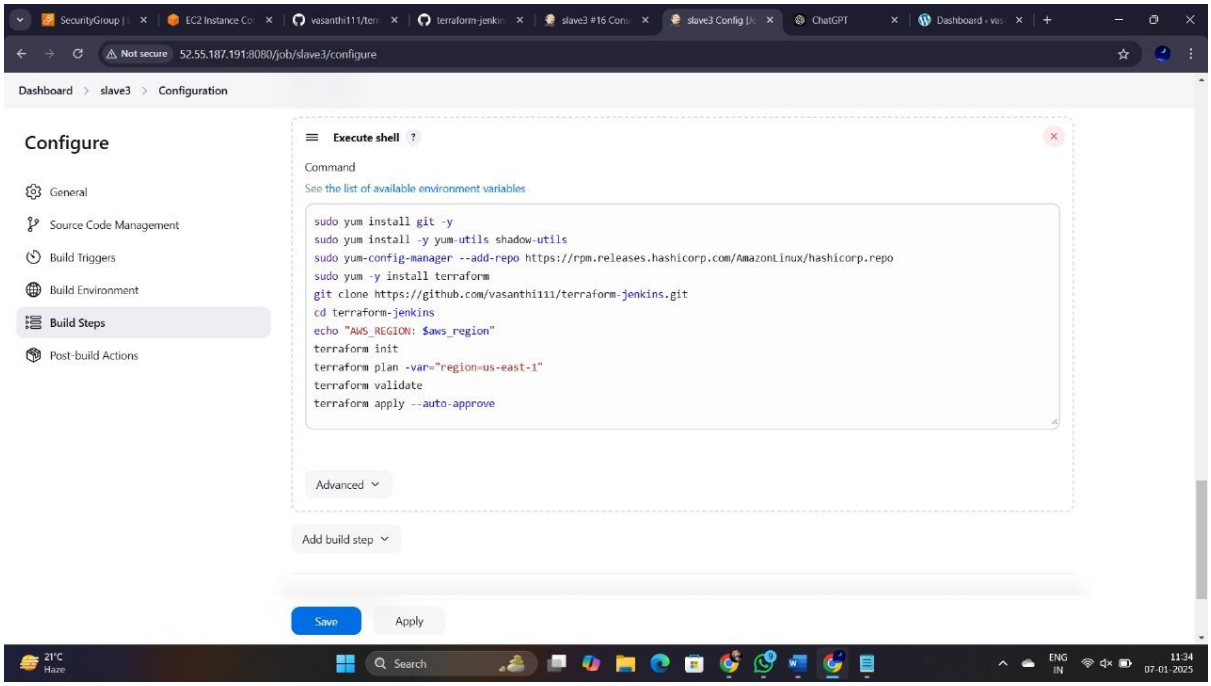


Now he had taken the public IP address of the slave and pasted it on the browser to check whether the app deployed successfully or not. It is deployed and we can able to see the application.

SLAVE-3:

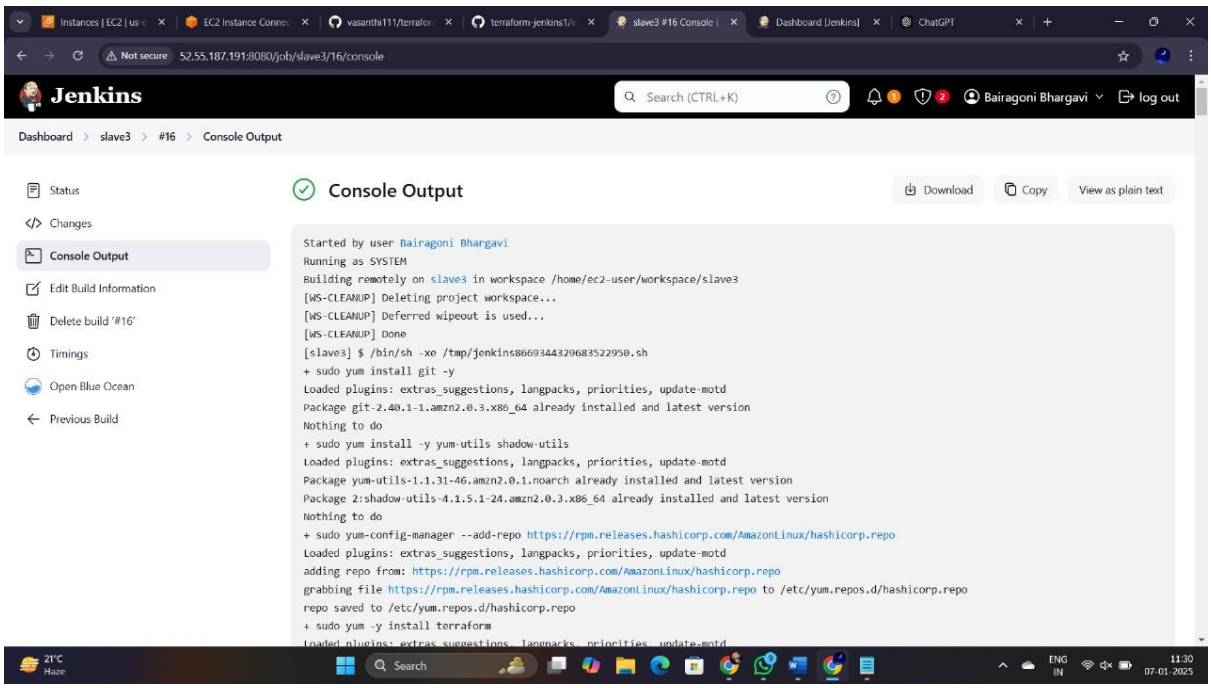


Vasanthi created a slave3 job in Jenkins to deploy the wordpress using the terraform with manually and automatically.



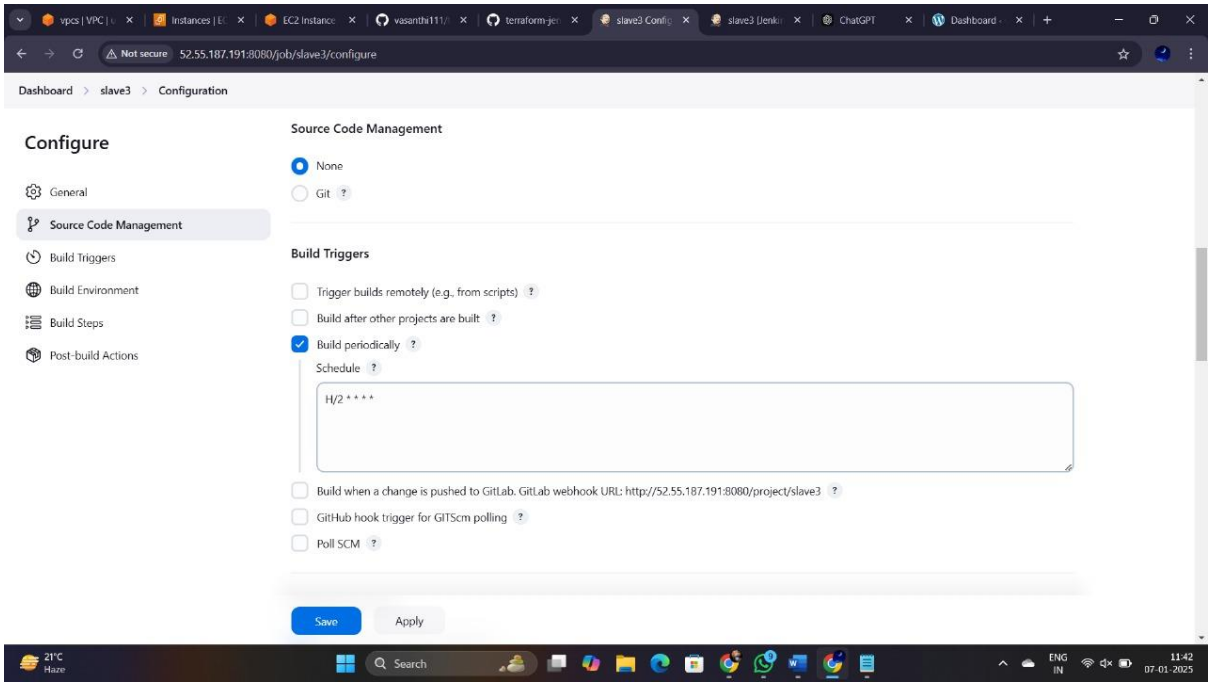
These are the commands she had used in the execute shell to create the terraform files and deploy the application.

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

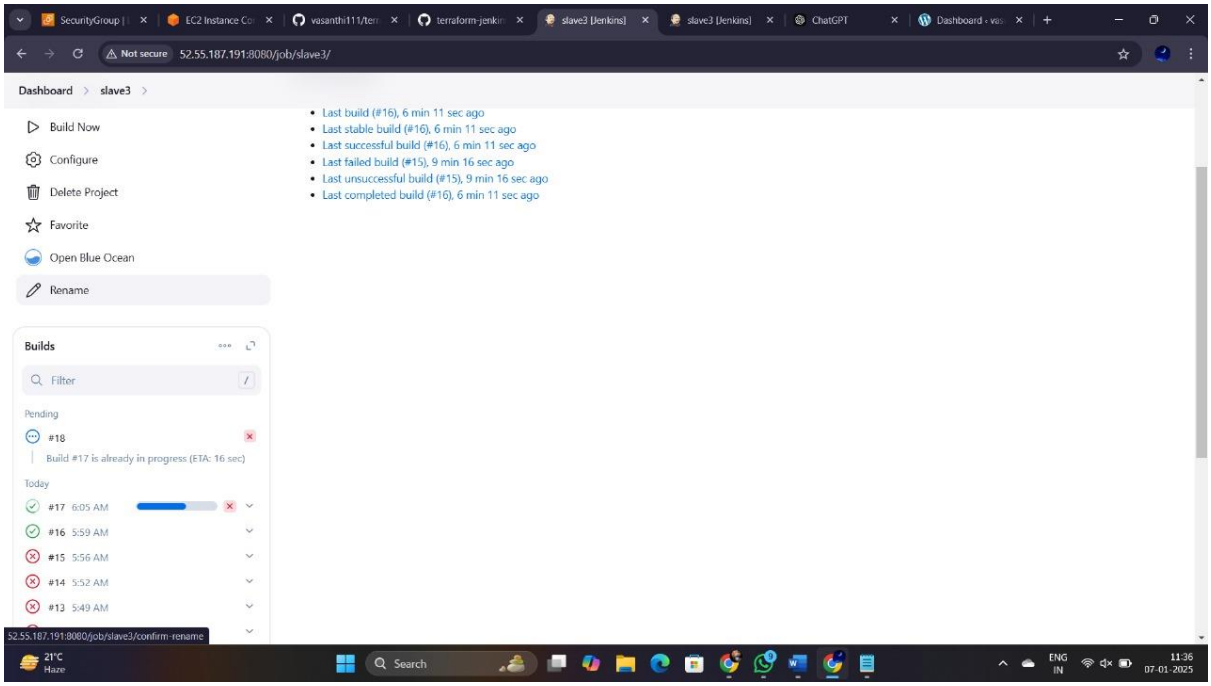


Now she had written commands in execute shell and executed the bulid. The build is success and we can see the step by step process in the console output.

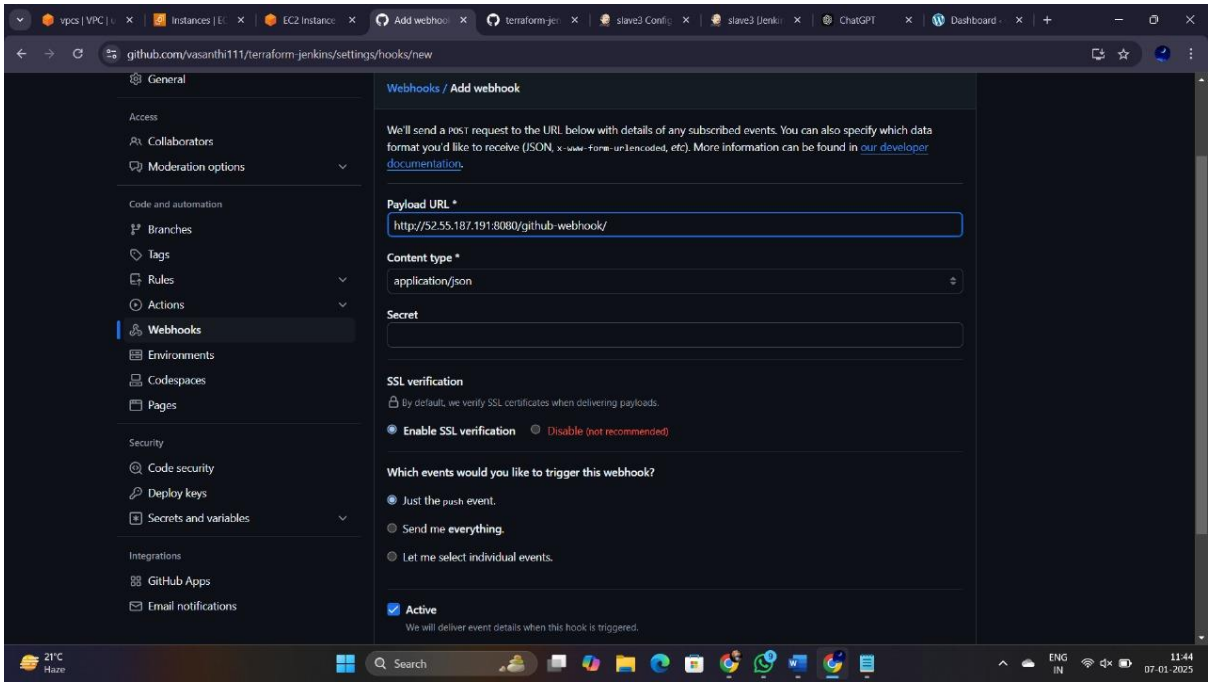
Slave-4:



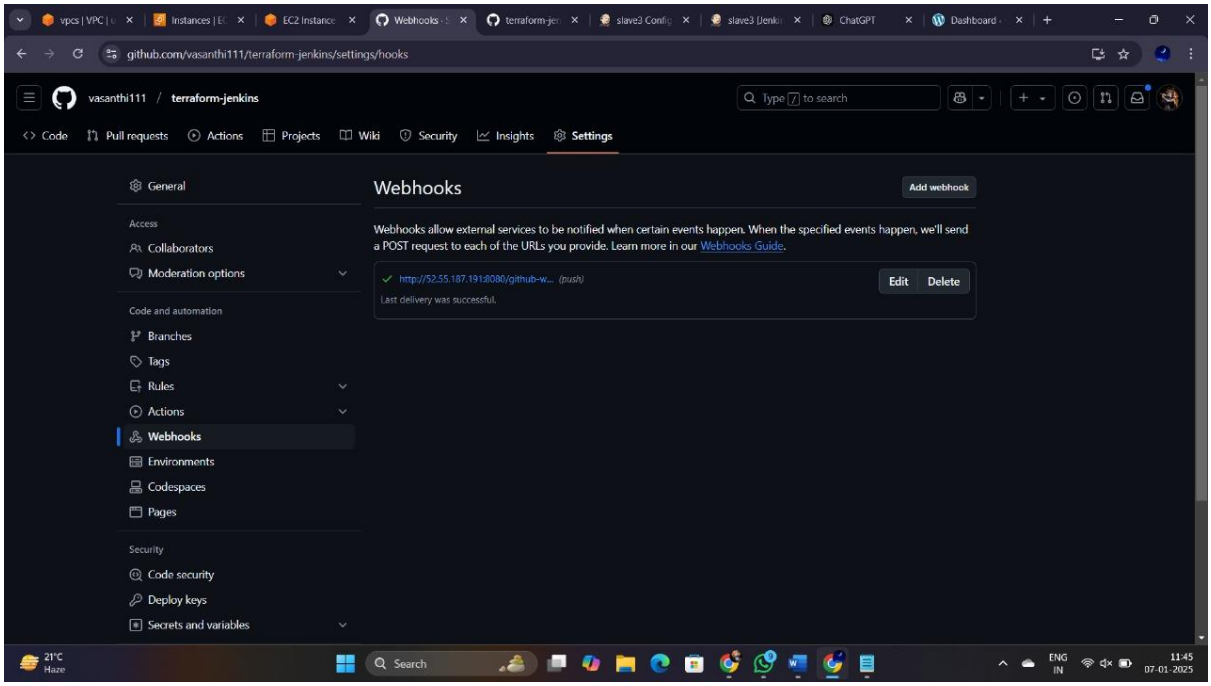
To automate the builds first she choosed the build periodically option with the syntax to deploy the application using terraform files.



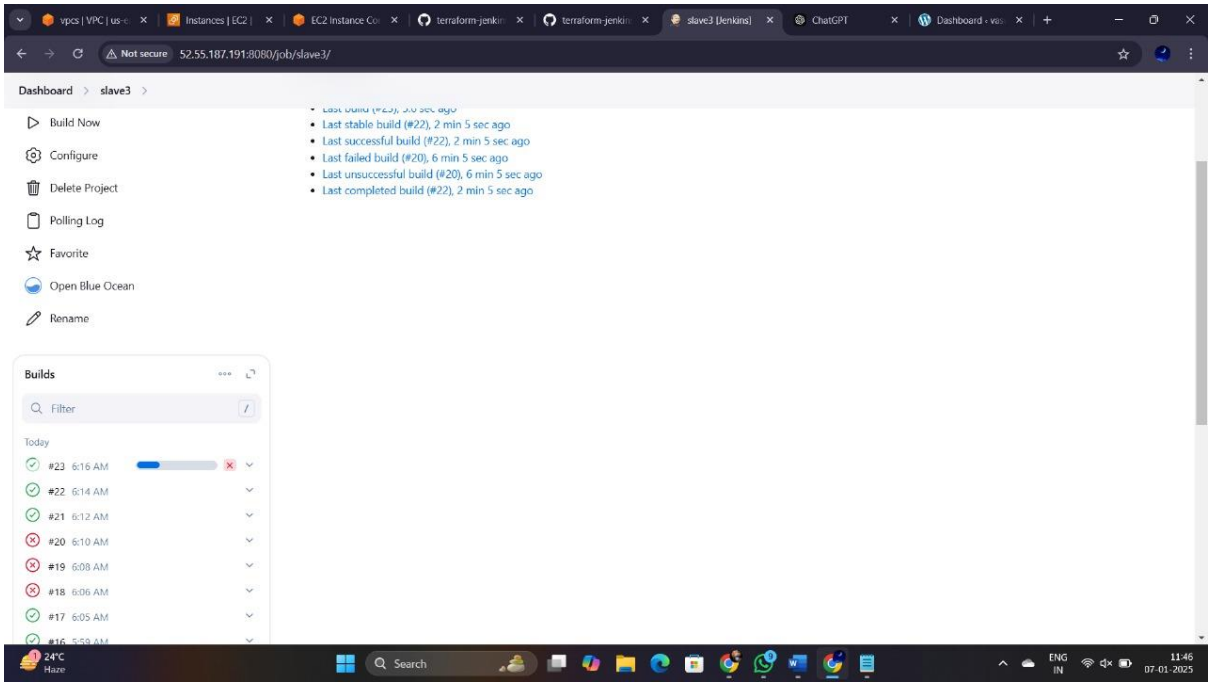
The builds are started automatically when she saved the above action. Now we can see builds are running related to provided the syntax.



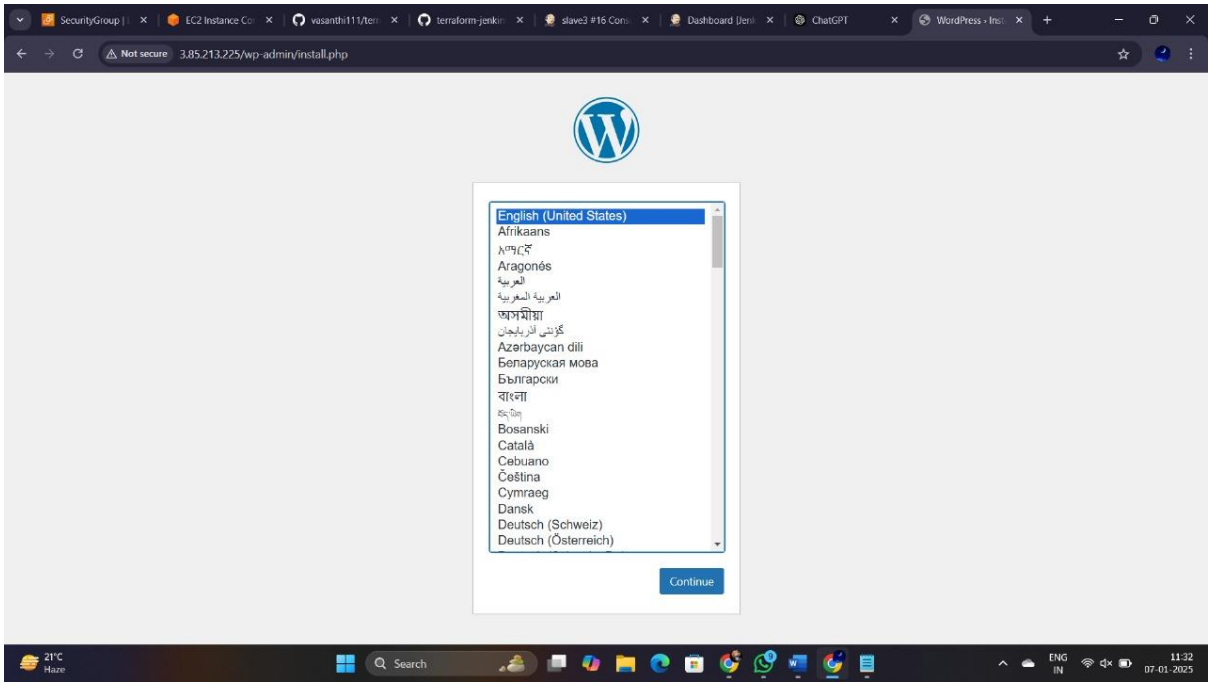
Creating Webhooks in the github where she is having the terraform files. She had provided the Jenkins url here to give updates to that project.



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

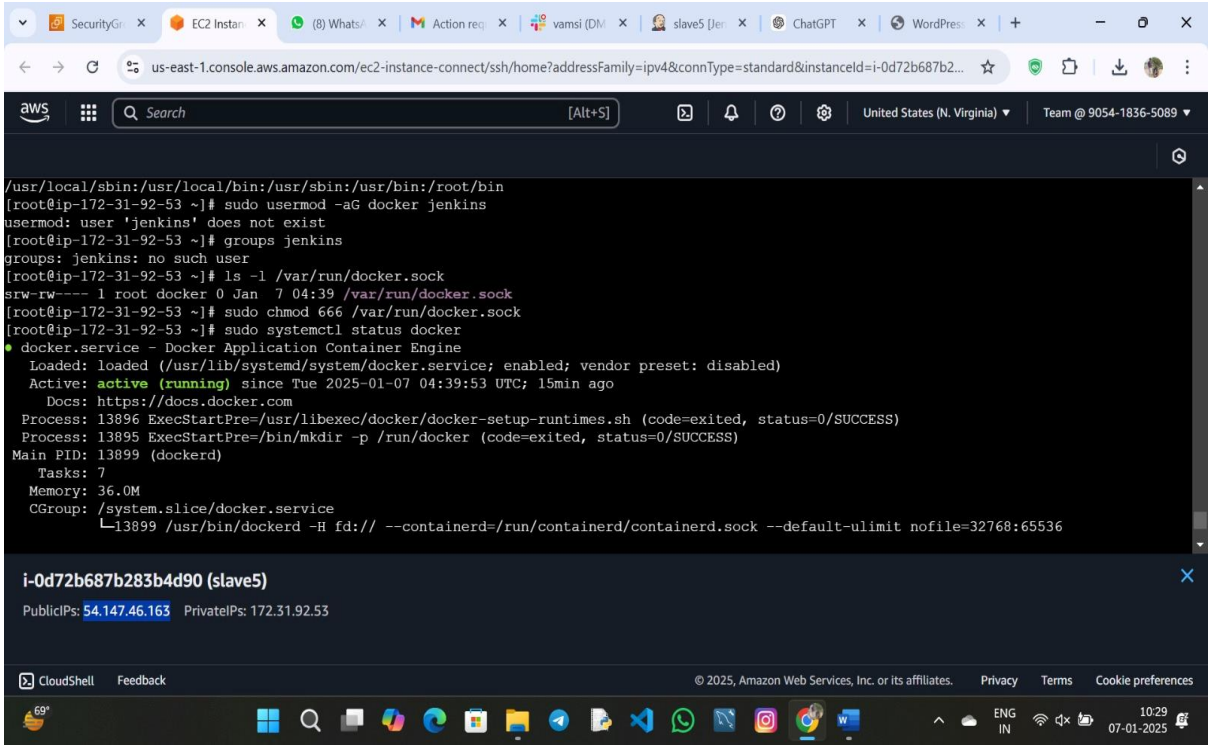


She had written the commands in execute shell and performed the build using the poll scm. We can see the builds are occurring for every two minutes using github webhooks.

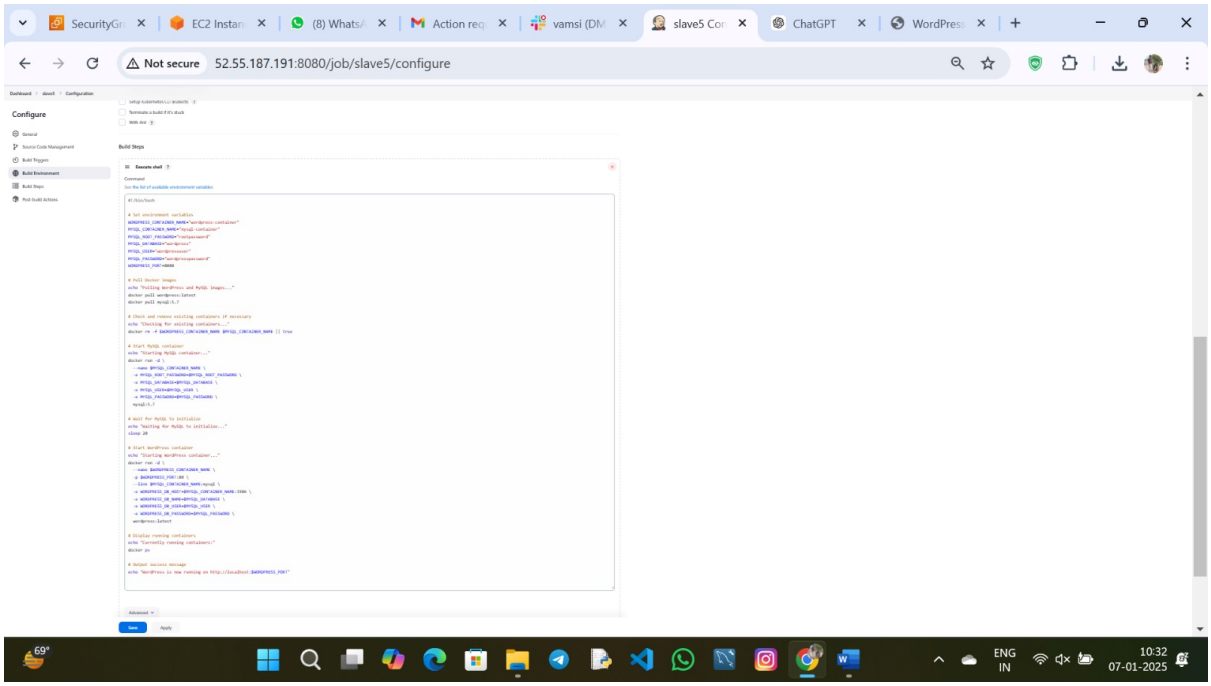


This is the wordpress application she had deployed using the terraform manually and automatically.

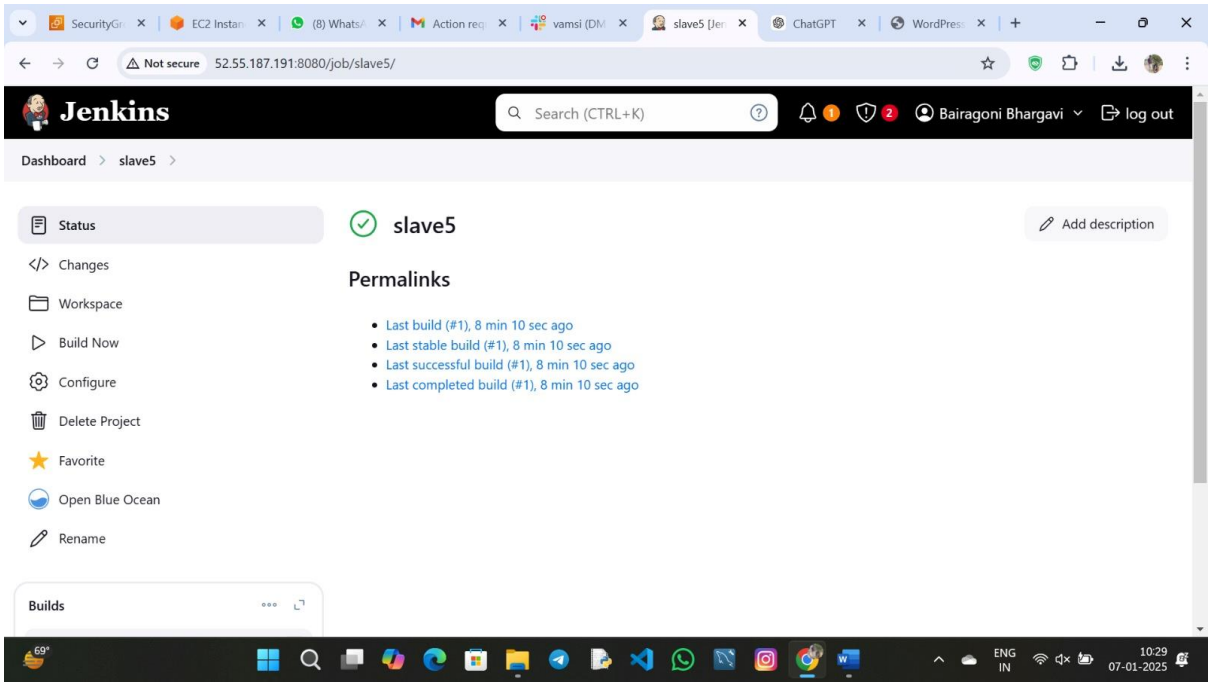
SLAVE-5:



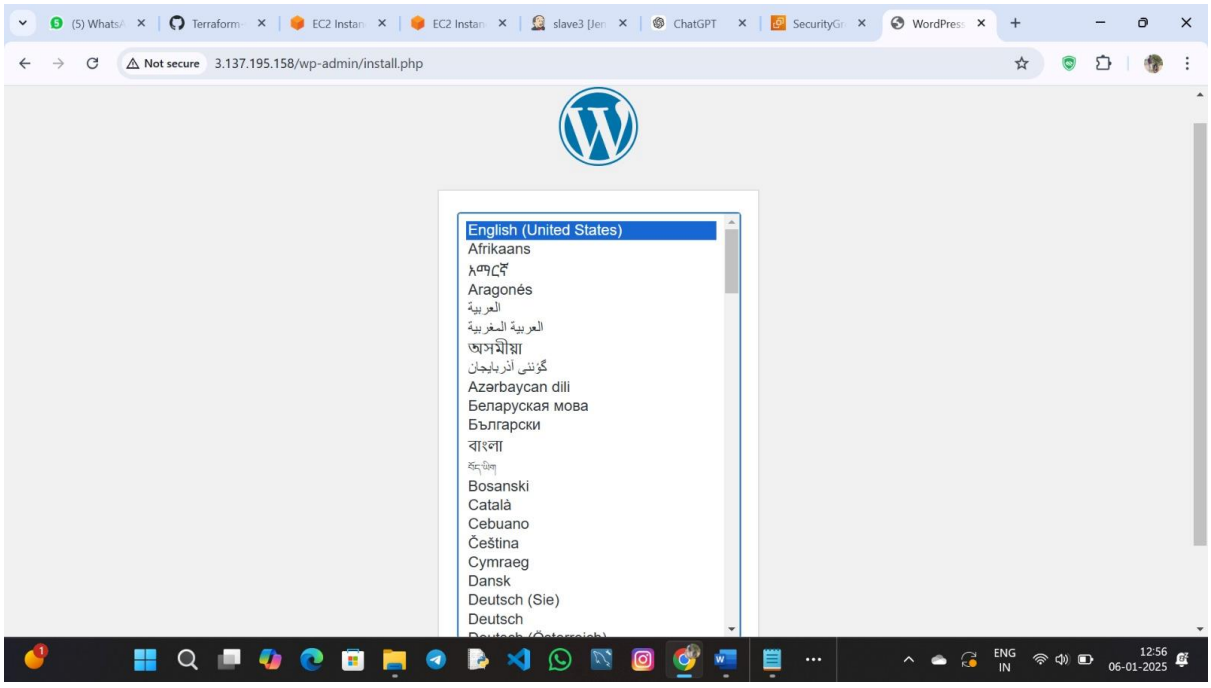
Now Balaji is deploying the wordpress application using docker compose file. For that he installed docker and docker compose. Provided the required permissions.



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”.



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.



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

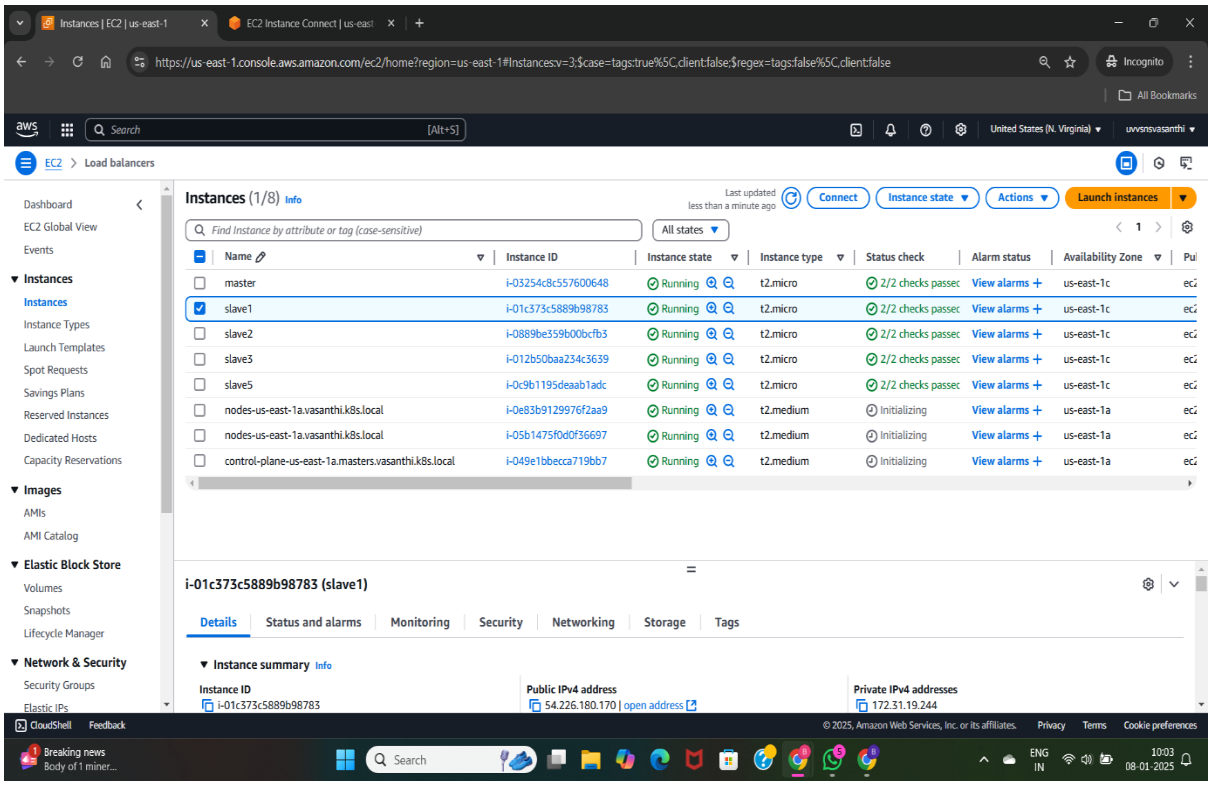
In method 5 we are setting master slave configuration using vasanthi aws account. Here we are creating 5 slaves with one master. Vasanthi is master and she is setting the Jenkins.

Slave1: Deployed wordpress application using k8s by vasanthi

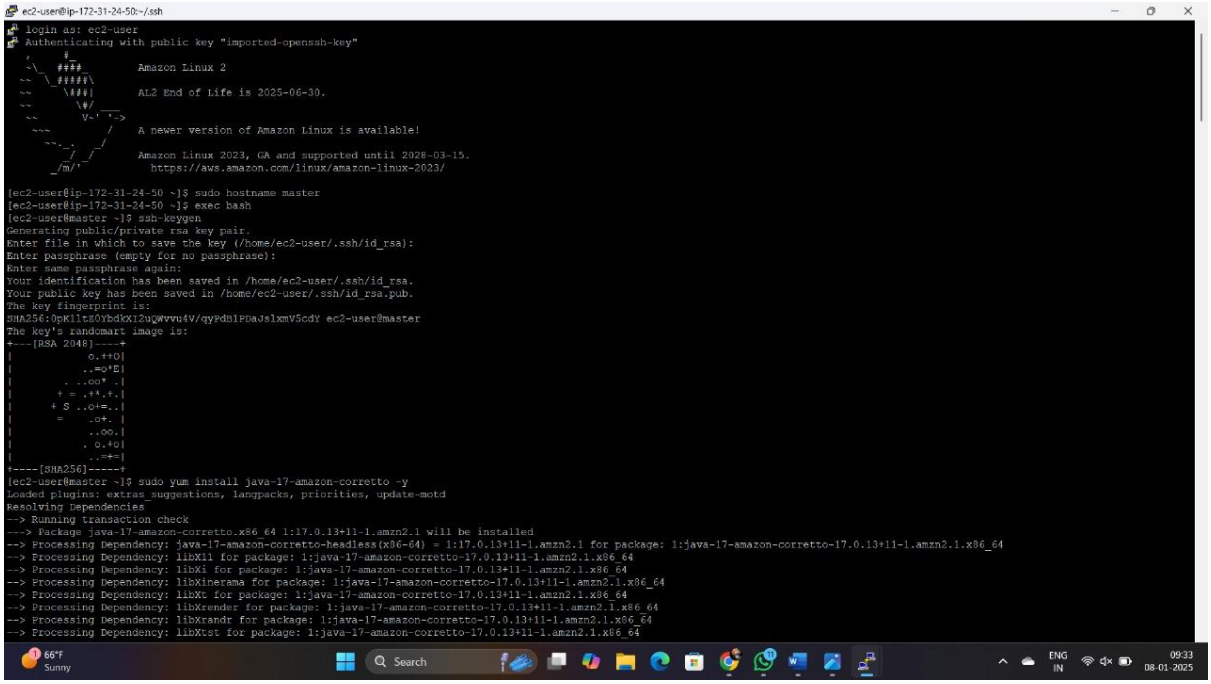
Slave2: Deployed wordpress application using docker-compose by Balaji

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

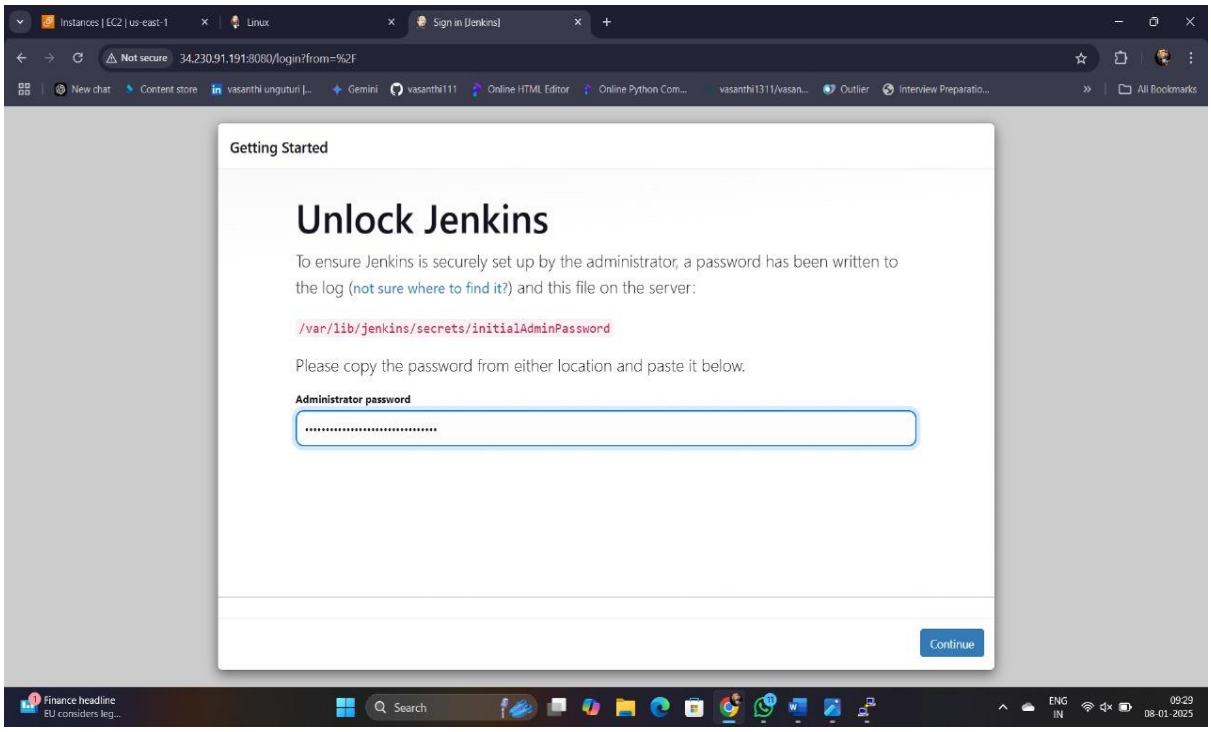
Slave5: Deployed wordpress application using docker compose bash script by vamsi.



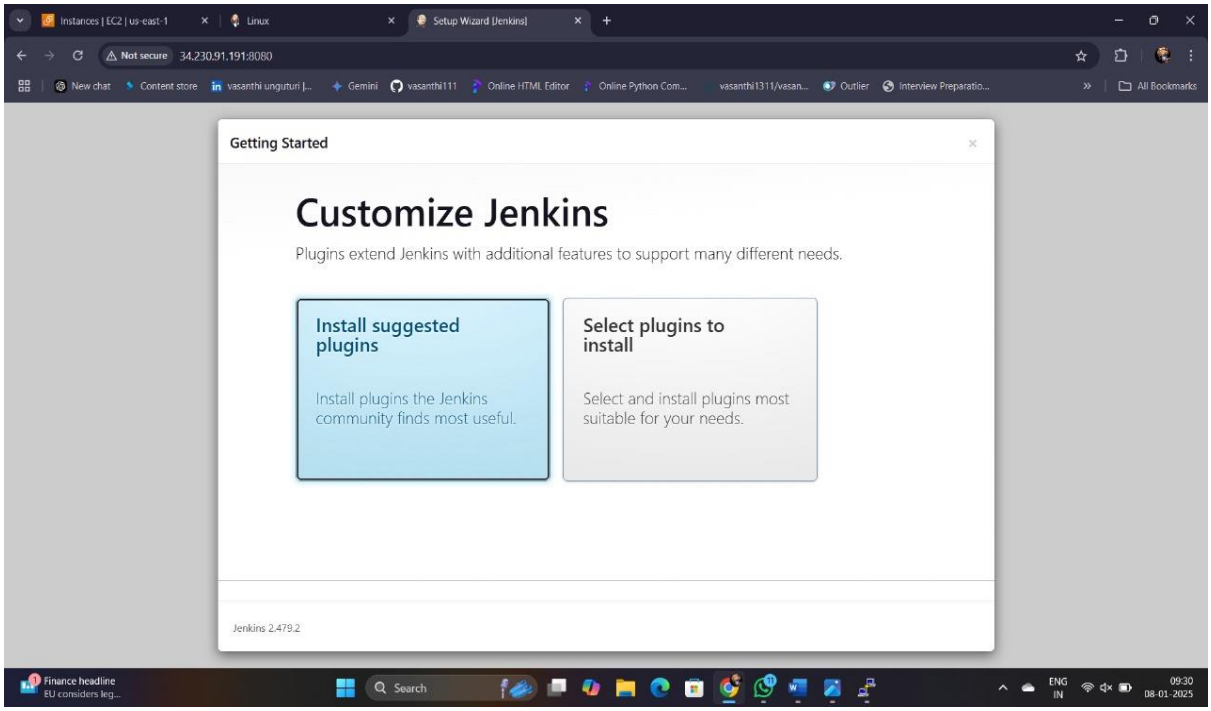
In this account 5 slave servers are created with one master server.



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

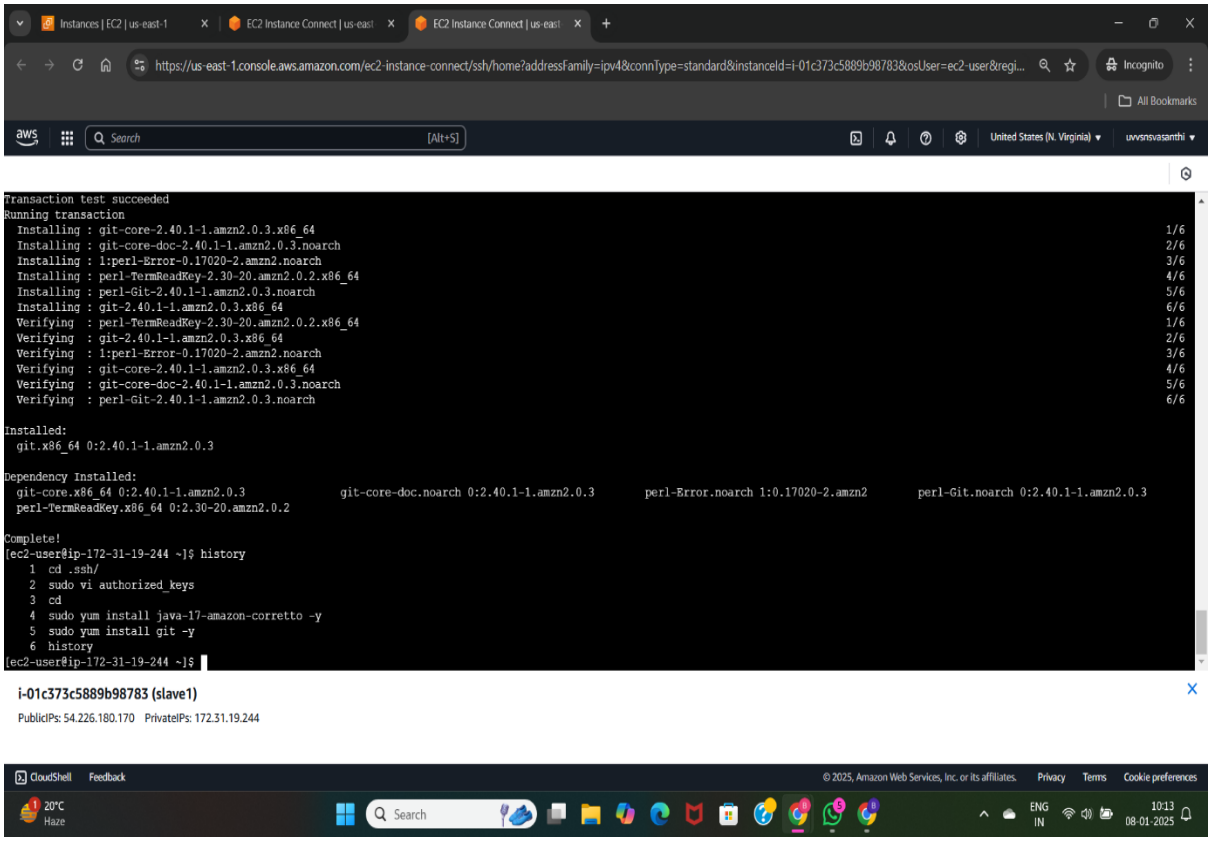


First Jenkins is installed and started by the master server.

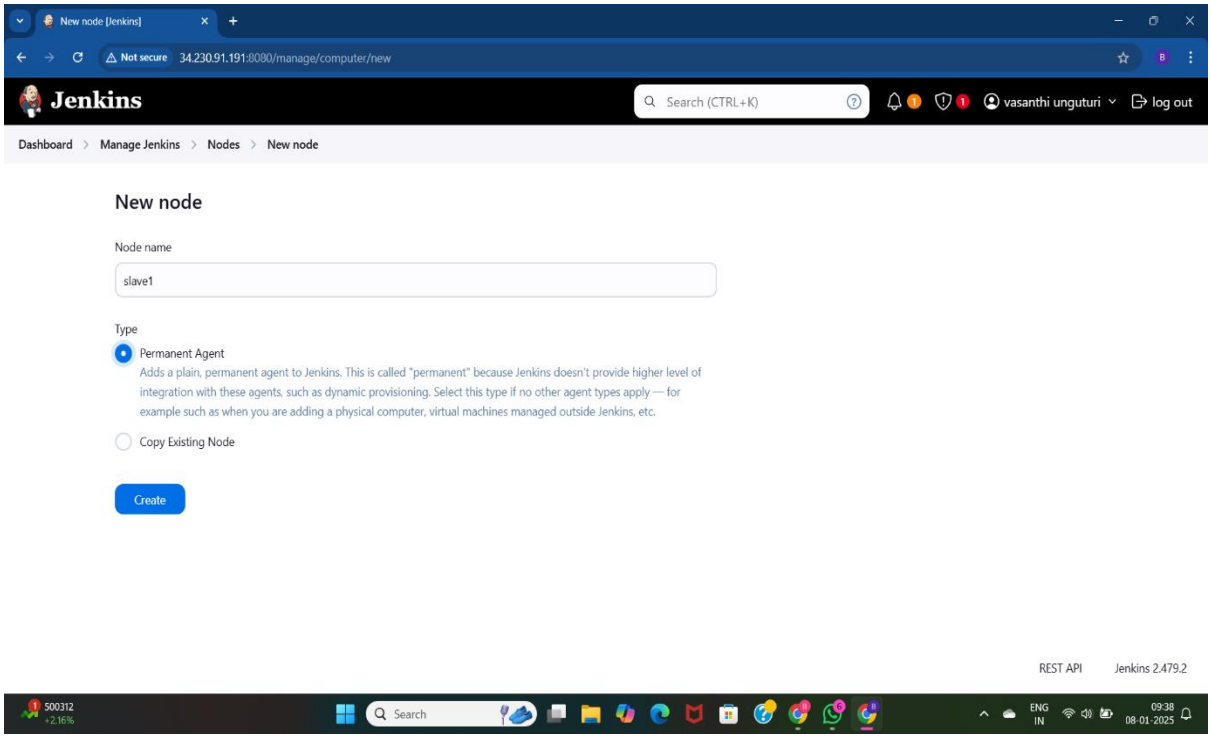


Installed required plugins in Jenkins to create jobs.

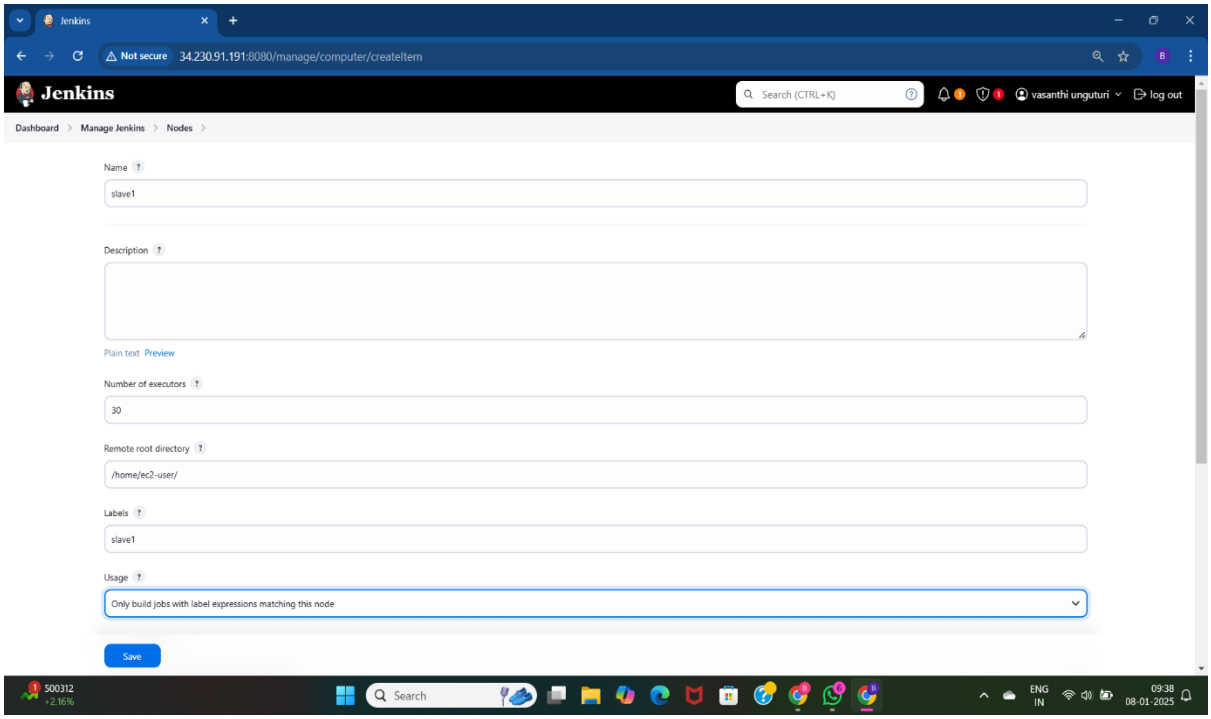
SLAVE-1:



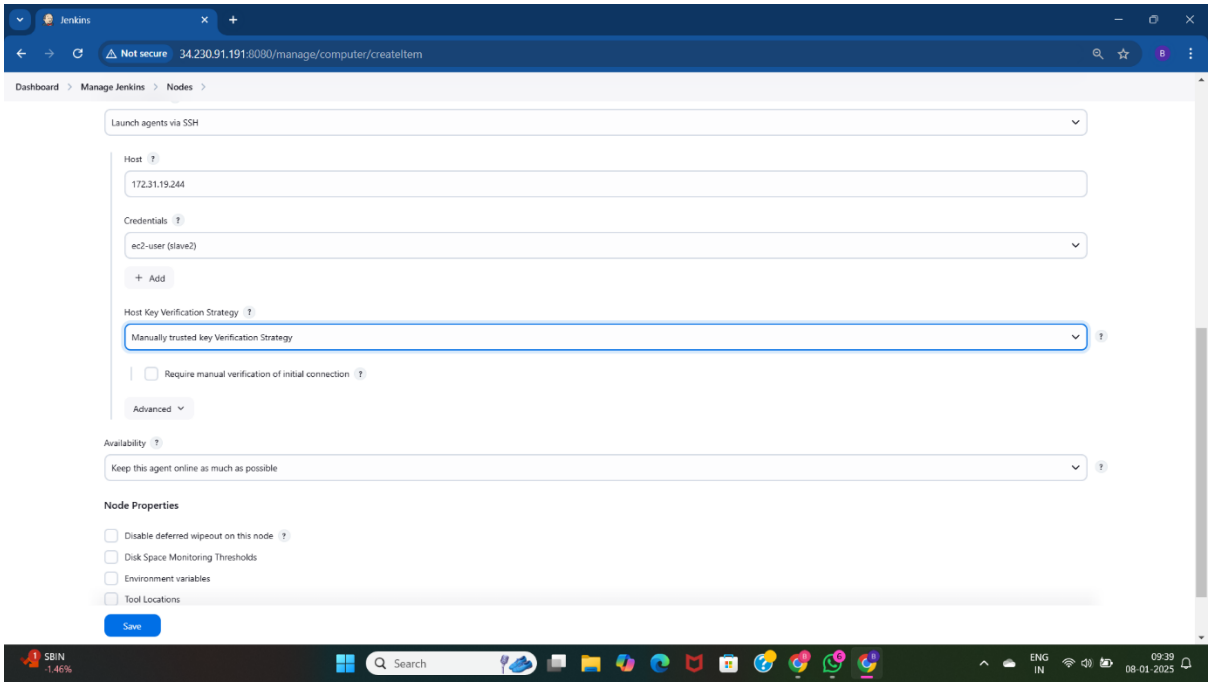
Installed the above mentioned commands here.



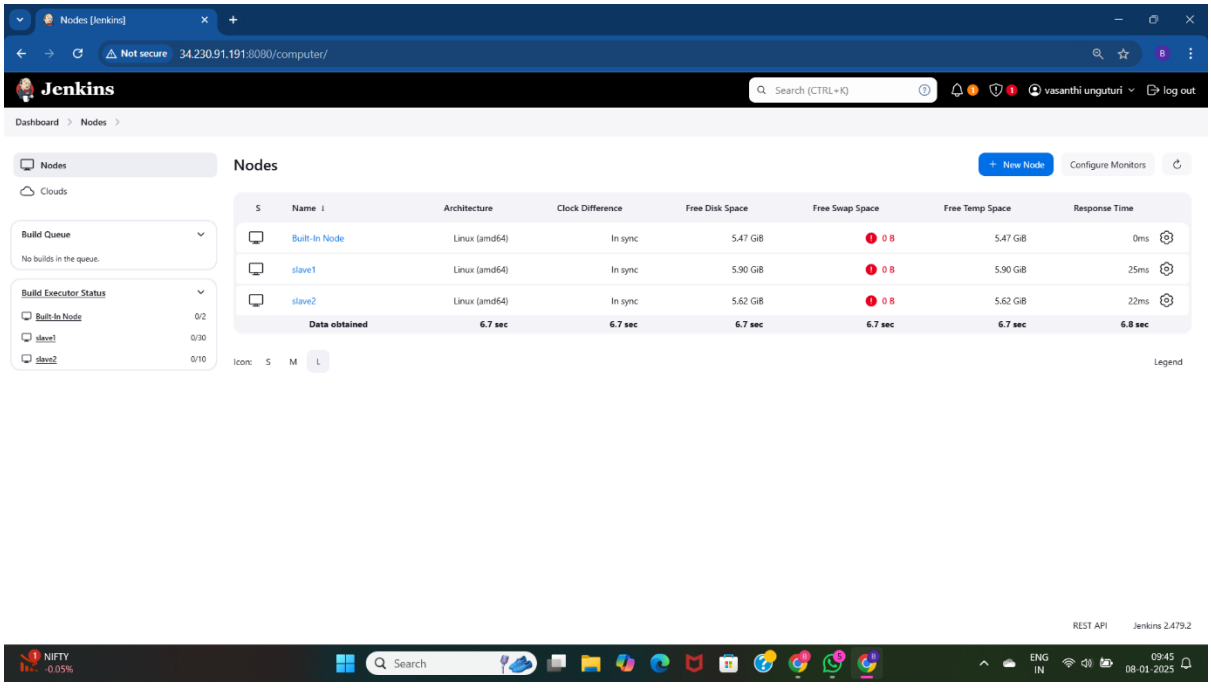
Creating a slave1 node in Jenkins by moving into manage Jenkins> nodes > give name for node amd select permanent agent and click on create.



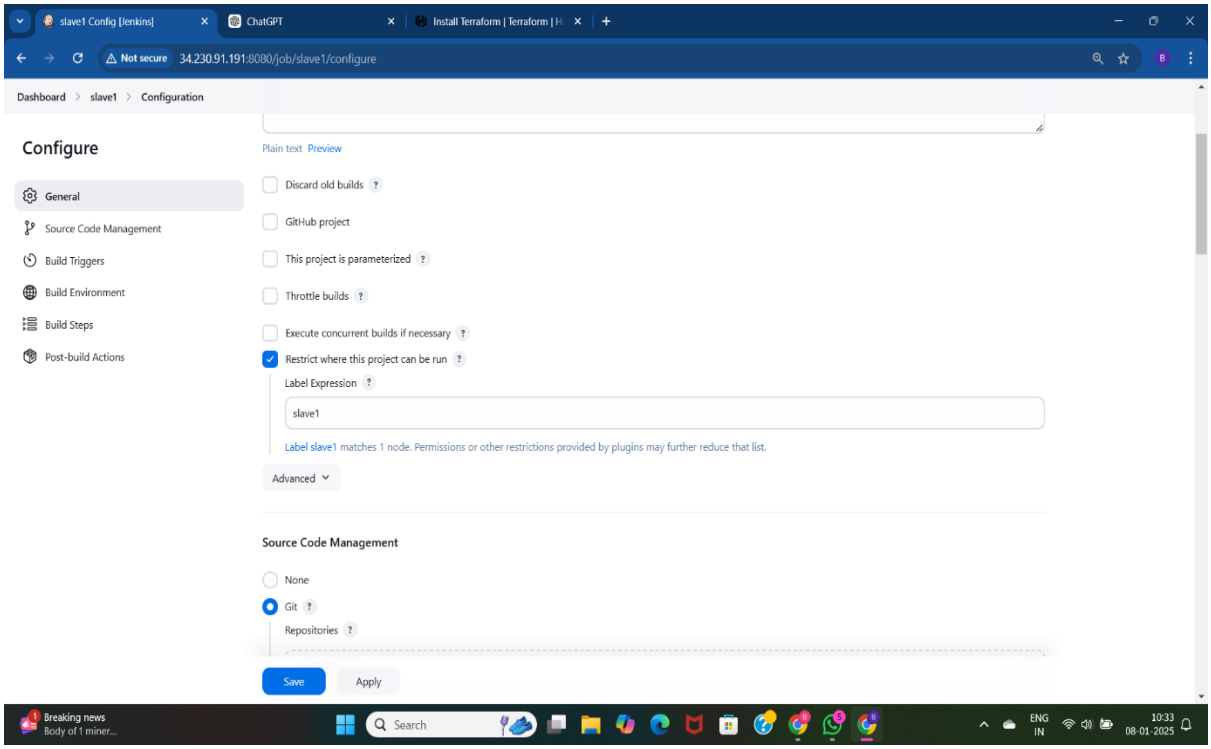
Now give name for the node with number of executors, remote root directory, label name and usage.



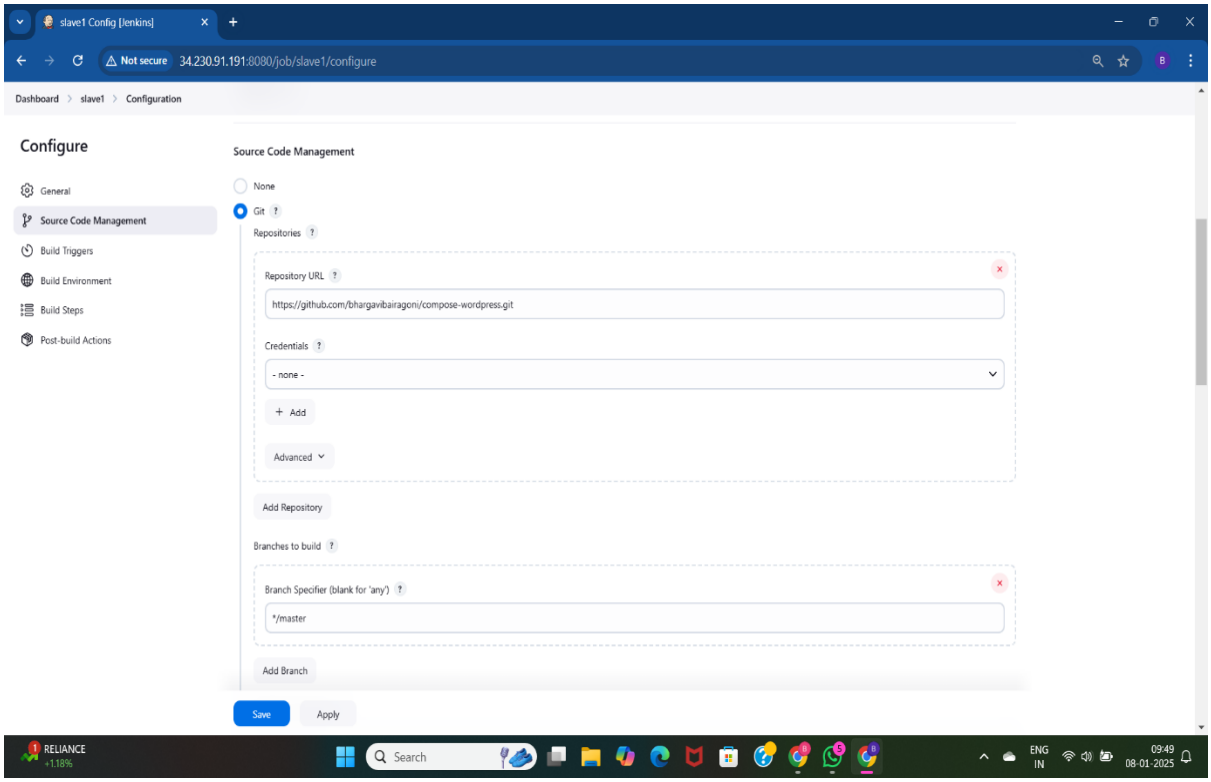
Provide the host as private ip address of the slave server, give credentials to access the slave server and select the key verification strategy.



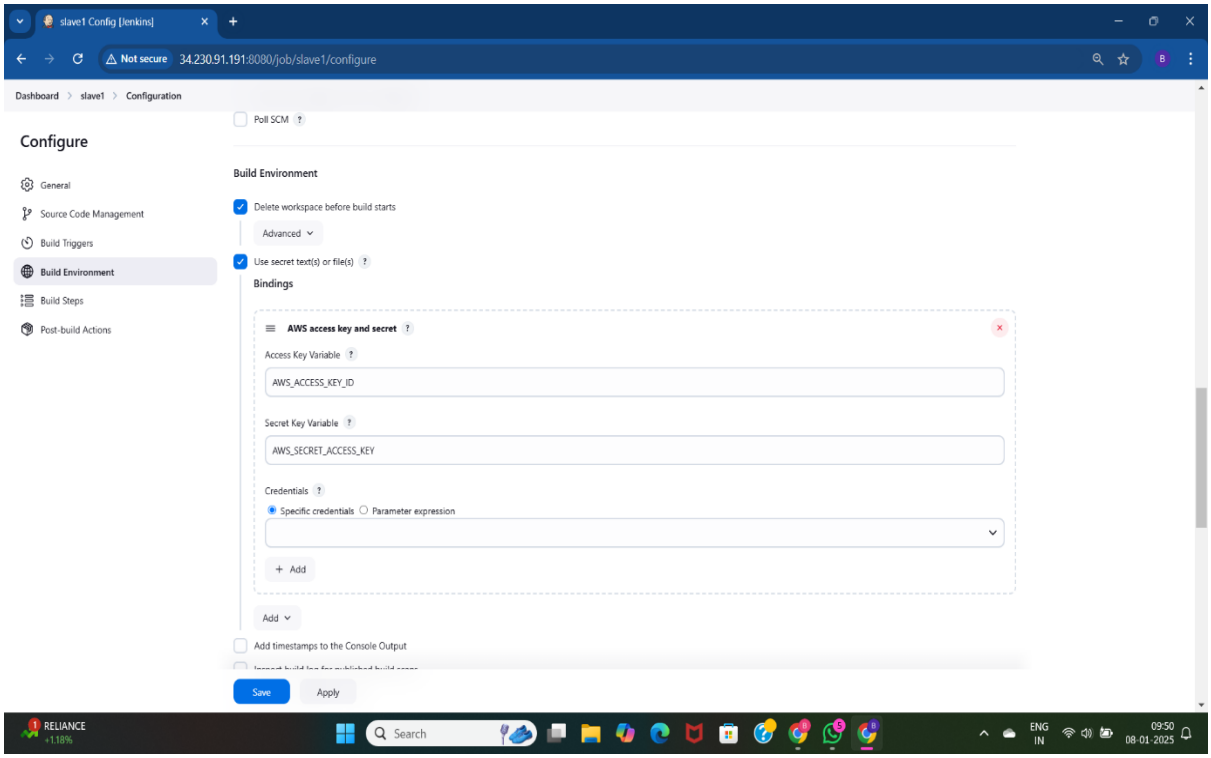
Here we can see two nodes are created.



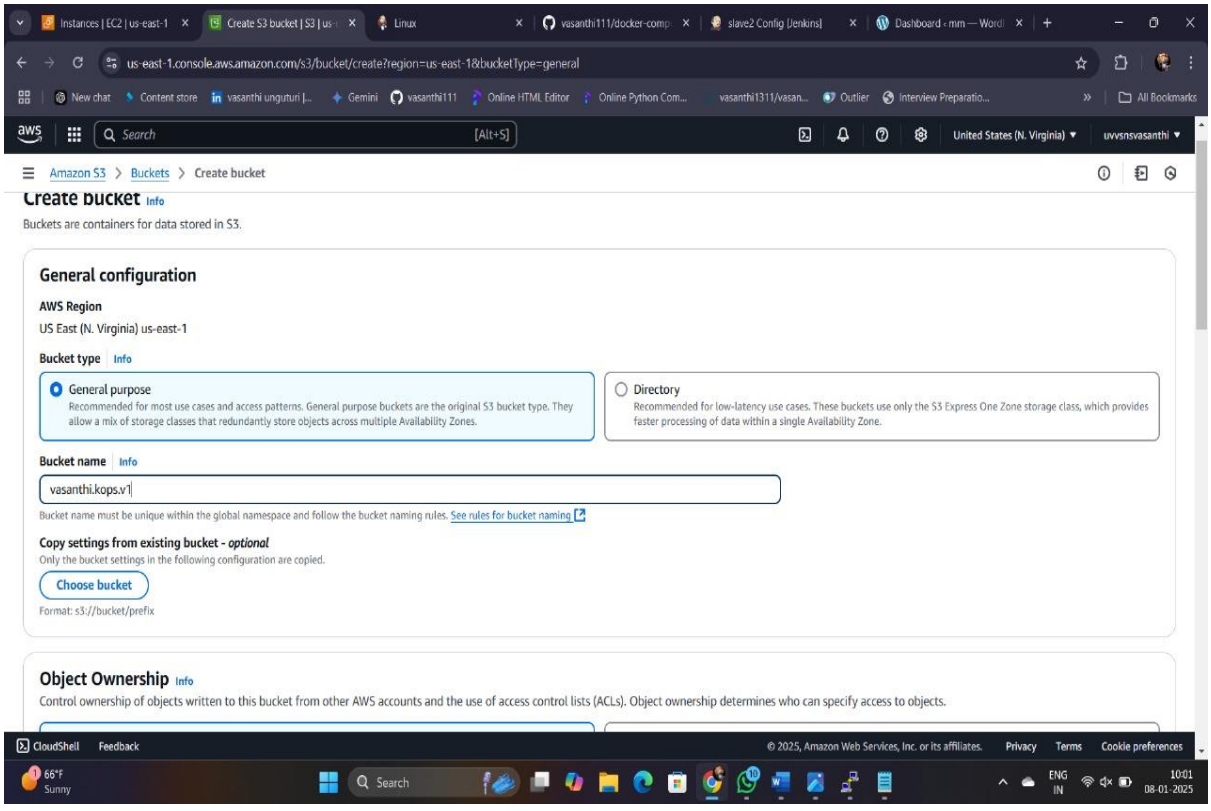
Create a Jenkins free style project, select the created label to execute builds.



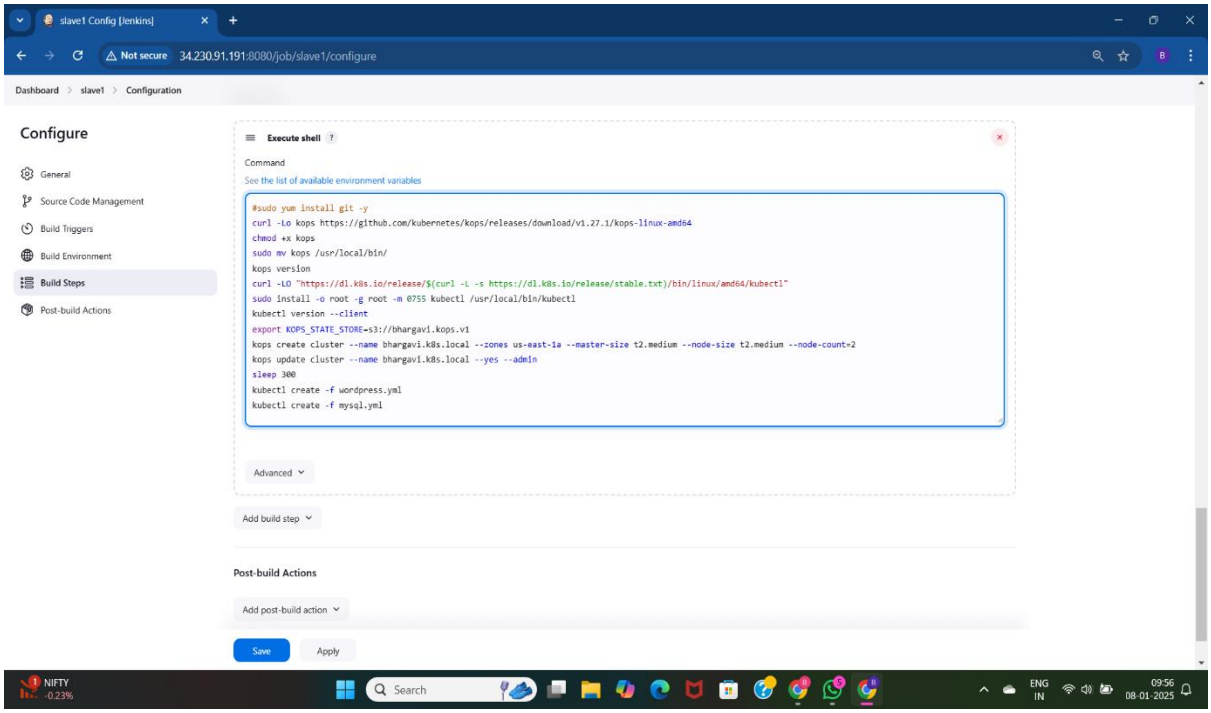
Give the gihub repository url and branch name to access the code and deploy the application.



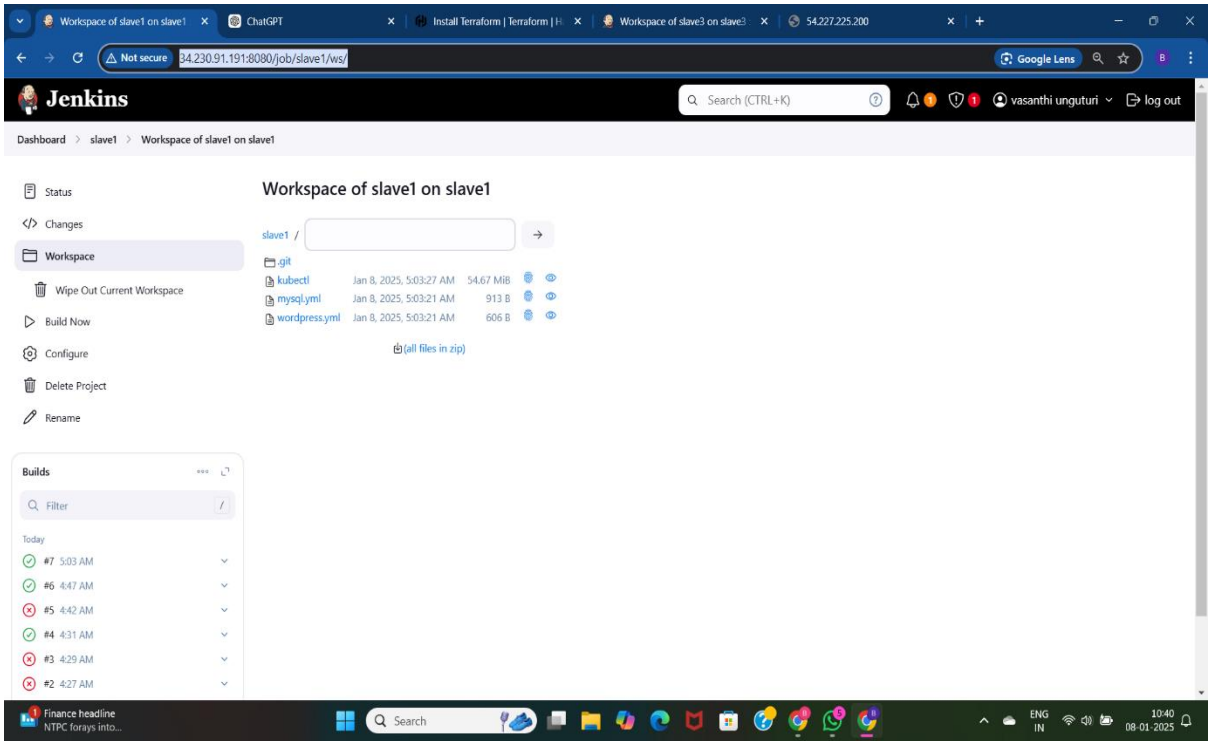
Here give the aws credentials by installing aws credentials plugin. It is used for other slaves also.



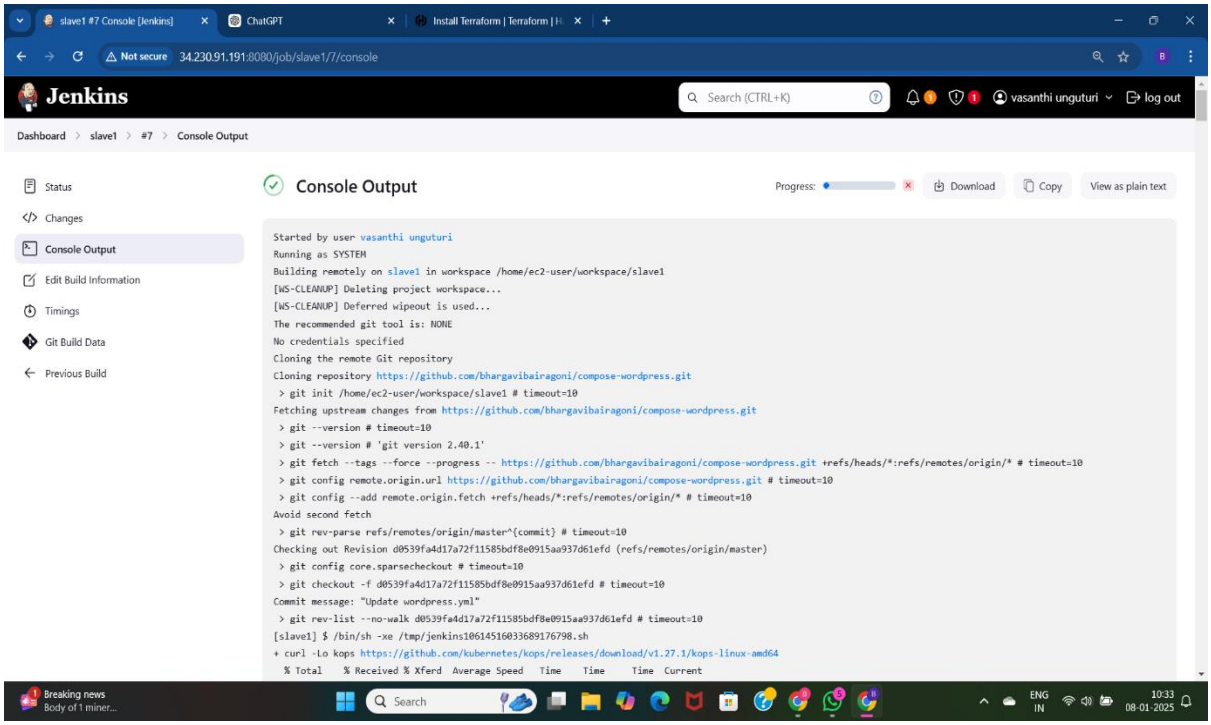
Now create a s3 bucket to deploy the application using k8s.



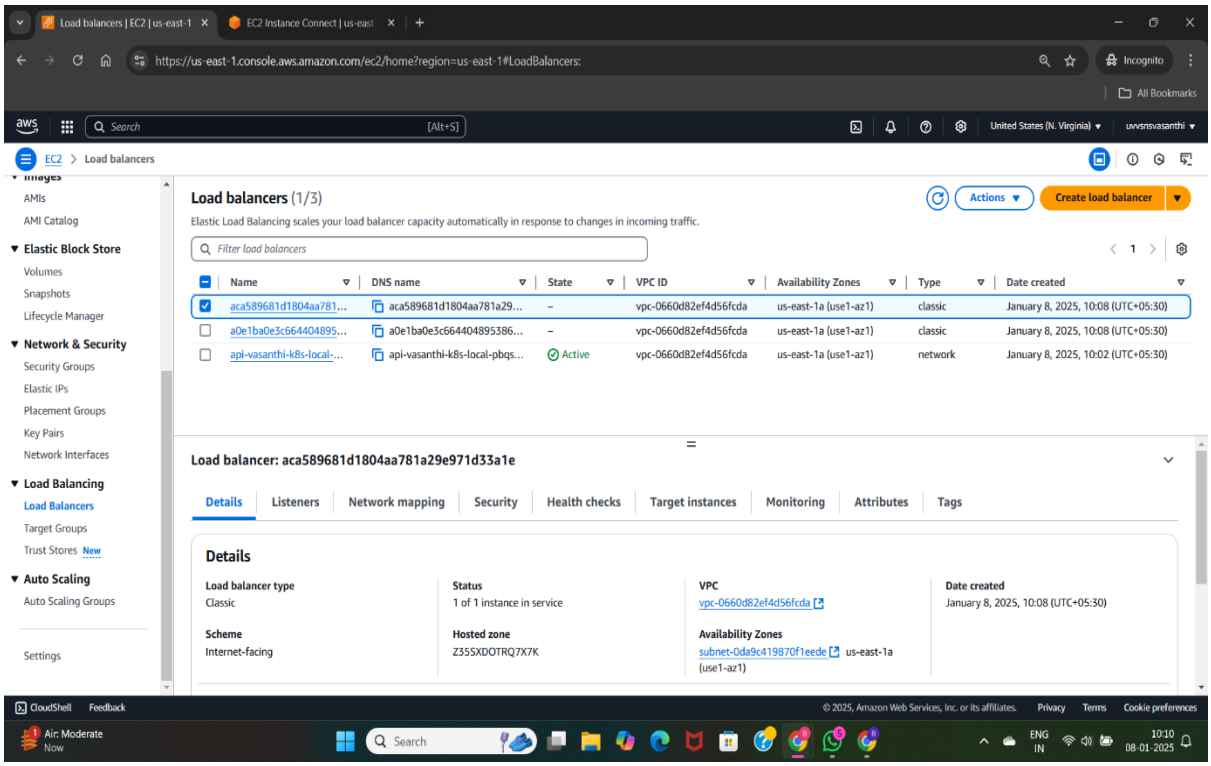
In execute shell write commands to install kops, kubectl, and export the created s3 bucket to store the data. Create the clusters using kops and create the deployment and service files.



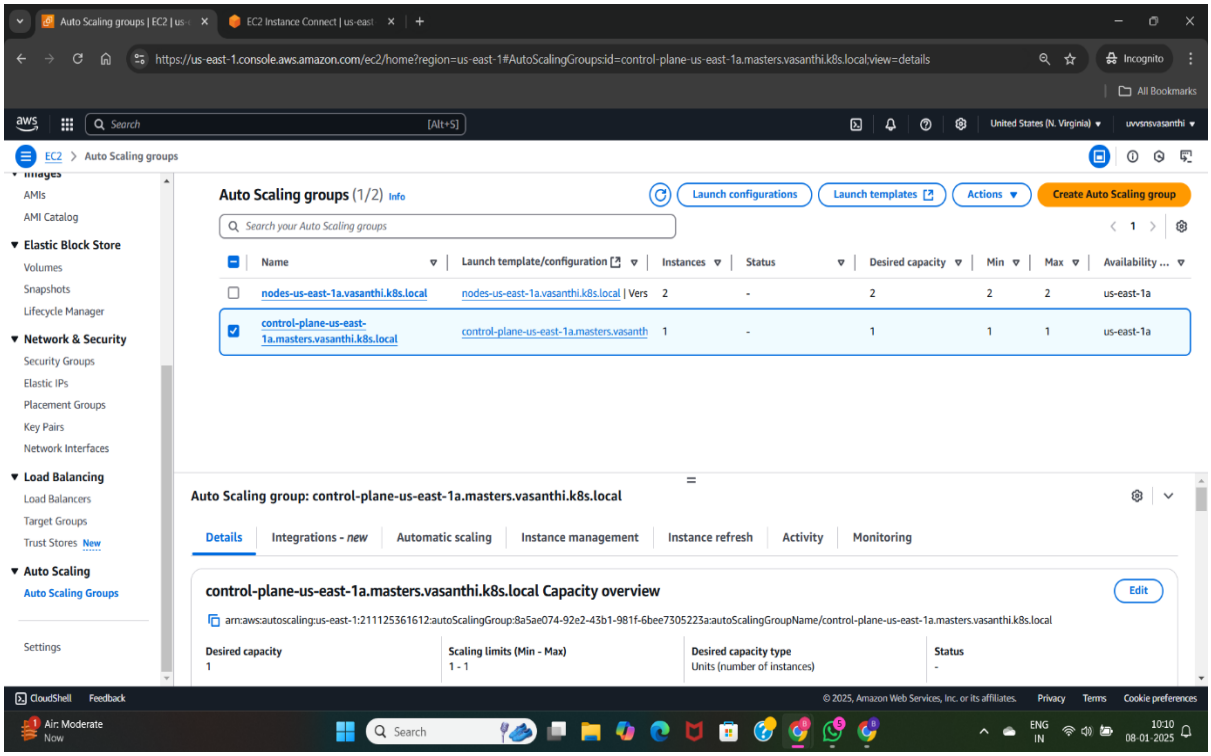
The build is success and we can see the files related to deployment and service of the wordpress application.



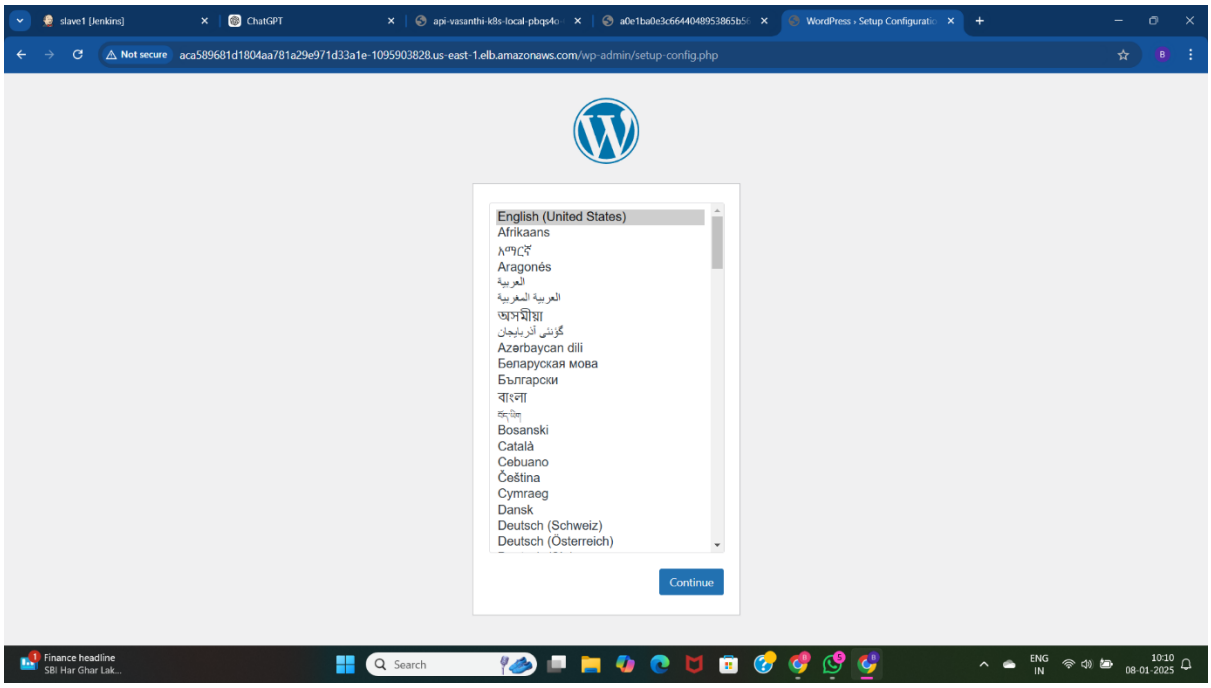
This is the step by step process of the build which is executing every step mentioned in the execute shell.



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

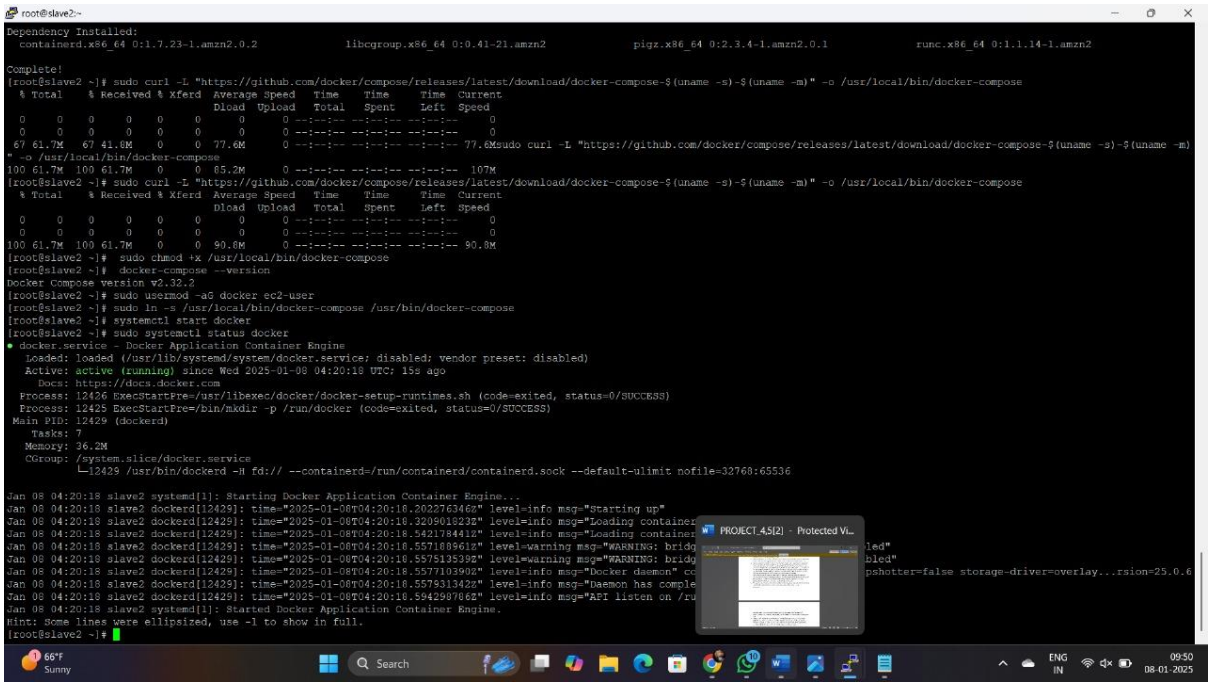


Auto scaling group is also created with the clusters to generate new instances while the load/traffic is increasing.

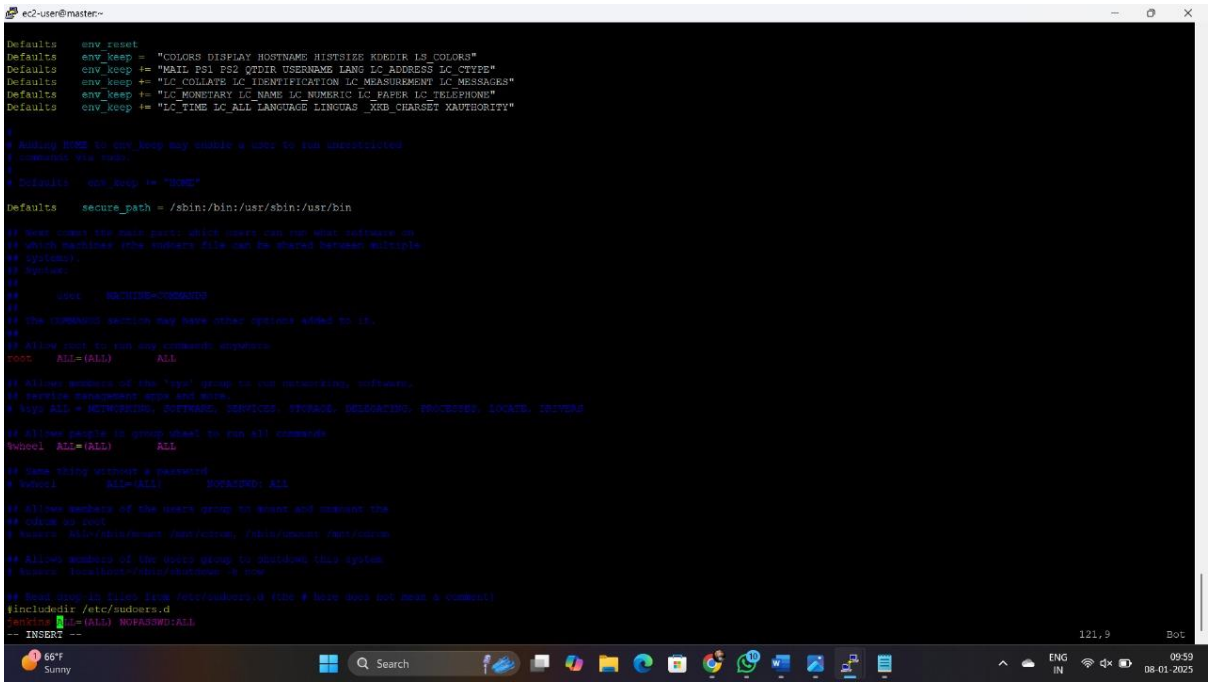


I have taken one of the load balancers DNS and accessed the wordpress application.

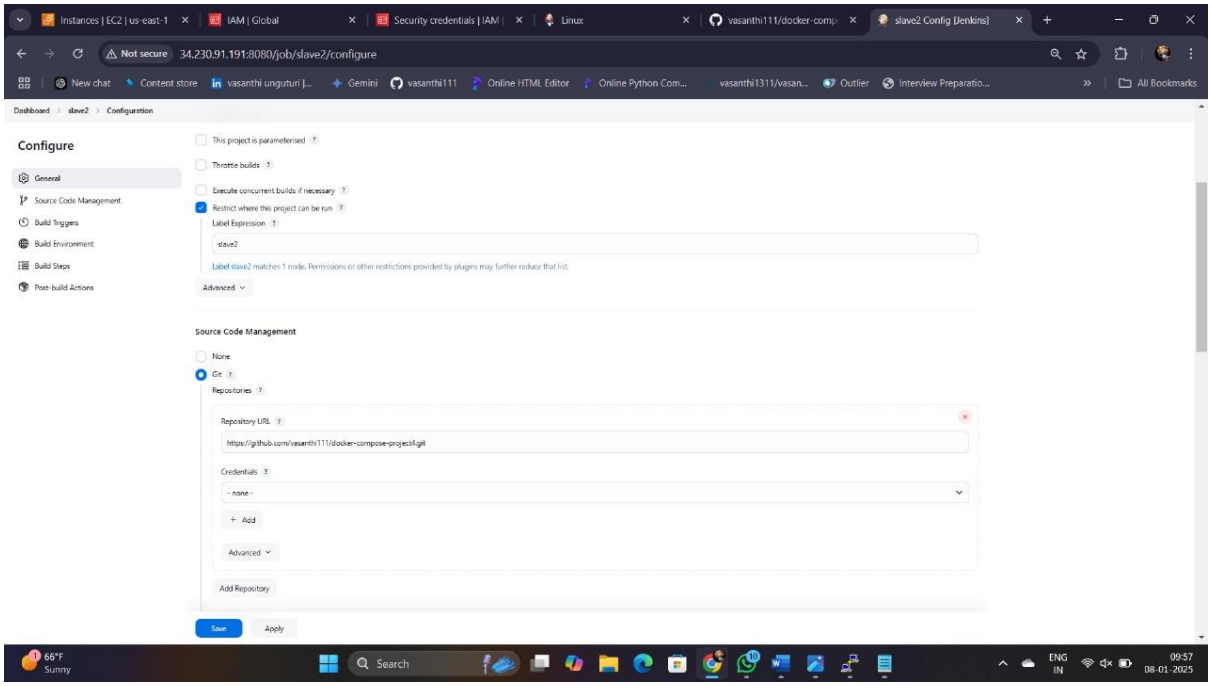
SLAVE-2:



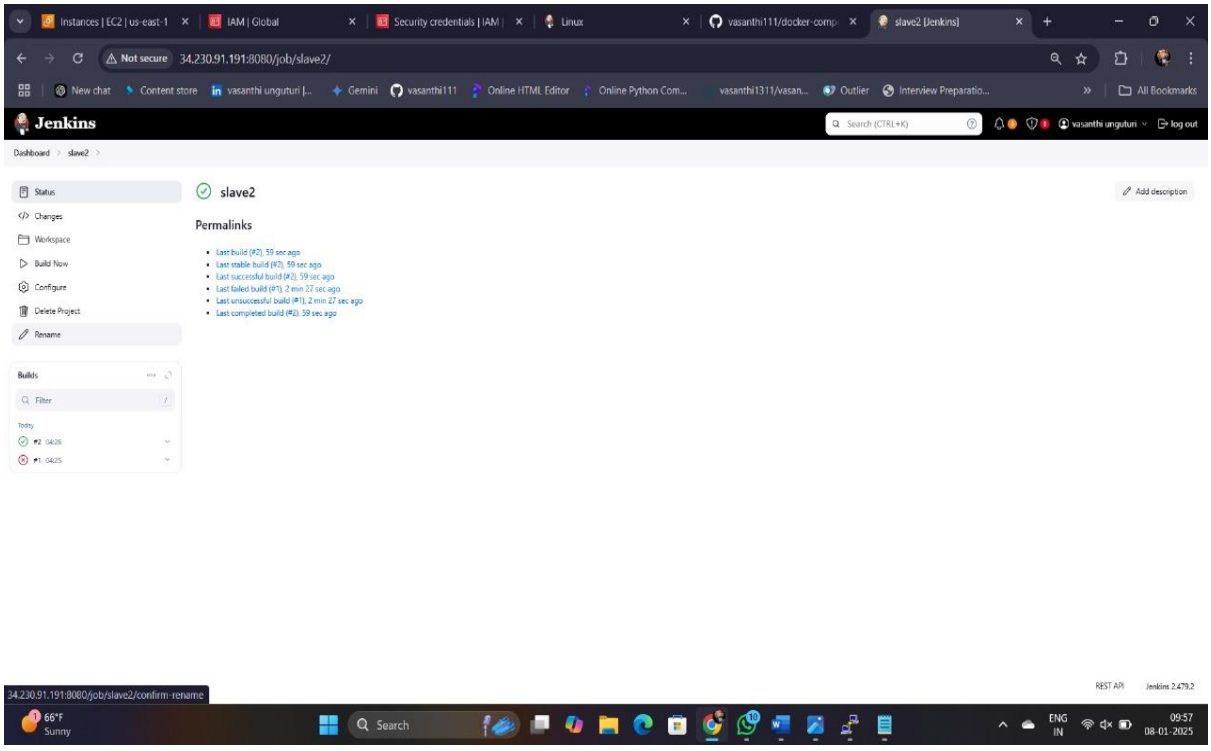
In slave2 we need to deploy the wordpress application using the docker and docker compose. Provided the permissions for docker compose. Started the docker service.



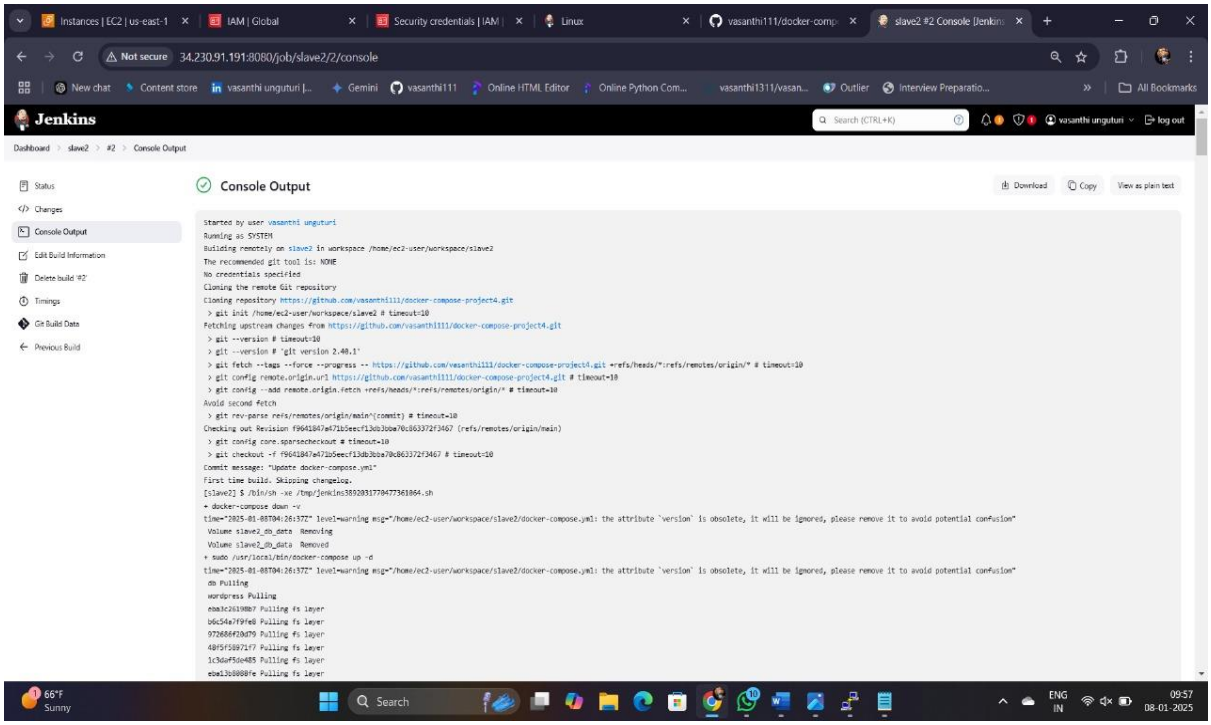
In Jenkins give the sudo permissions to install any required packages or softwares in the slave servers.



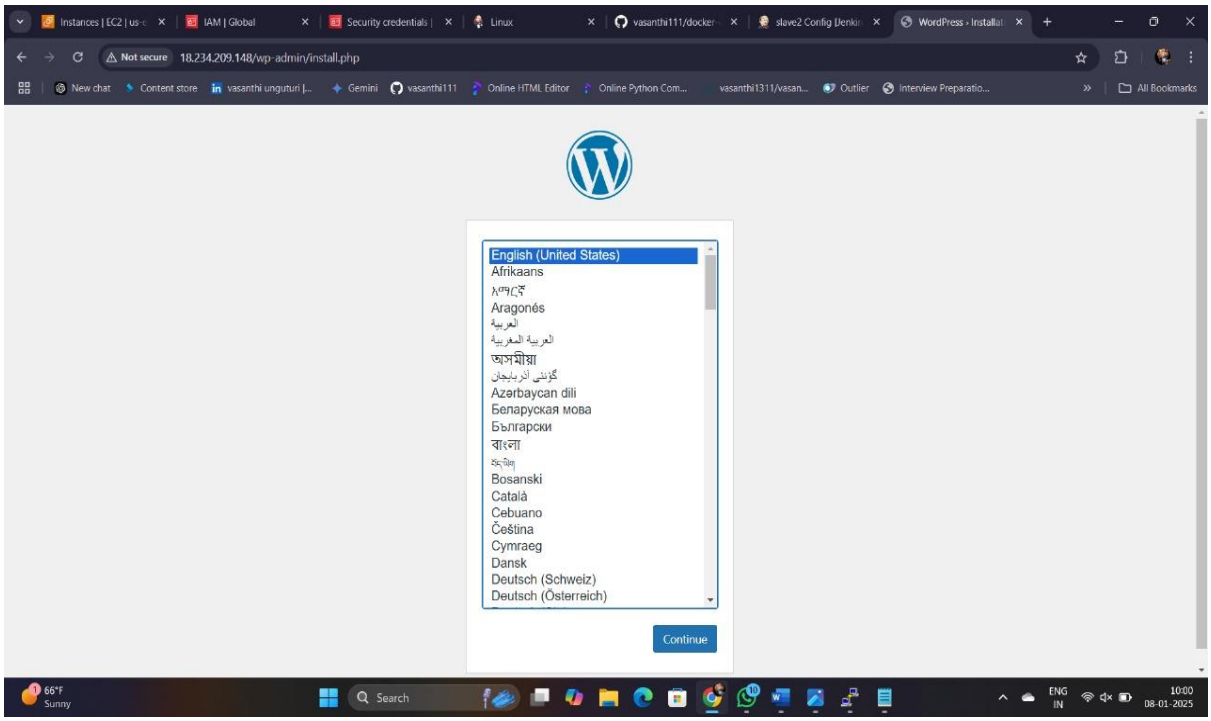
Select the created label and give the github repository url here.



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

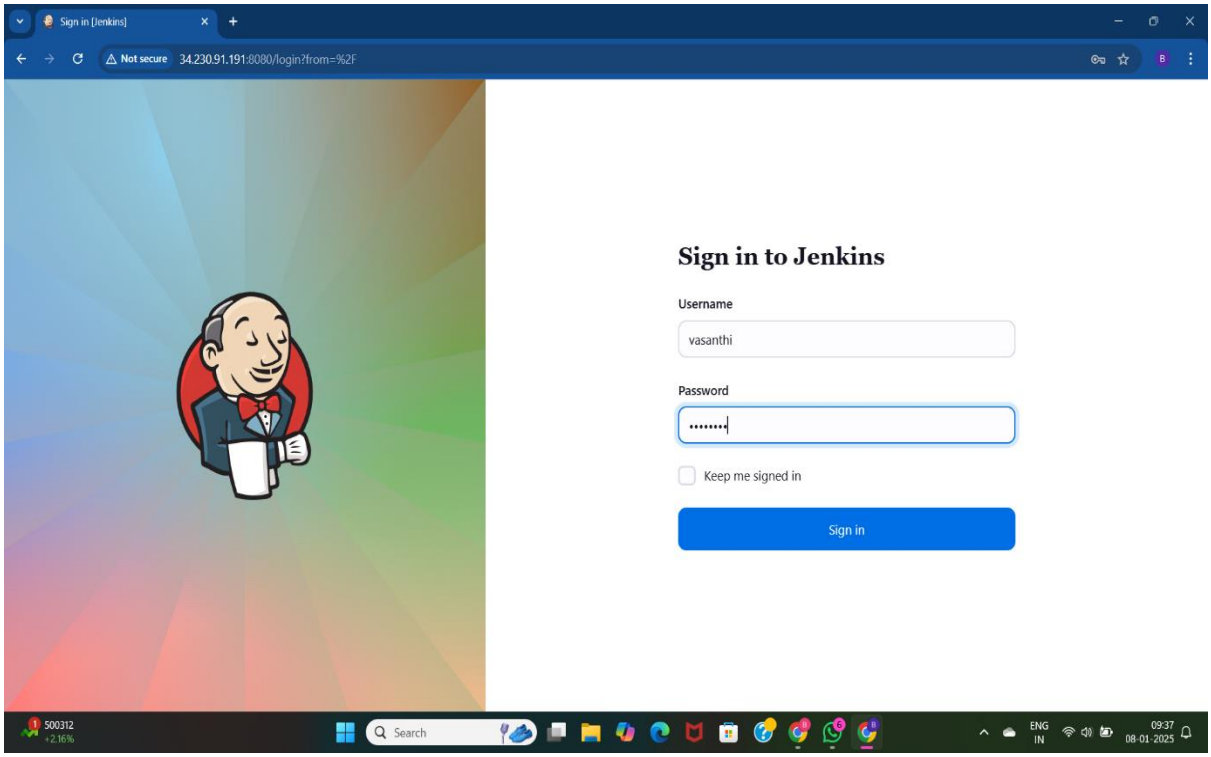


In the console output we can check the detailed information of the project.

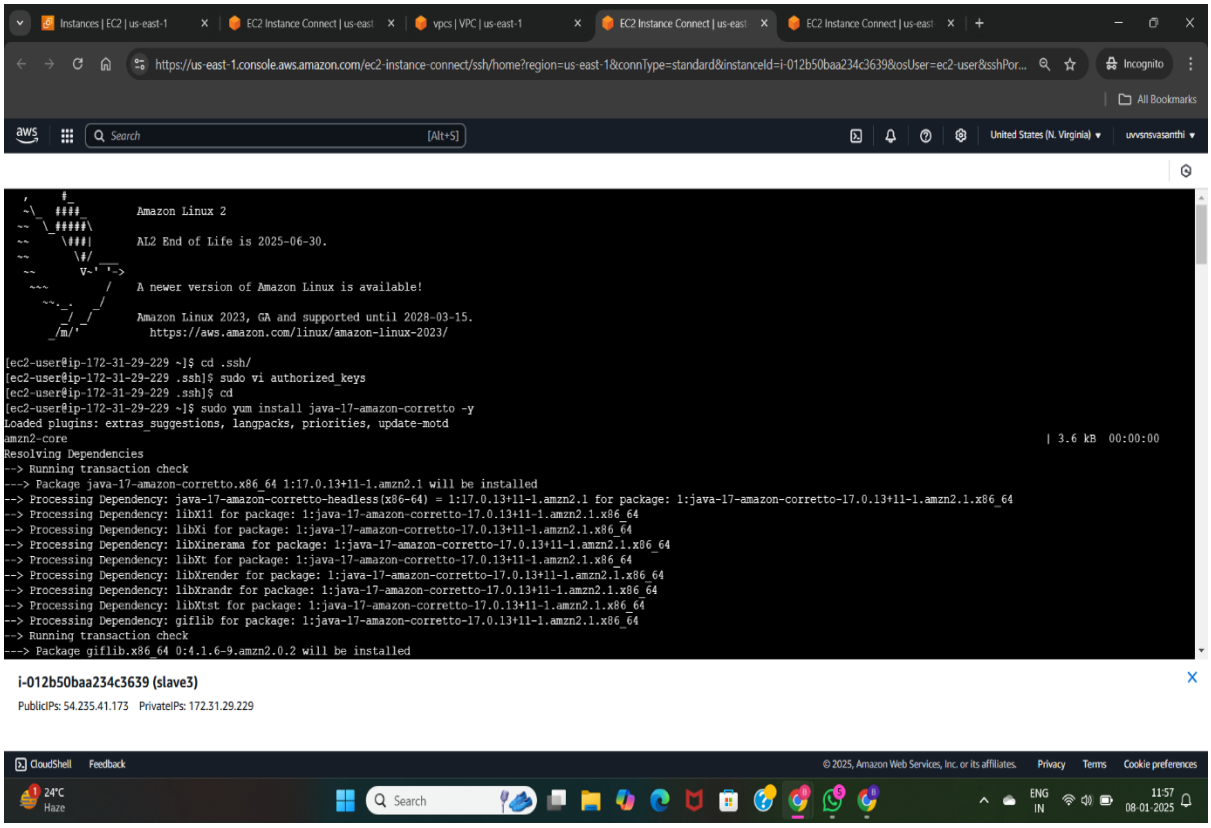


Now access the deployed application using the public IP address of the slave server.

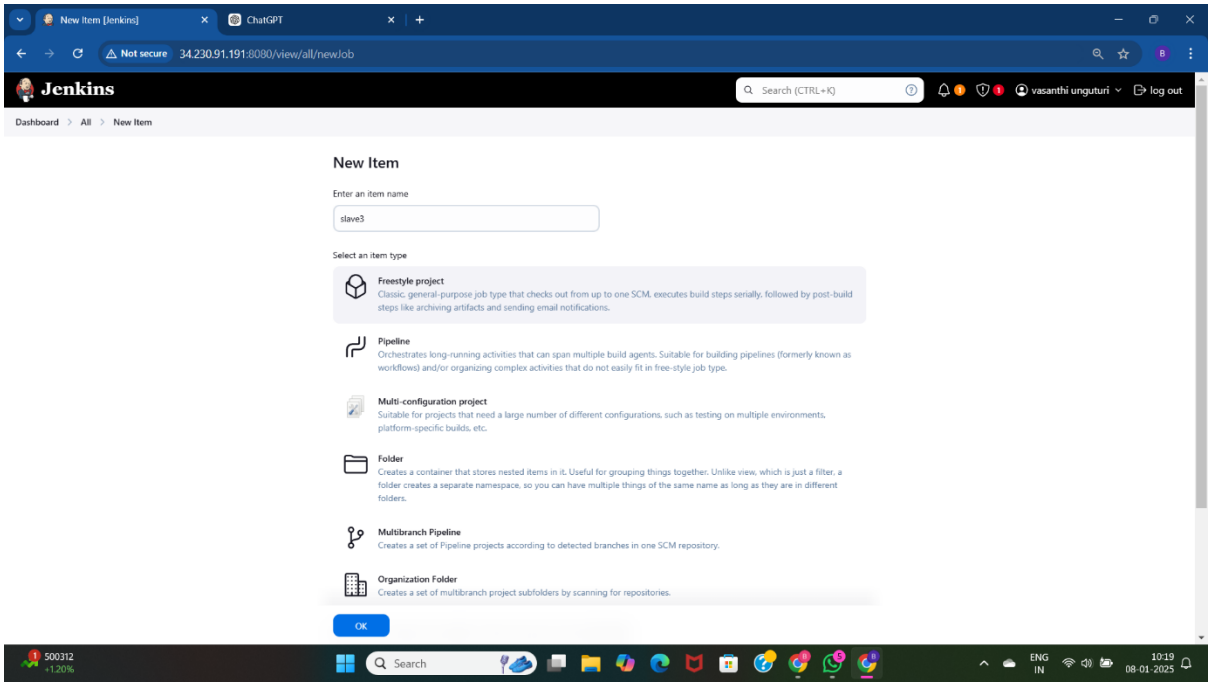
SLAVE3:



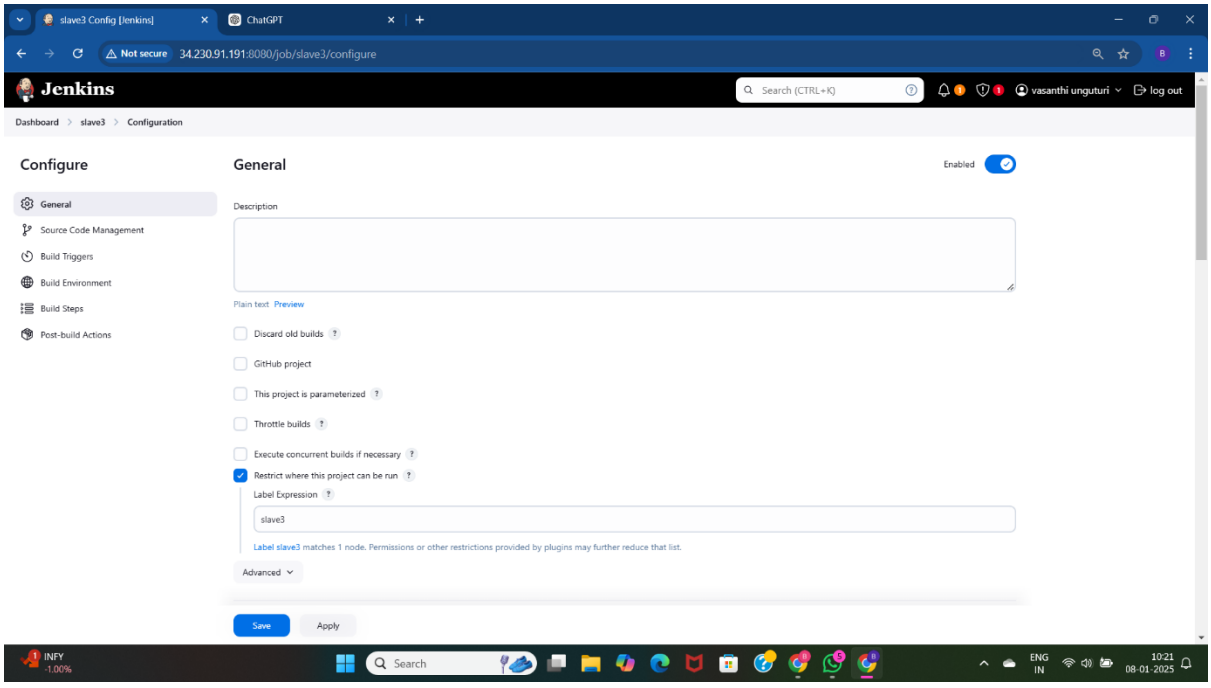
As the master is different person the Jenkins url is shared with the team. Here we are accessing the Jenkins using the username and password.



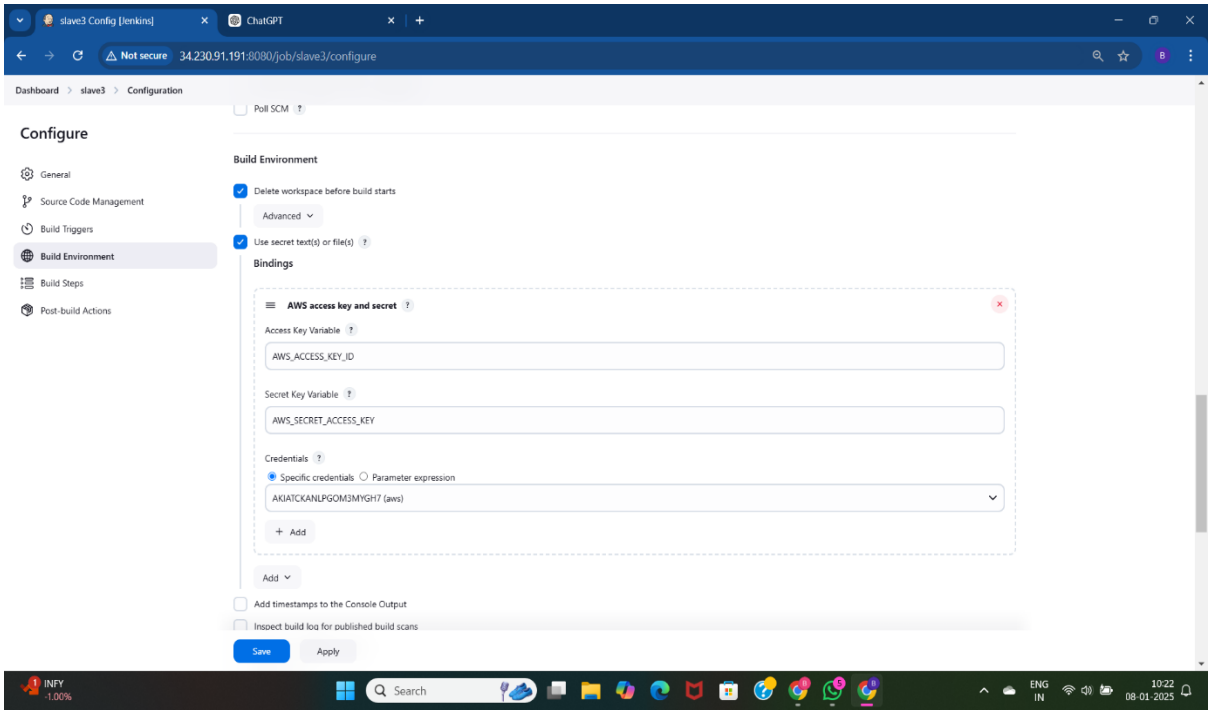
In slave3 server I have installed the java17 and connected it with the master using public key of master.



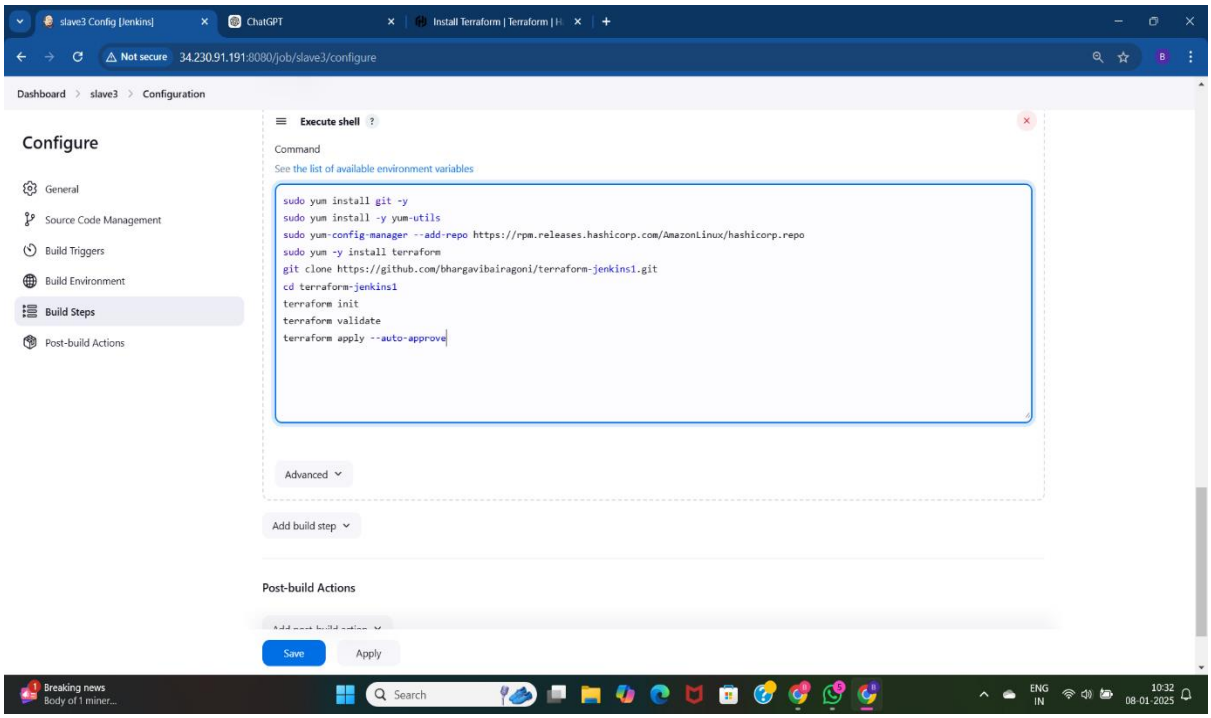
I have created a slave3 free style job for deploying the application using the terraform.



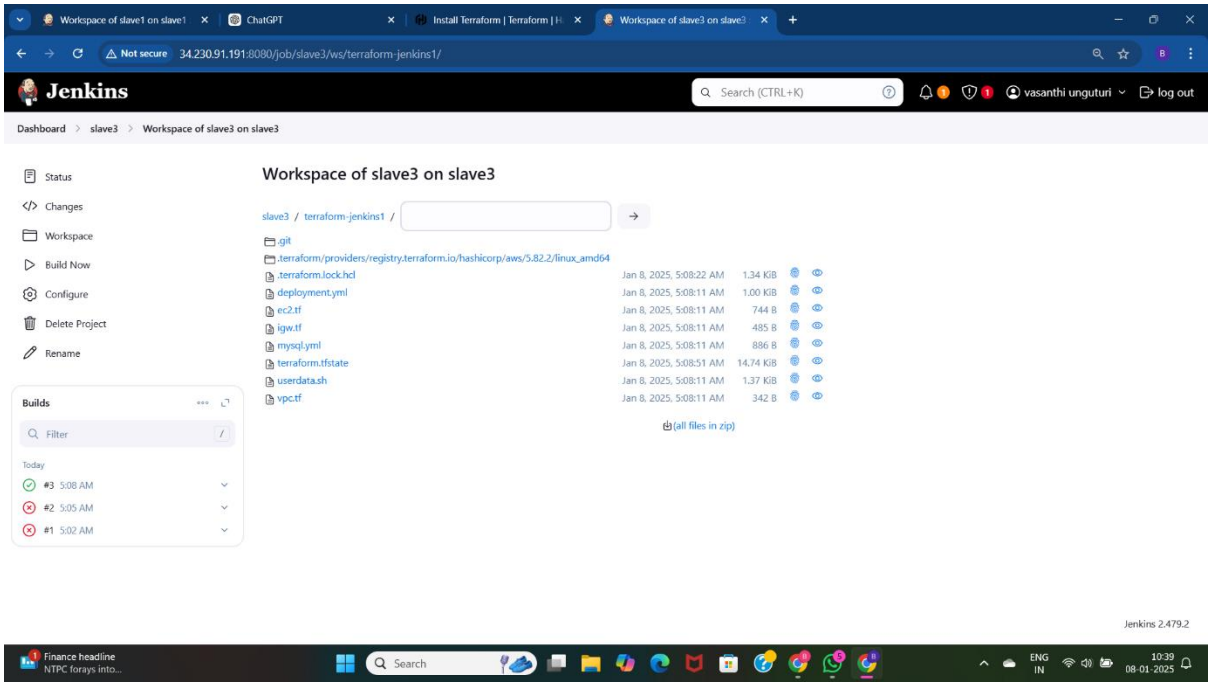
Select the created label here to run the builds.



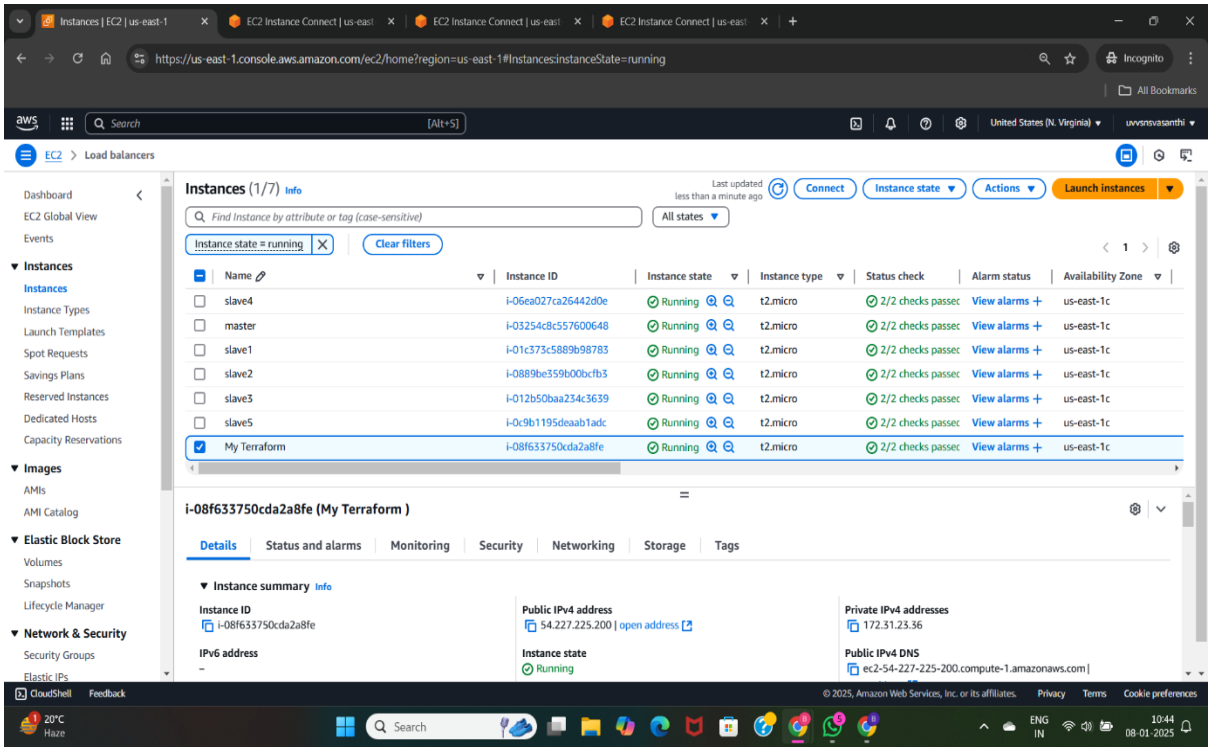
Now give the aws access keys and secret access keys using the build environment.



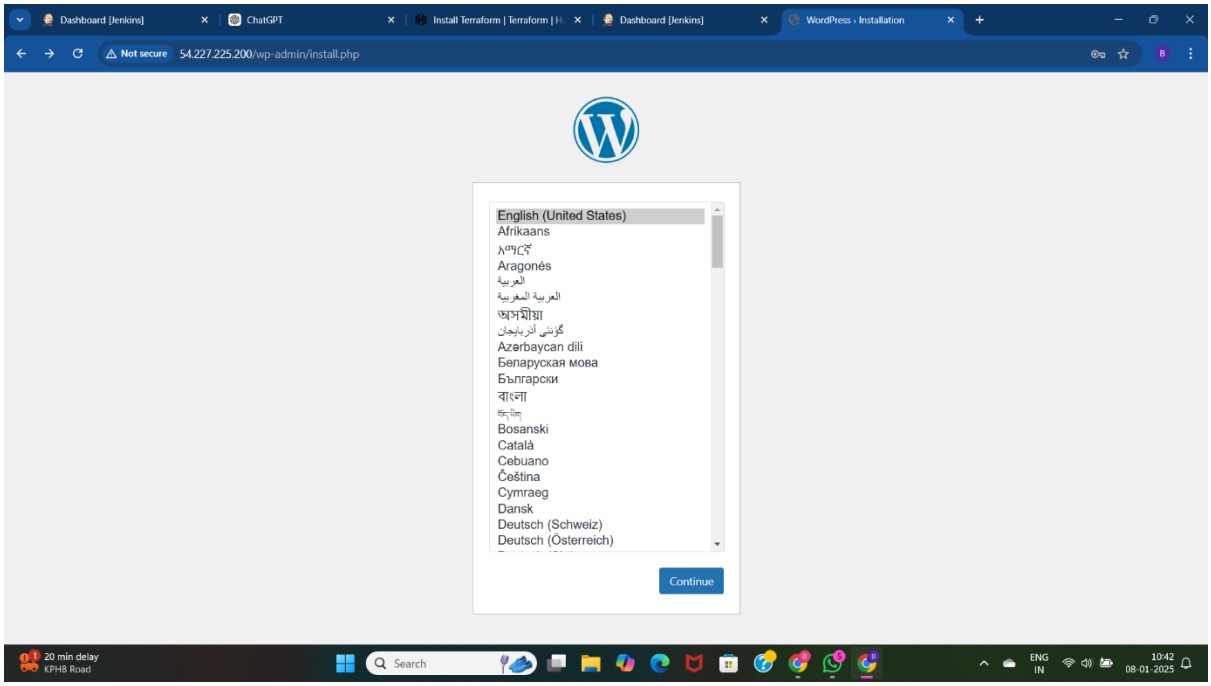
In execute shell write the commands to install git, terraform. Provide the github repository url which is having the terraform files to create the aws resources. Select the branch where we are having the code/files. Give the terraform commands to see and create the resources.



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

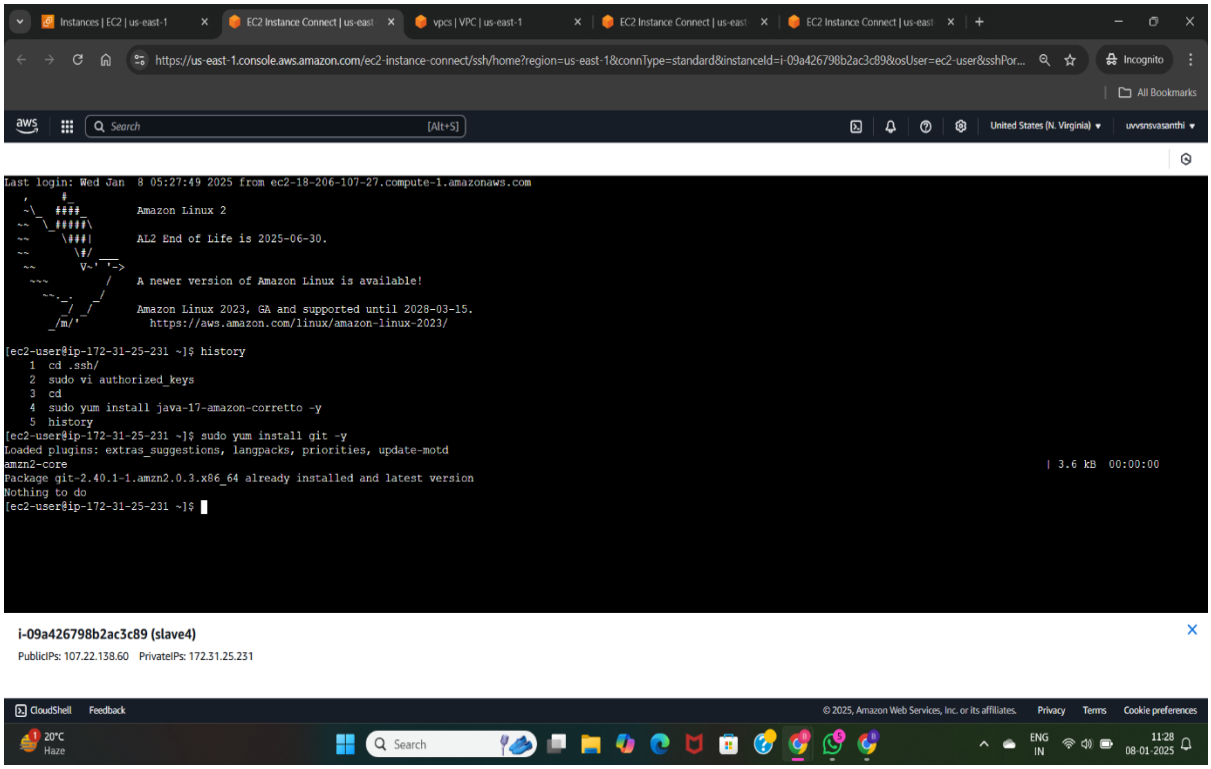


A new ec2 instance is created when I have build the project. Now use this instance and access the application.

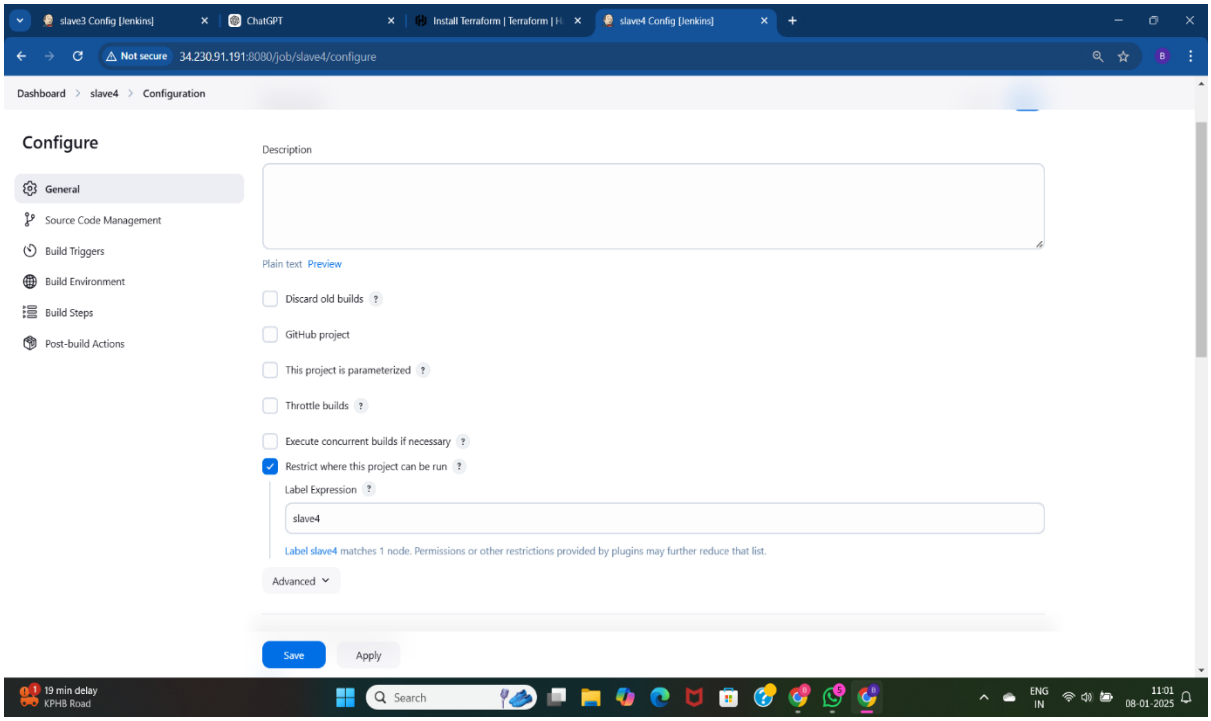


I have taken the newly created instance public IP address and accessed the wordpress application.

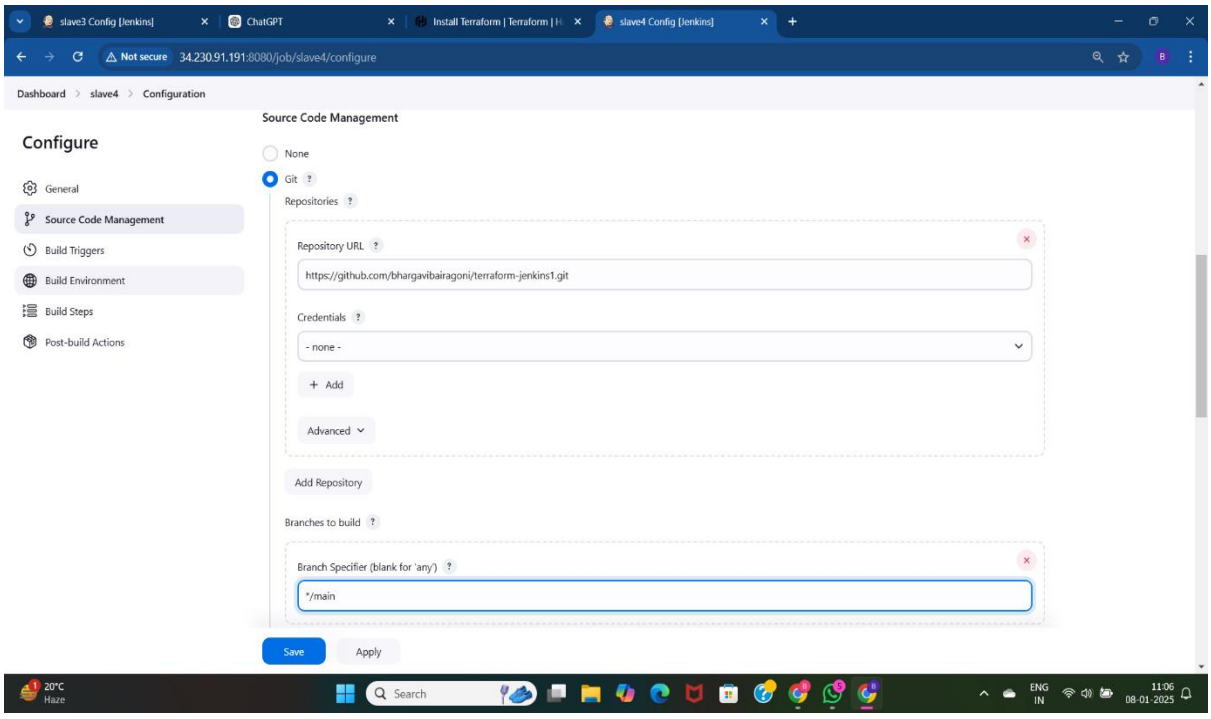
SLAVE-4:



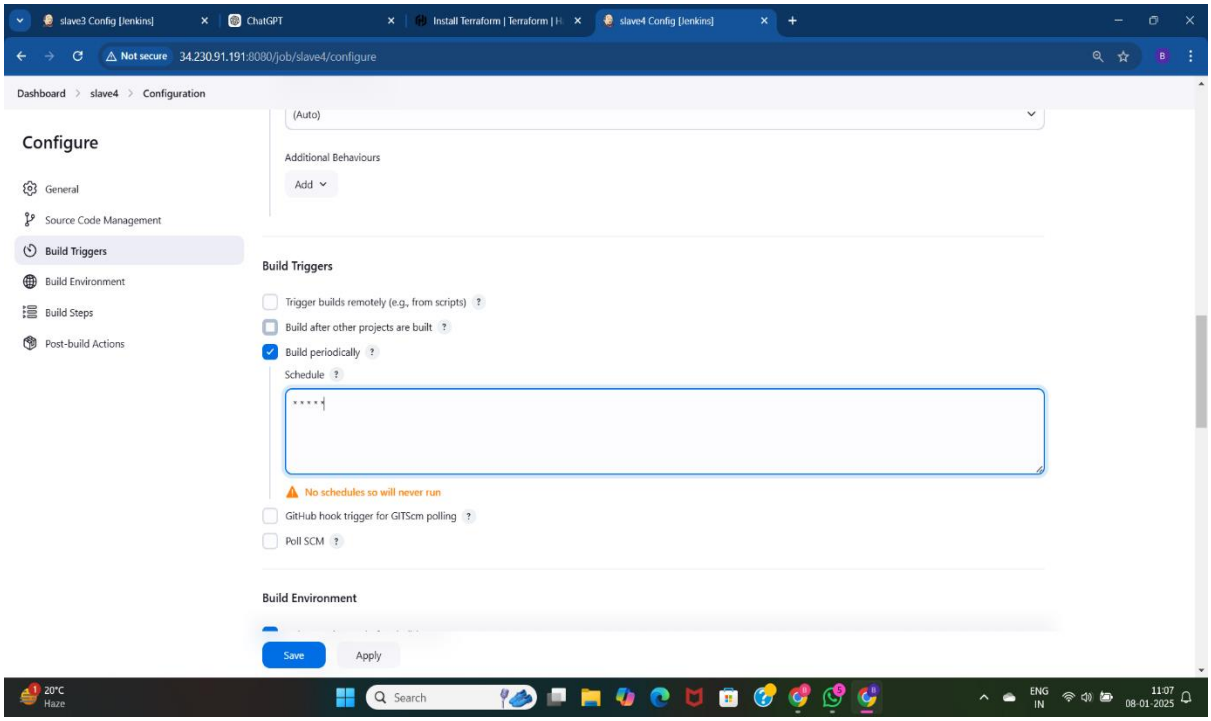
In slave 4 I have installed java17 and git. Connected it with the master.



In Jenkins job select the created label for slave4.

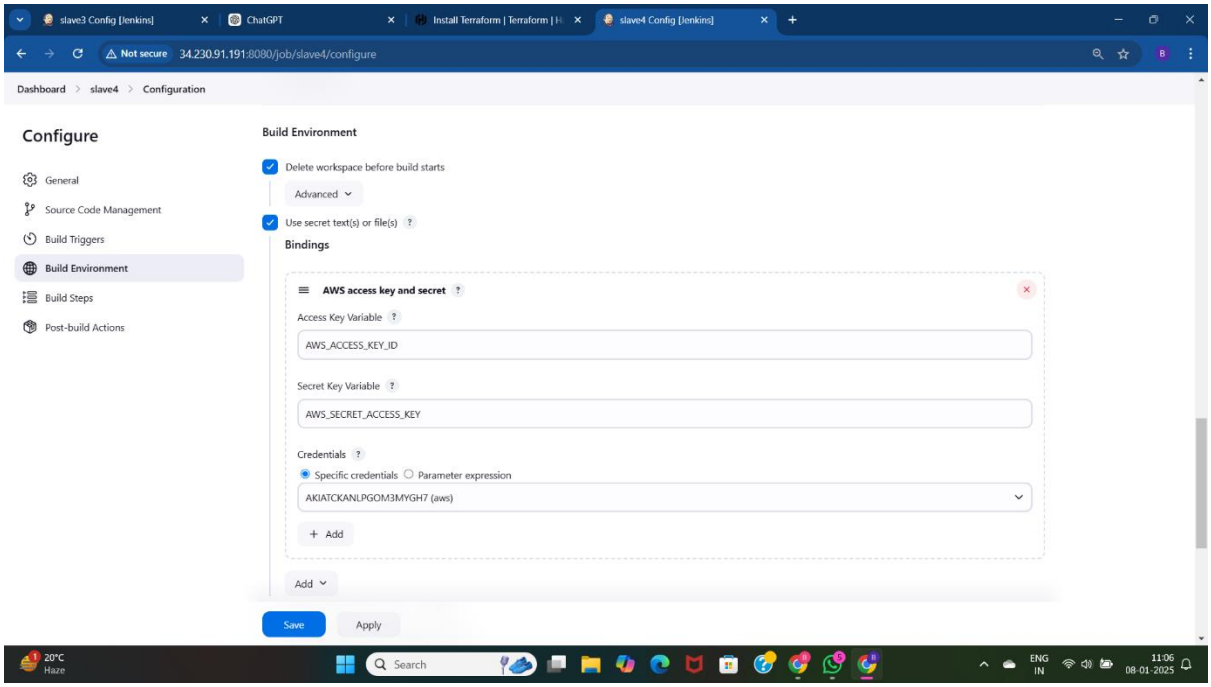


Select the github repository and the branch where we are having the terraform files.

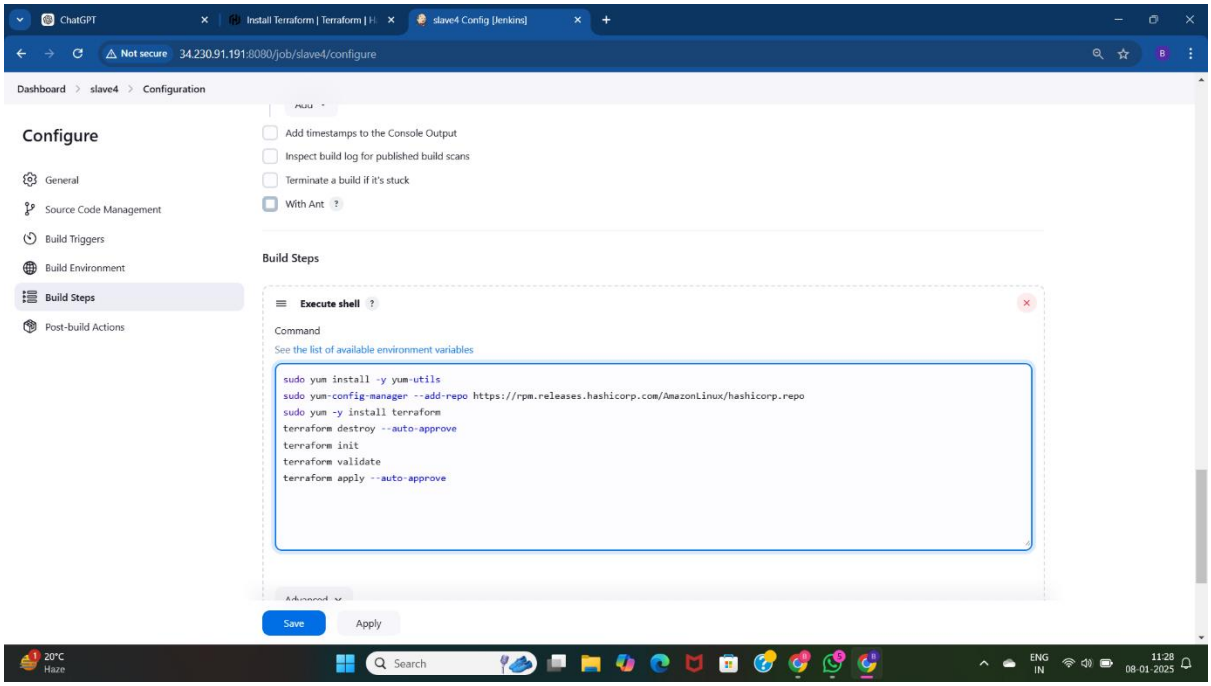


To deploy the application automatically select the build periodically option, give the syntax there.

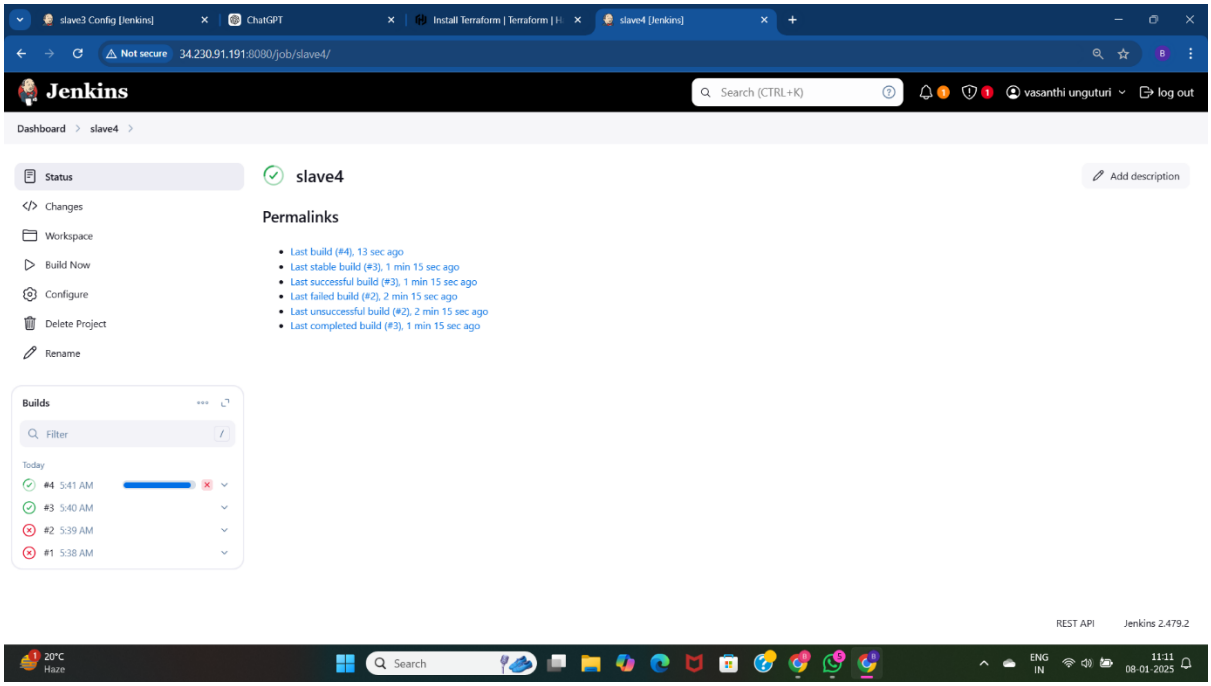
(minute, hour, day of the month, month, day of the week) .



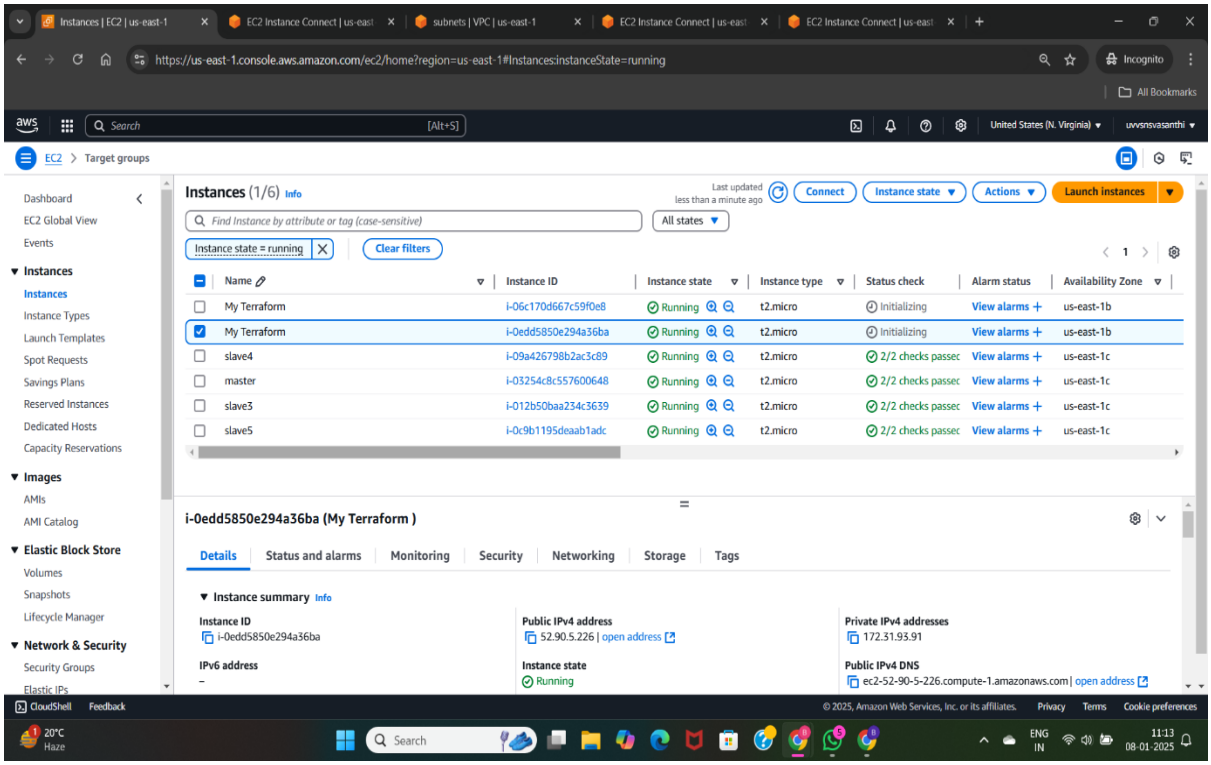
Select the created credentials using the build environment> use secret text or files.



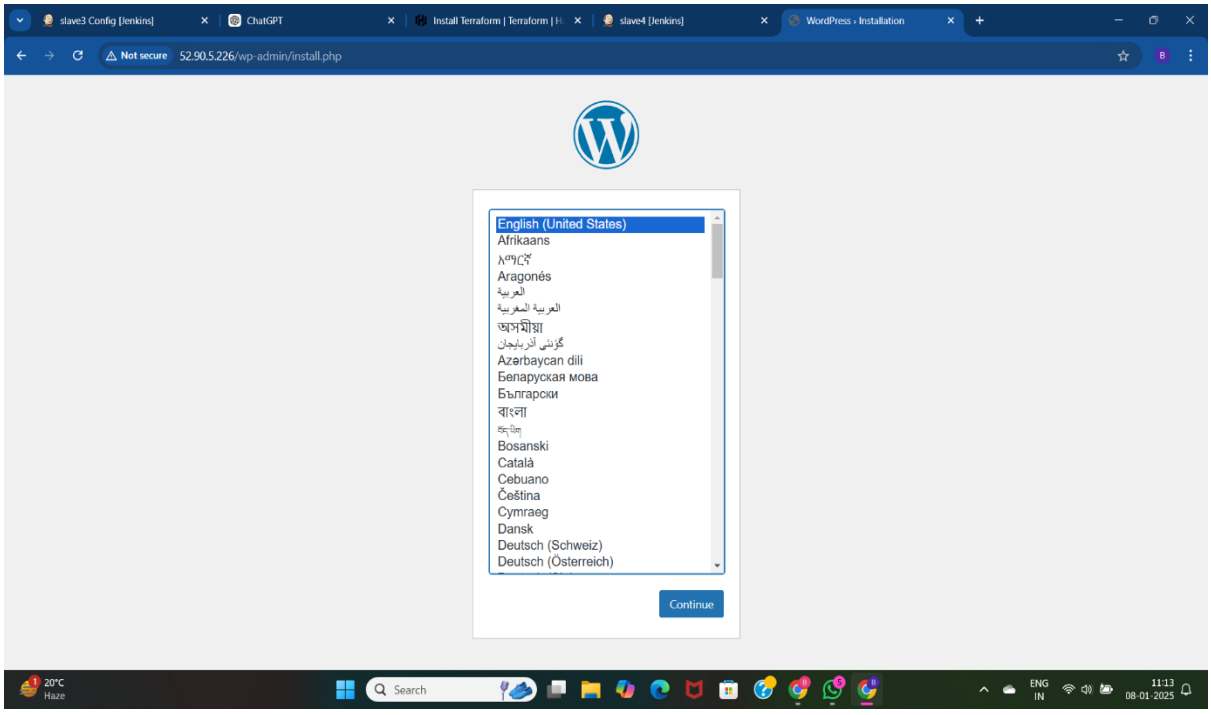
In execute shell provide the commands to install terraform and check the terraform plan and create the resources using “terraform apply –auto-approve”.



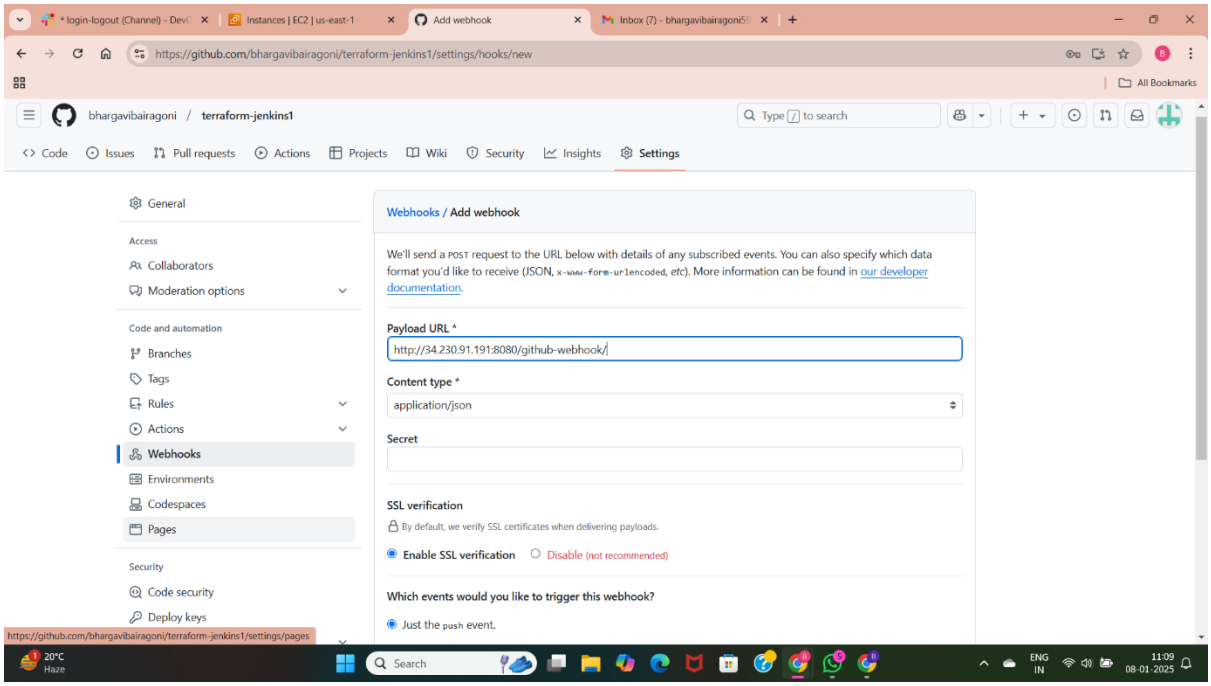
We can see the build is started automatically. We can also see the time gap between every build is one minute which matches which the syntax.



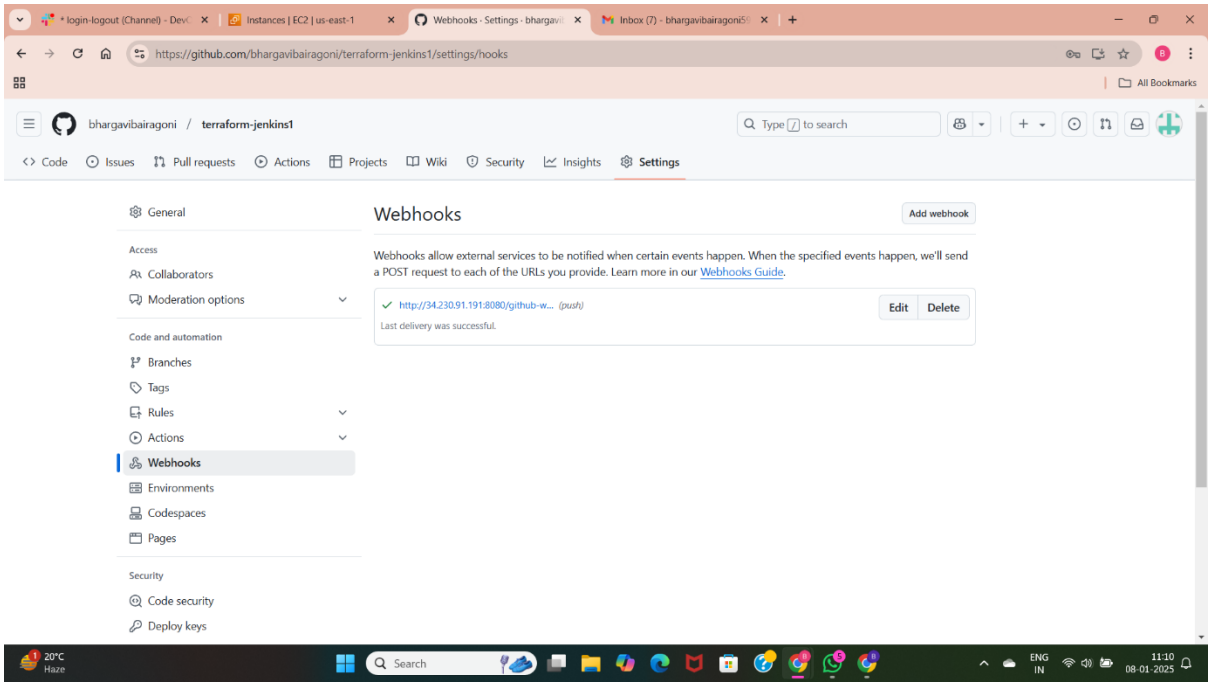
As the builds are running automatically new resources are creating in the aws console. Now use this public ip to check with the application.



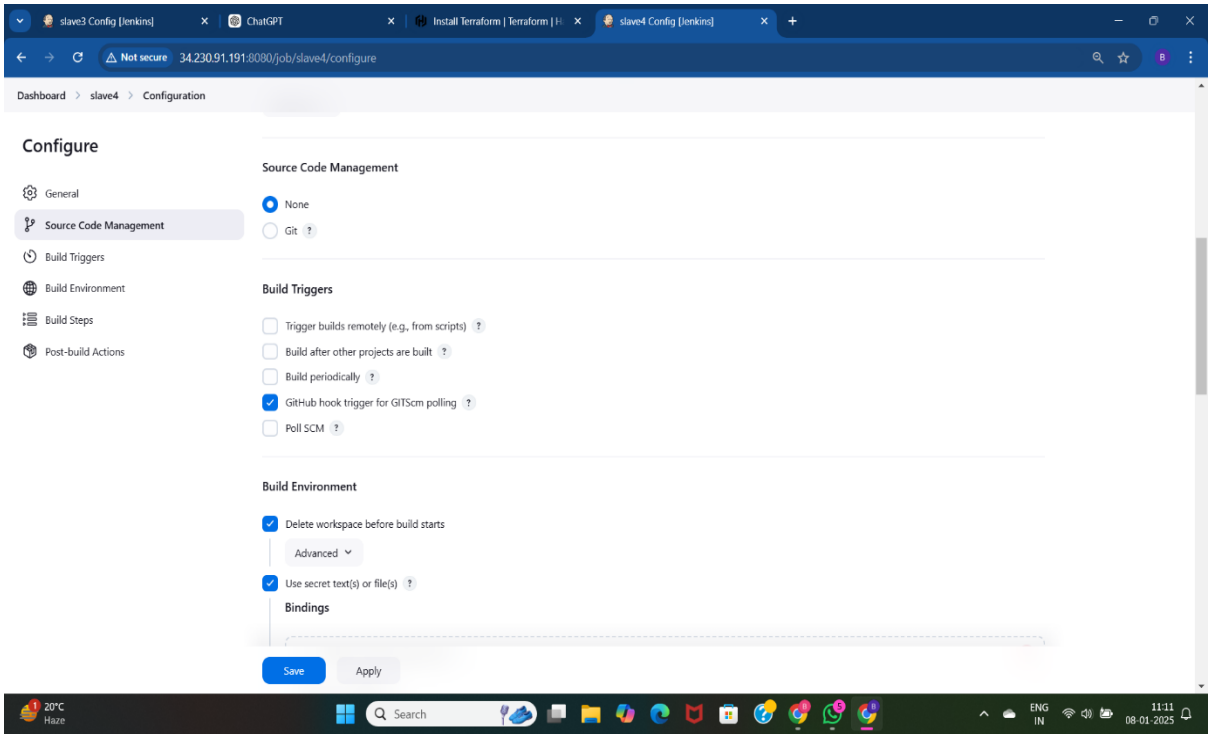
The application is deployed using the build periodically option.



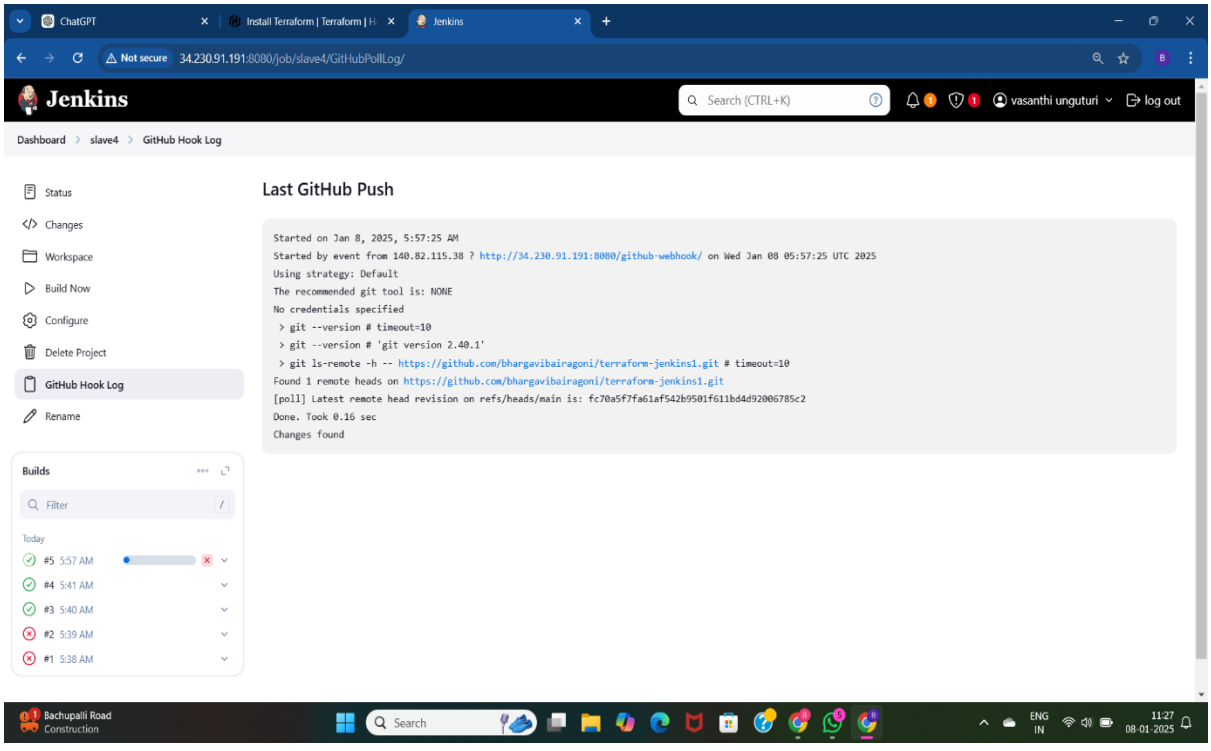
Now create the github webhooks in the repository where we are having the terraform files. Give the Jenkins url in the playload url. Select the content type as application/json.



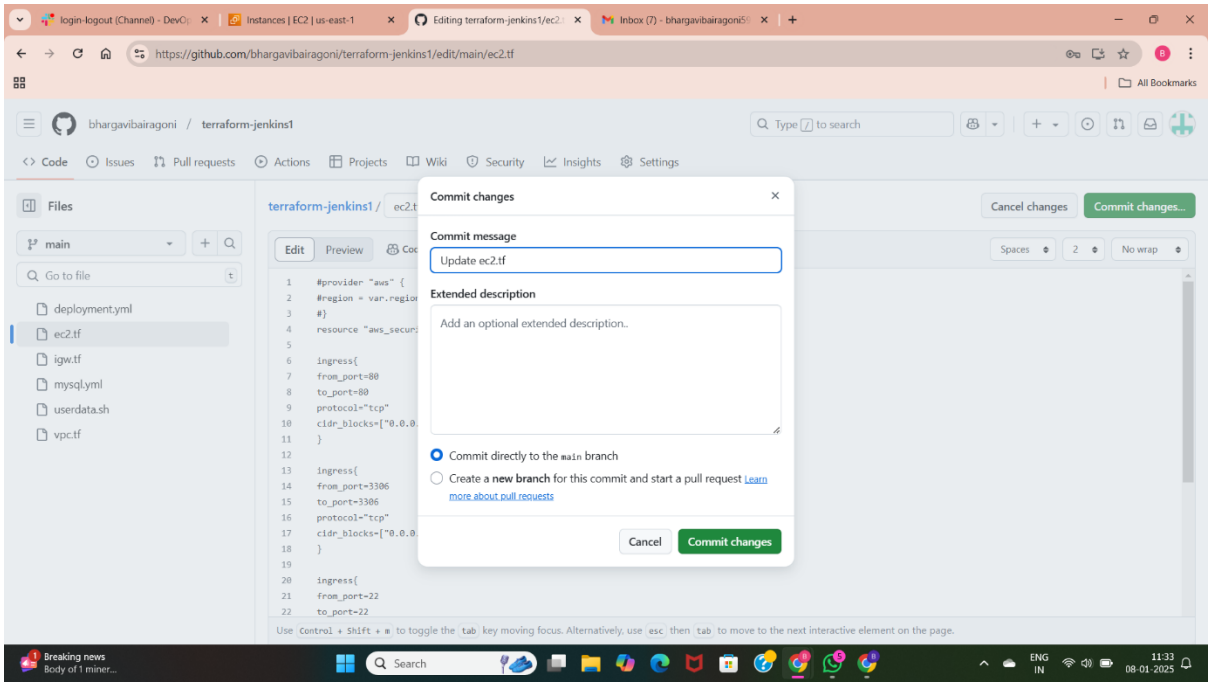
The webhook is created successfully. From now whenever we made changes we can automate the builds.



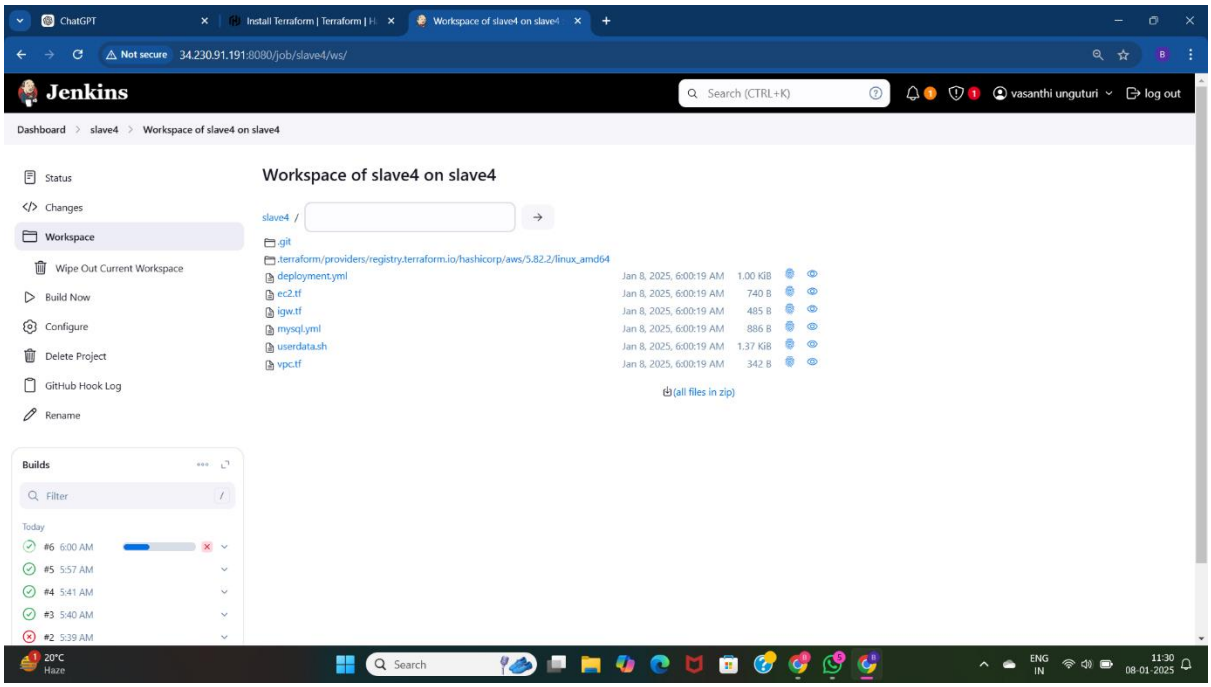
Now select the github hook trigger for gitscm polling and save the Jenkins job, Change code in the github repository to trigger the builds.



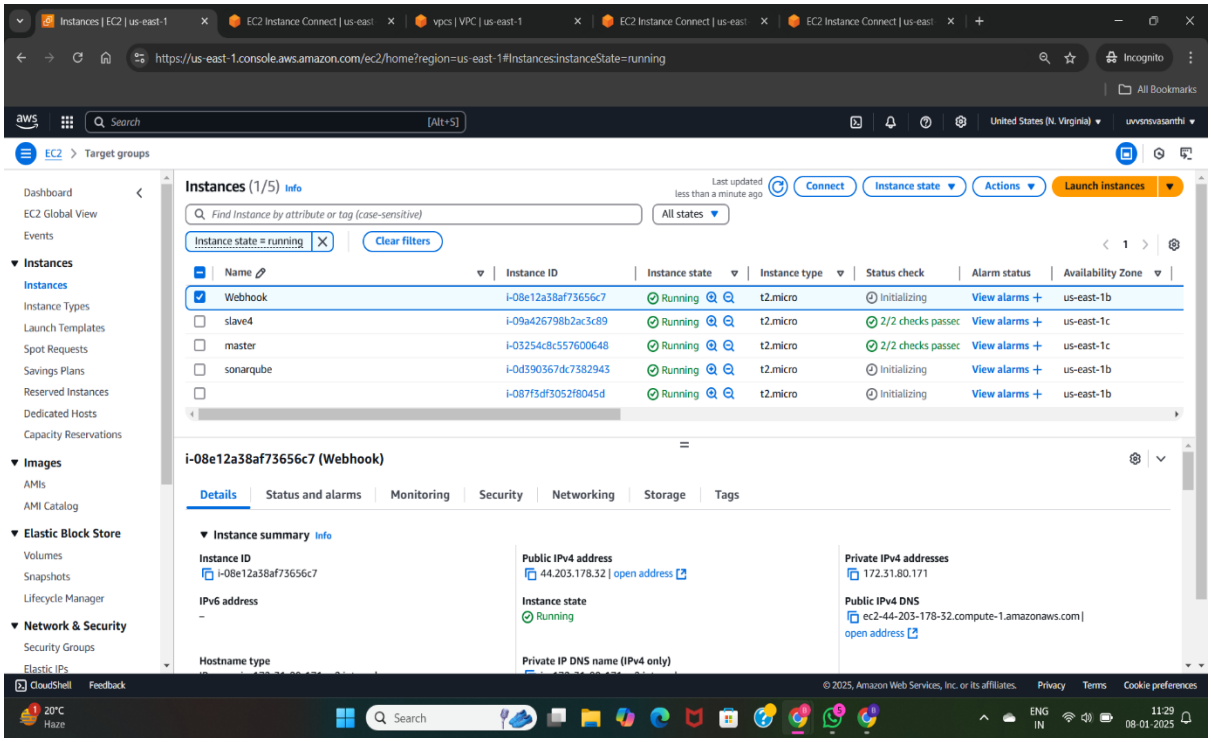
Here we can see the new option “github hook log” which will trigger from the github repository.



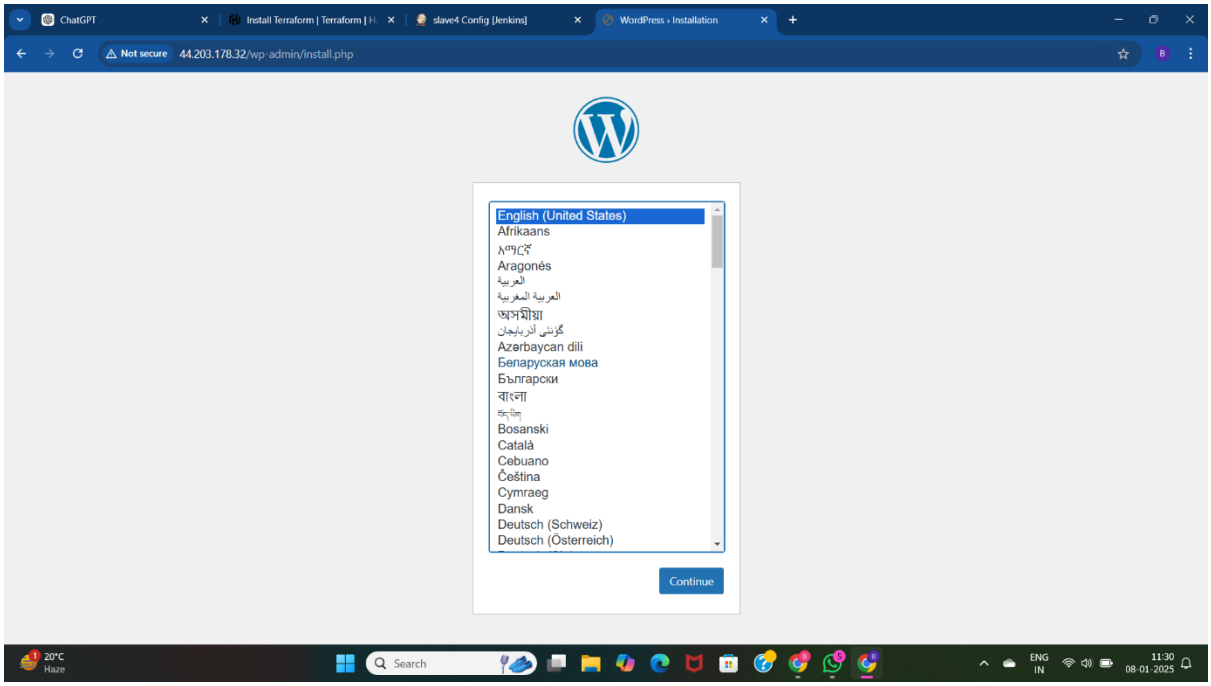
Here am changing the code in the terraform repository and commiting the changes to trigger the builds.



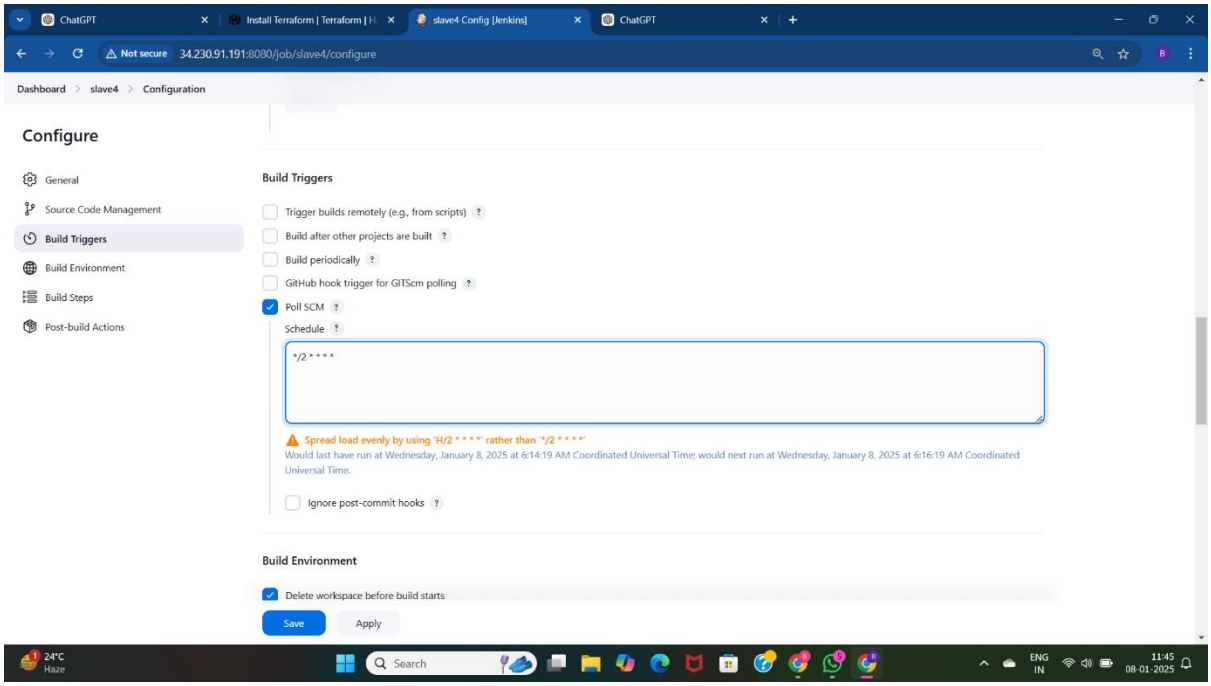
Here we can see the builds are happening automatically when we do changes in the code.



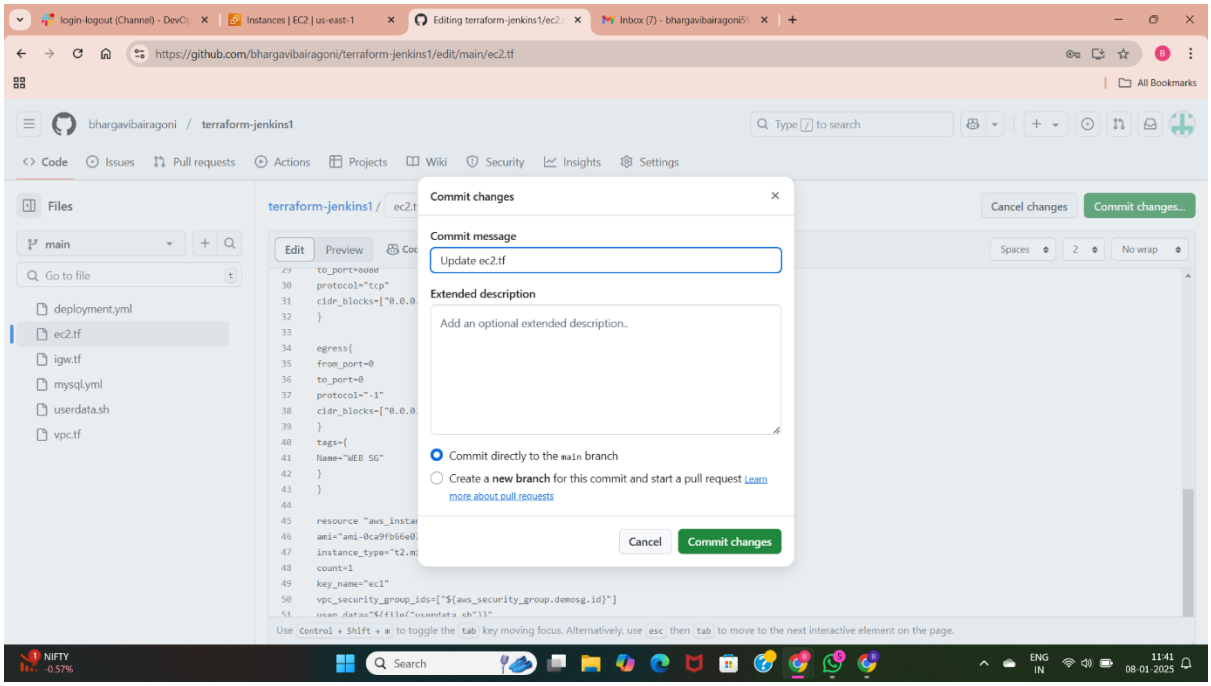
Here a new instance is created with the terraform code.



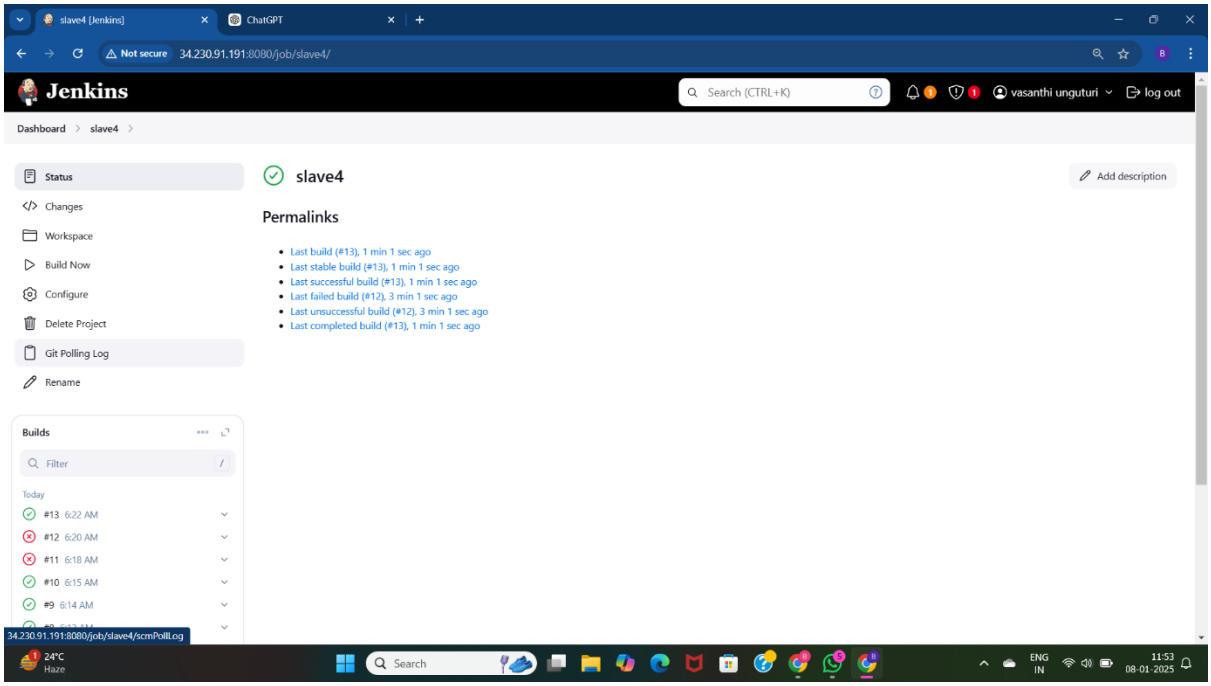
With the created new instance public Ip address a able to access the application.



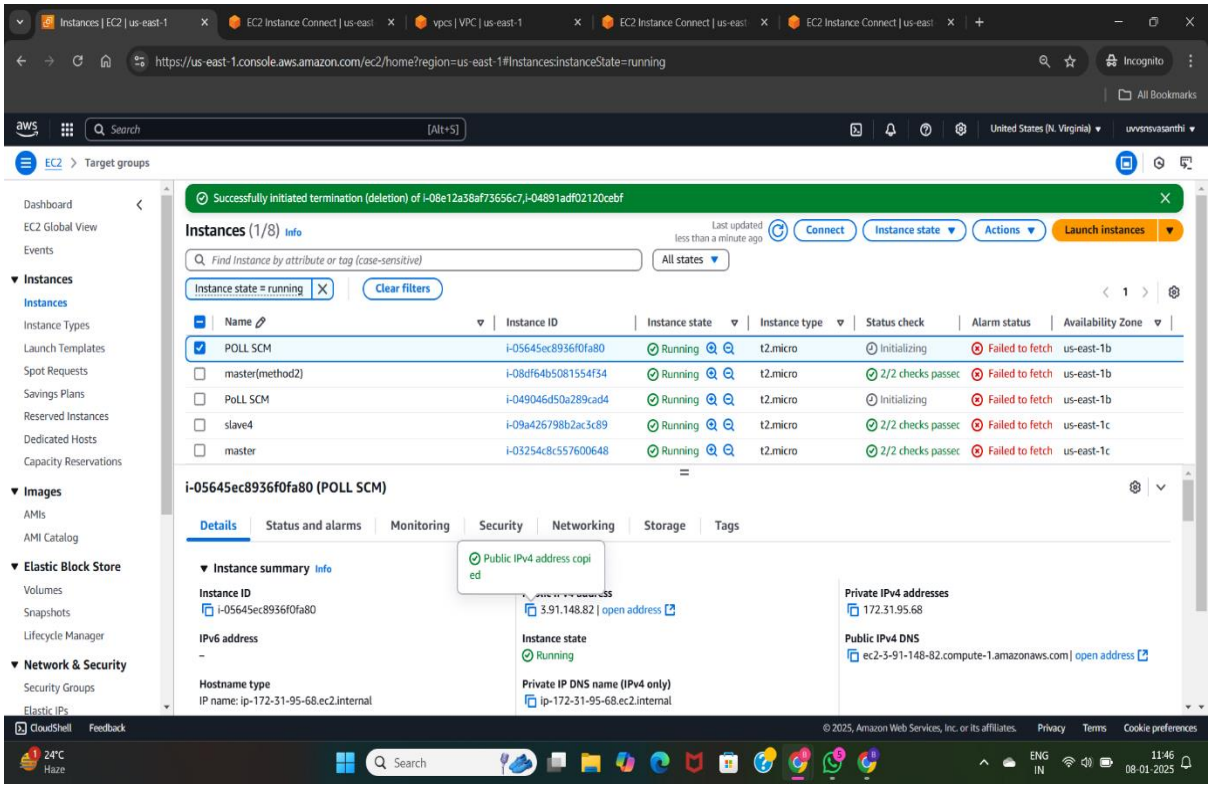
Now select the poll scm for checking the automatic builds in the Jenkins using webhook.



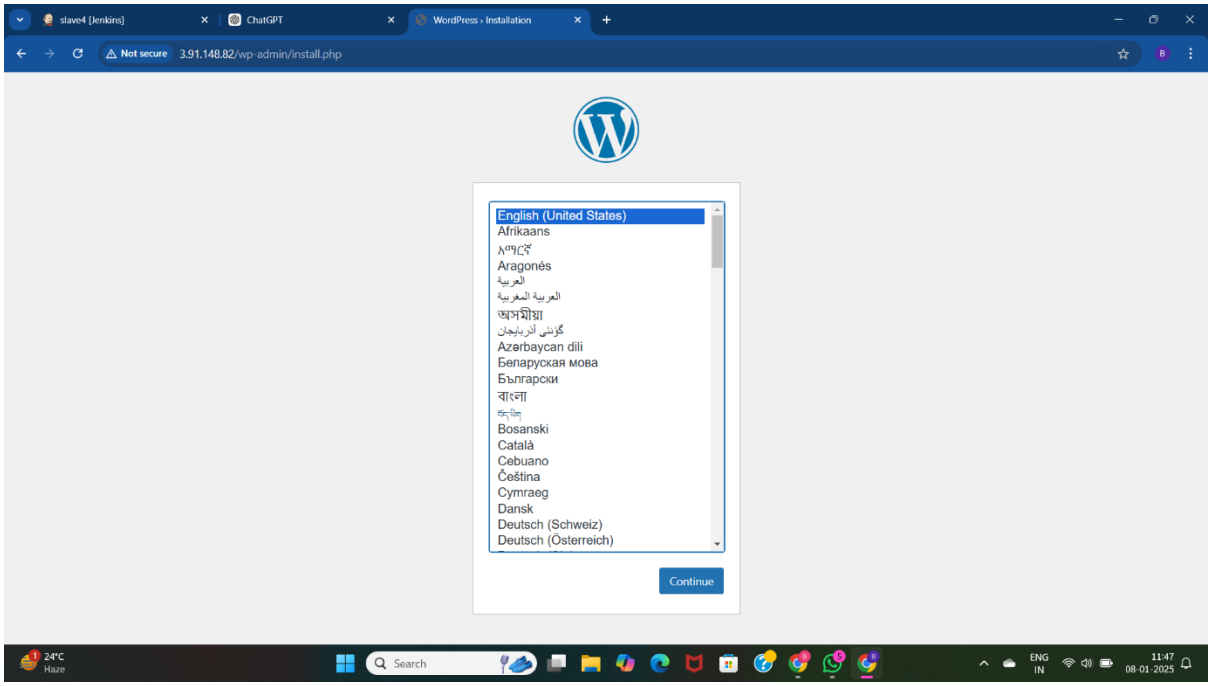
Change the code in the terraform files and commit the changes.



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

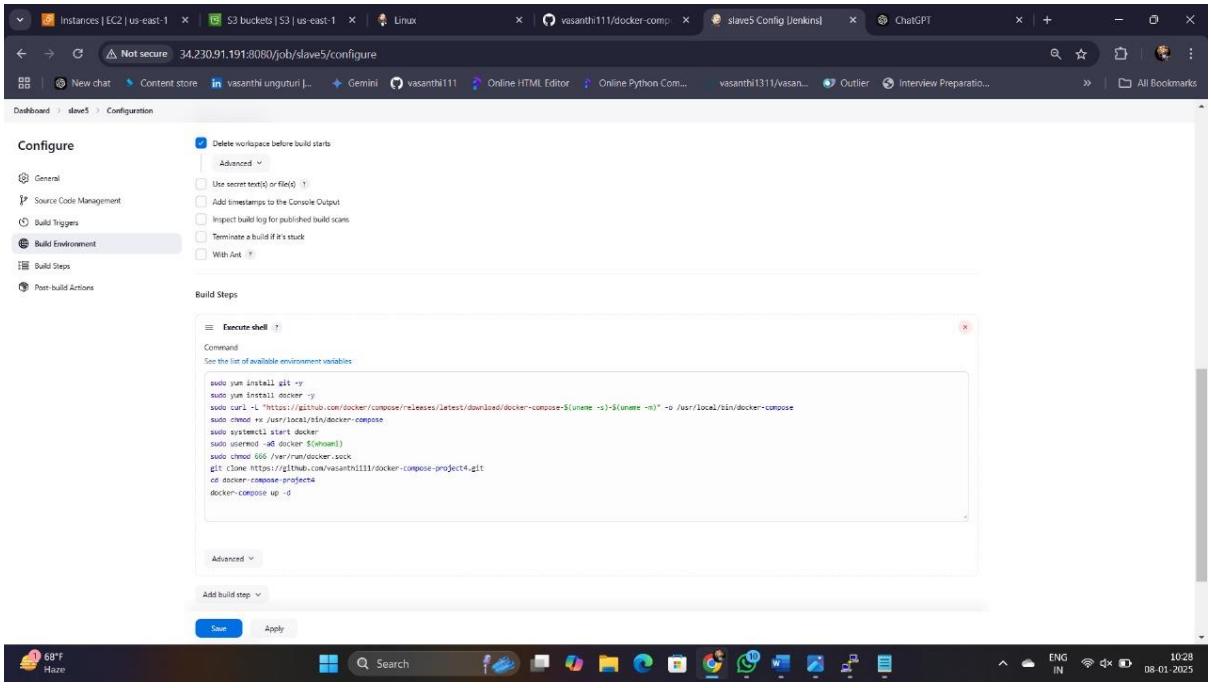


New instances are created using poll scm and terraform.

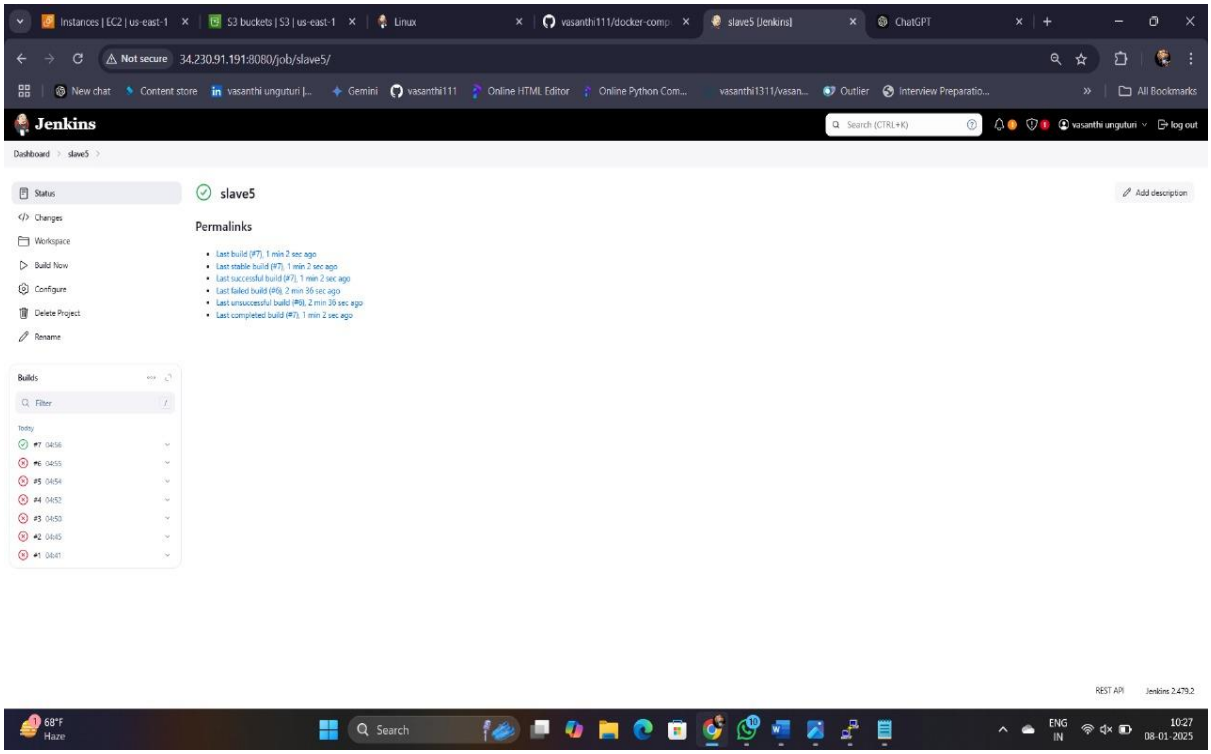


The application is deployed using the terraform, poll scm.

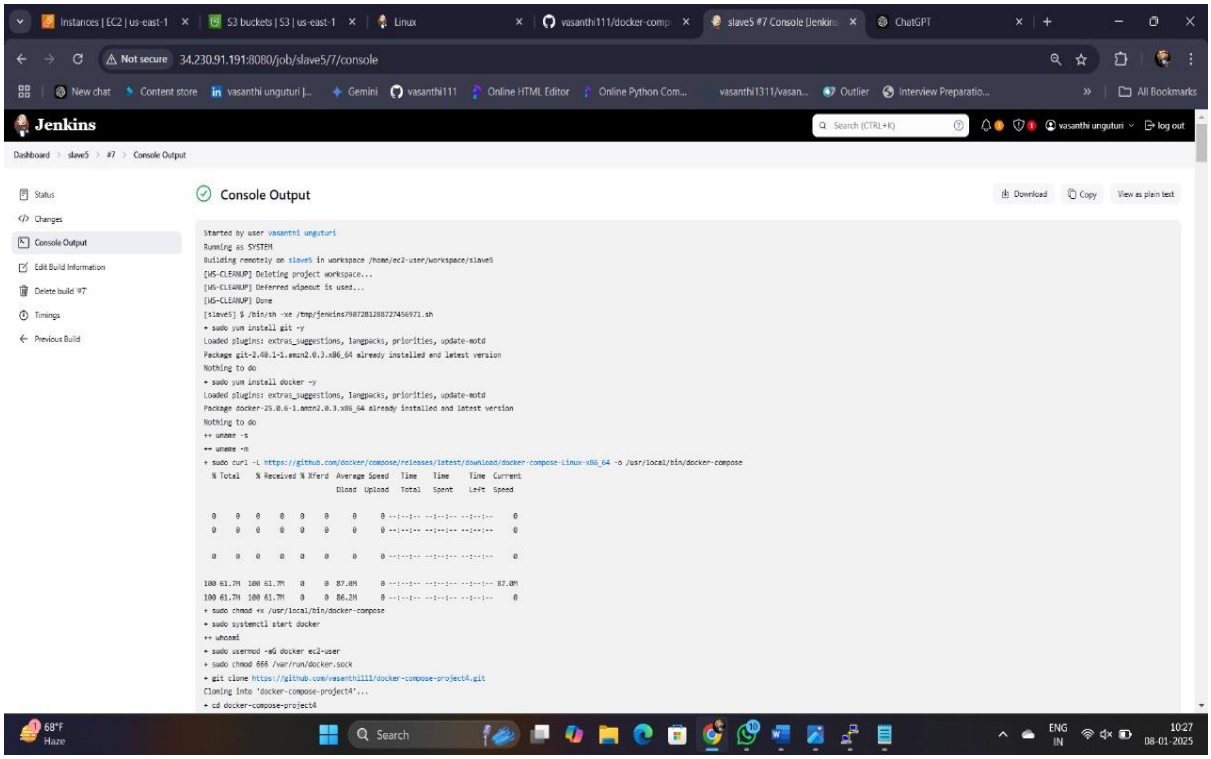
SLAVE-5:



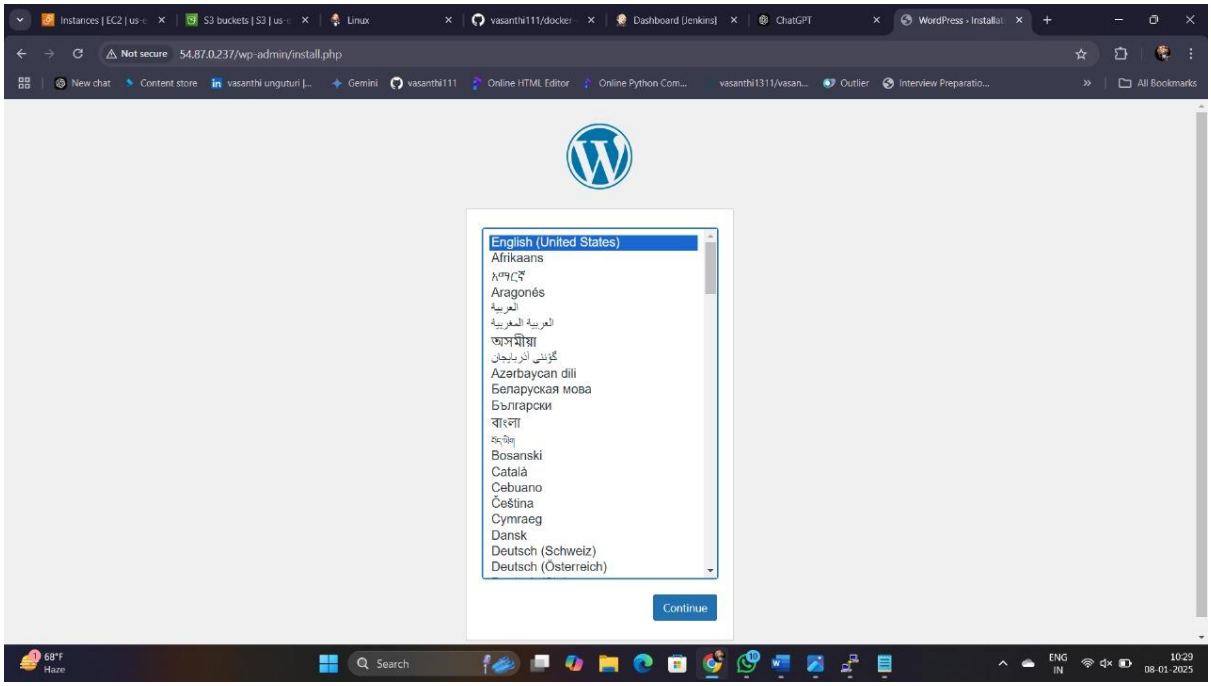
Now deploying the application using the bash script with docker, docker-compose. For that install and start docker. Install docker compose and give permissions.



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



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.