PROJECT

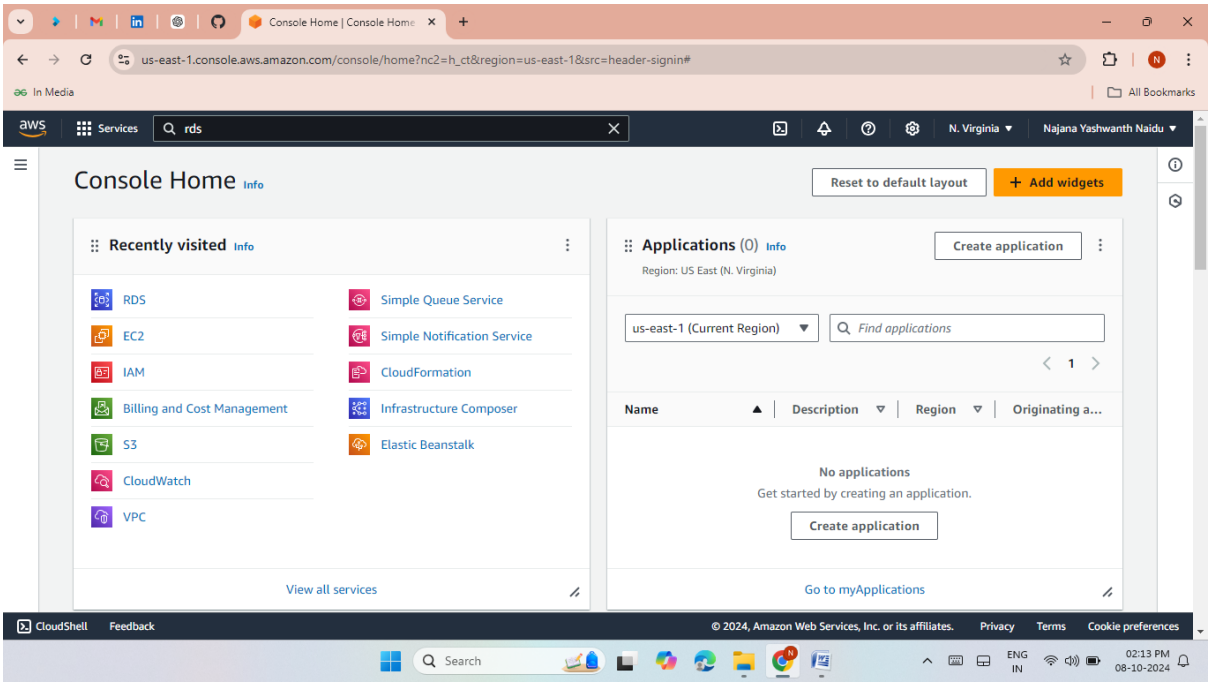
DEPLOY WORDPRESS

N.Yashwanth Naidu

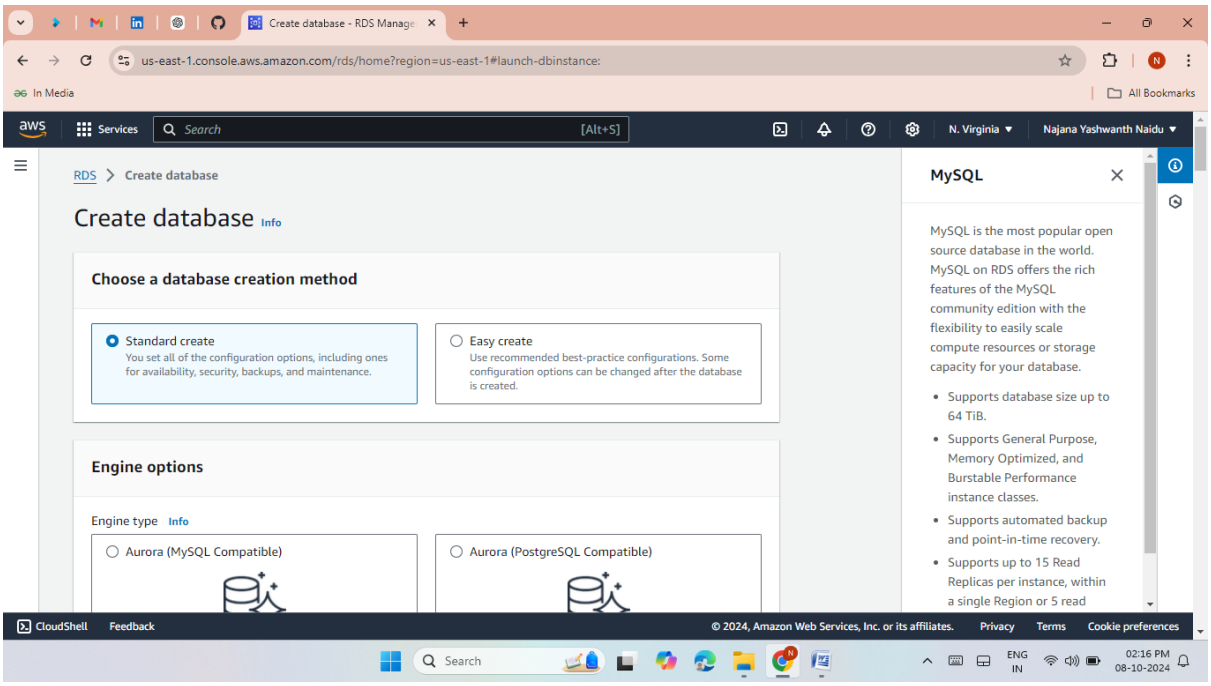
METHOD-1

Deploy WordPress web application by using AWS RDS(MYSQL) service (manually)

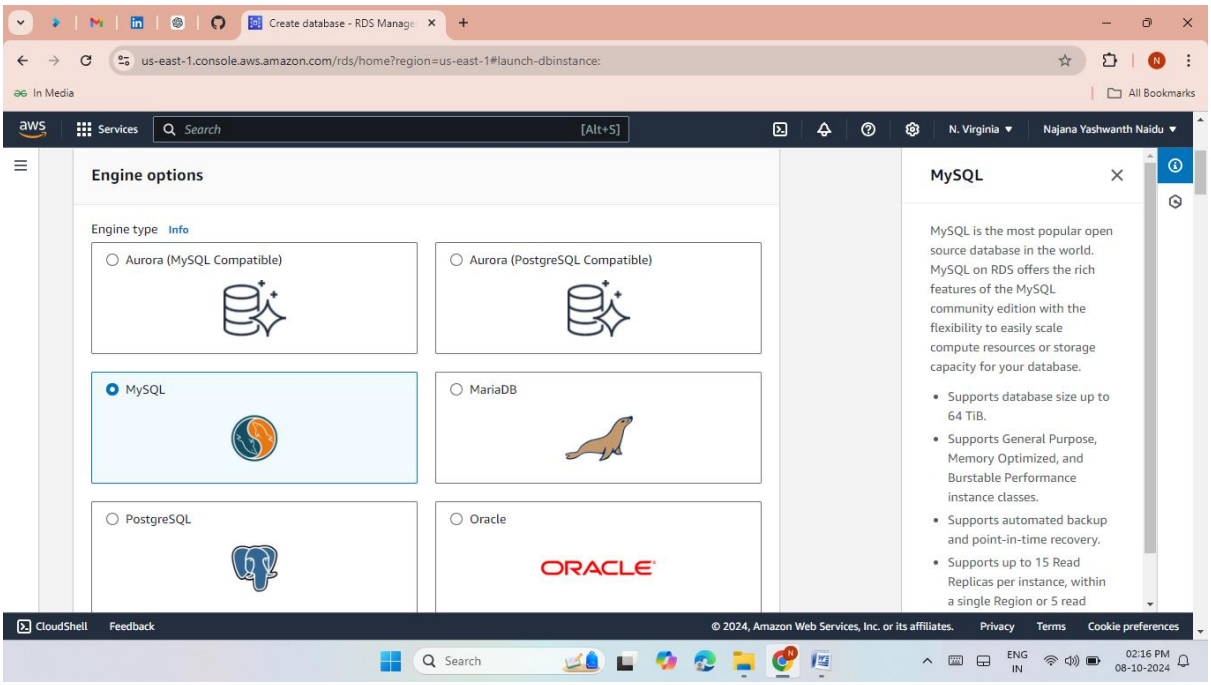
- First login into the AWS account then click on RDS.



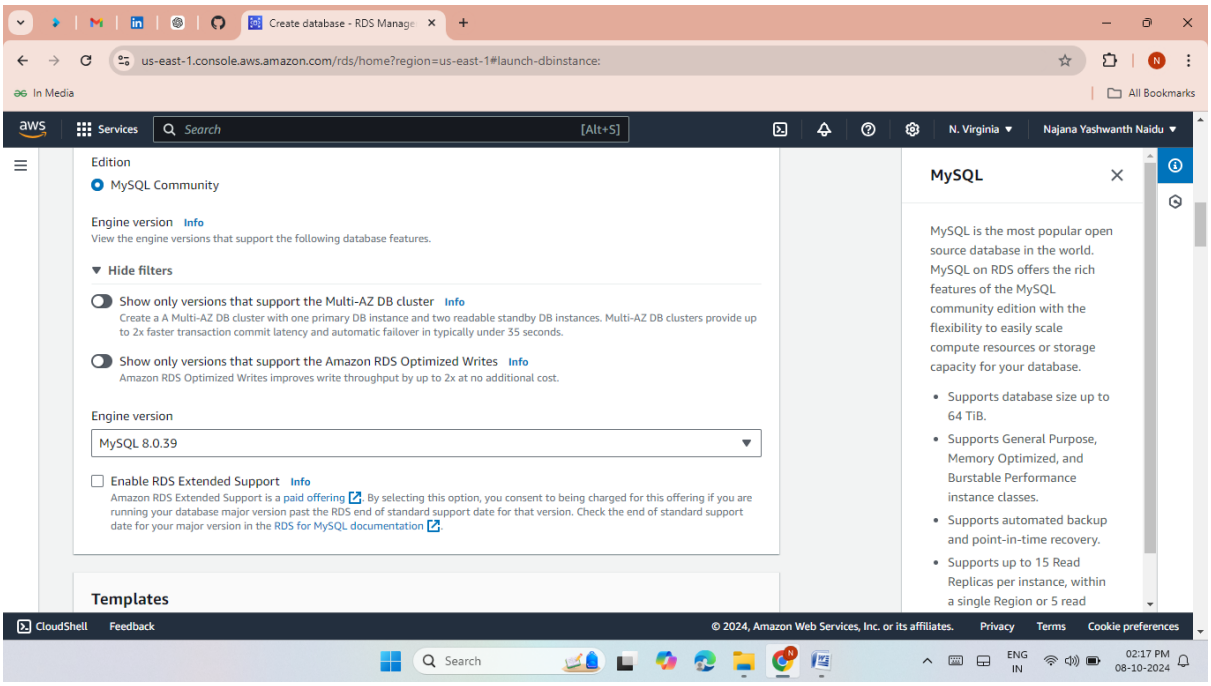
- Choose a database creation method (standard creation).



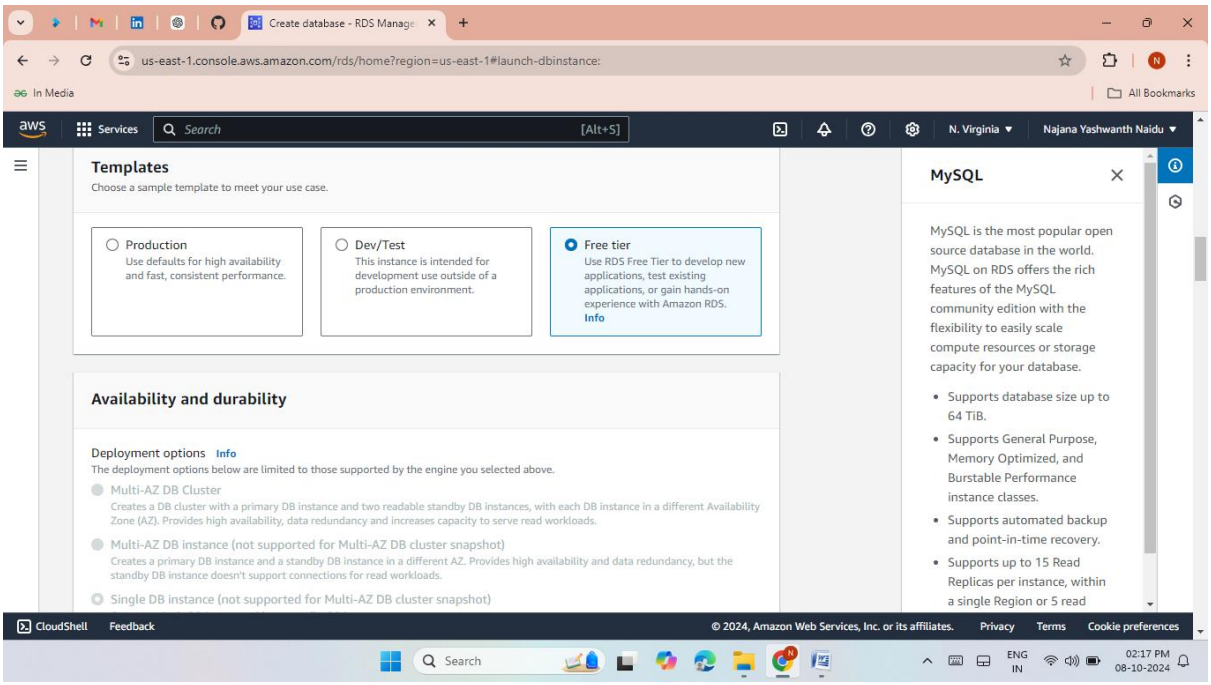
- Choose a database engine option(MYSQL).



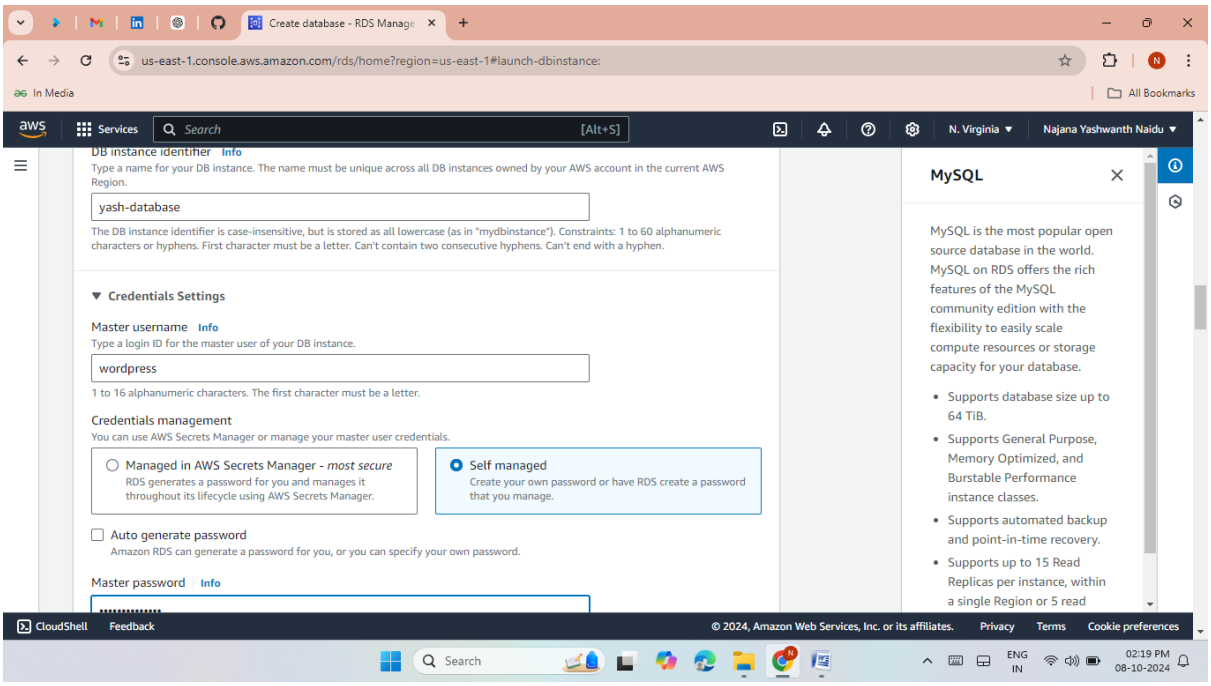
- Select the Engine version.



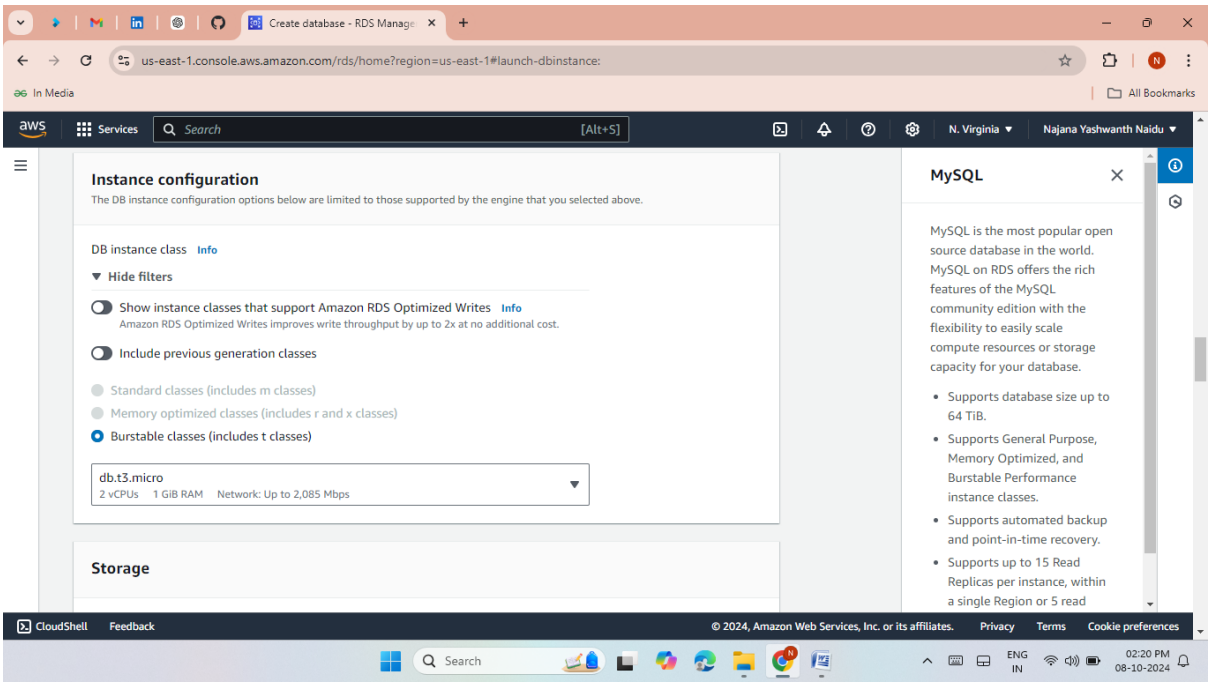
- Select the template as free tire.



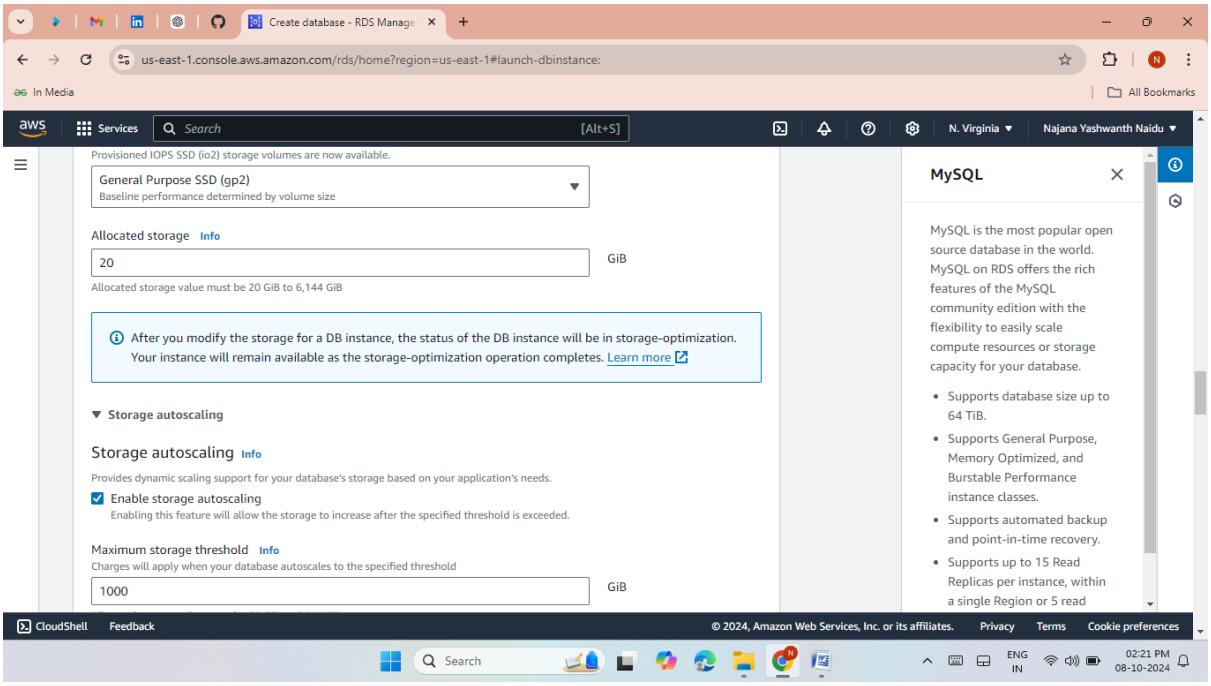
- Enter the Master username and password for database access.



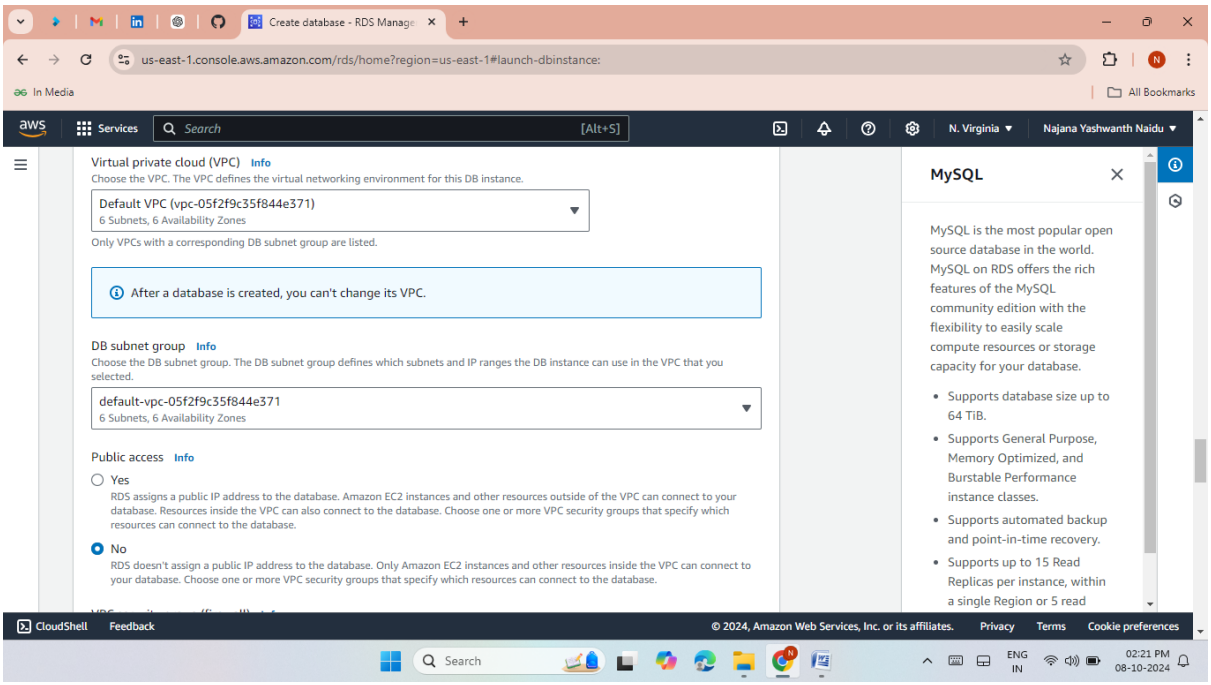
- Select db.t3.micro.



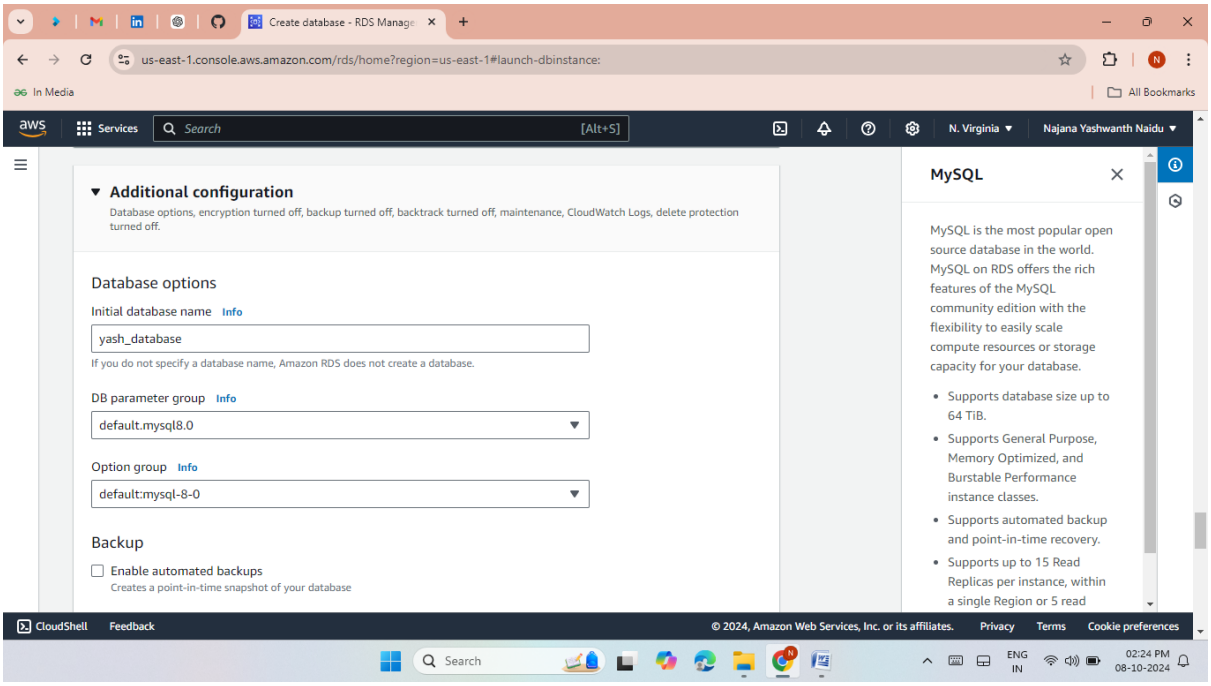
- Now select the storage type as general purpose SSD(gp2) and Enter the storage values as max(1000gb) and min(20gb).



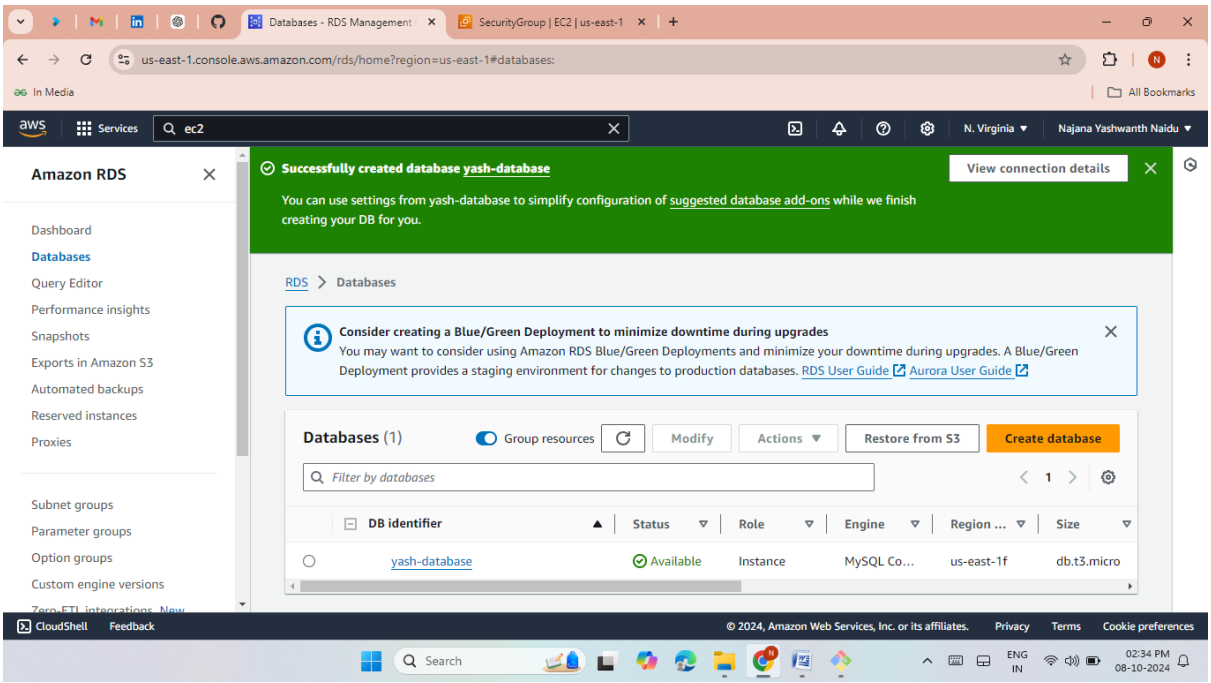
- Now select default VPC and it automatically select the Database subnet group.



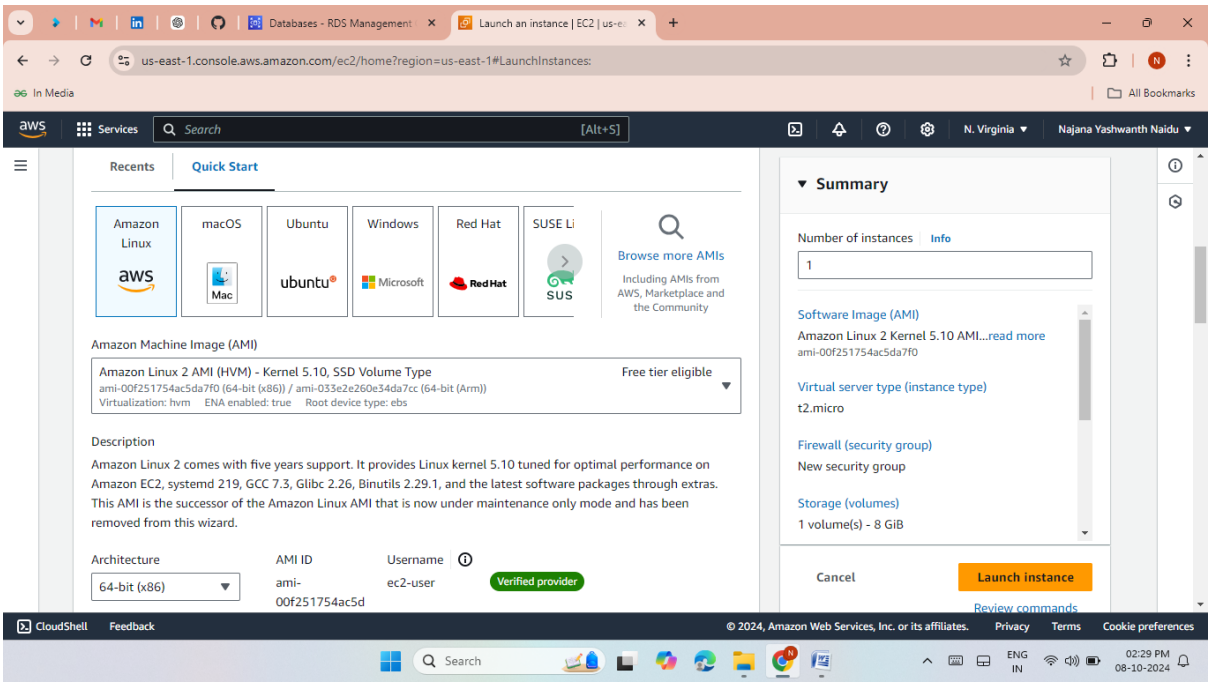
- Now give the name of the database which you give at the stage of DB instance identifier enter same name here.



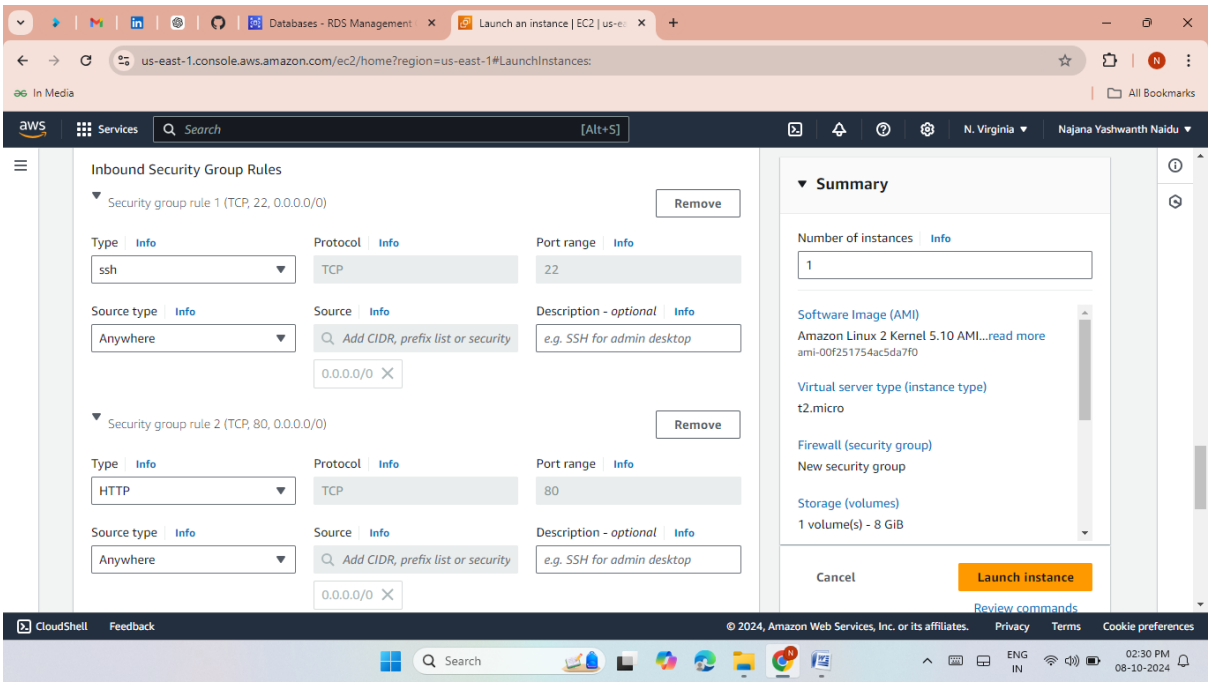
- Now click on create database button and it will create the MYSQL Database.



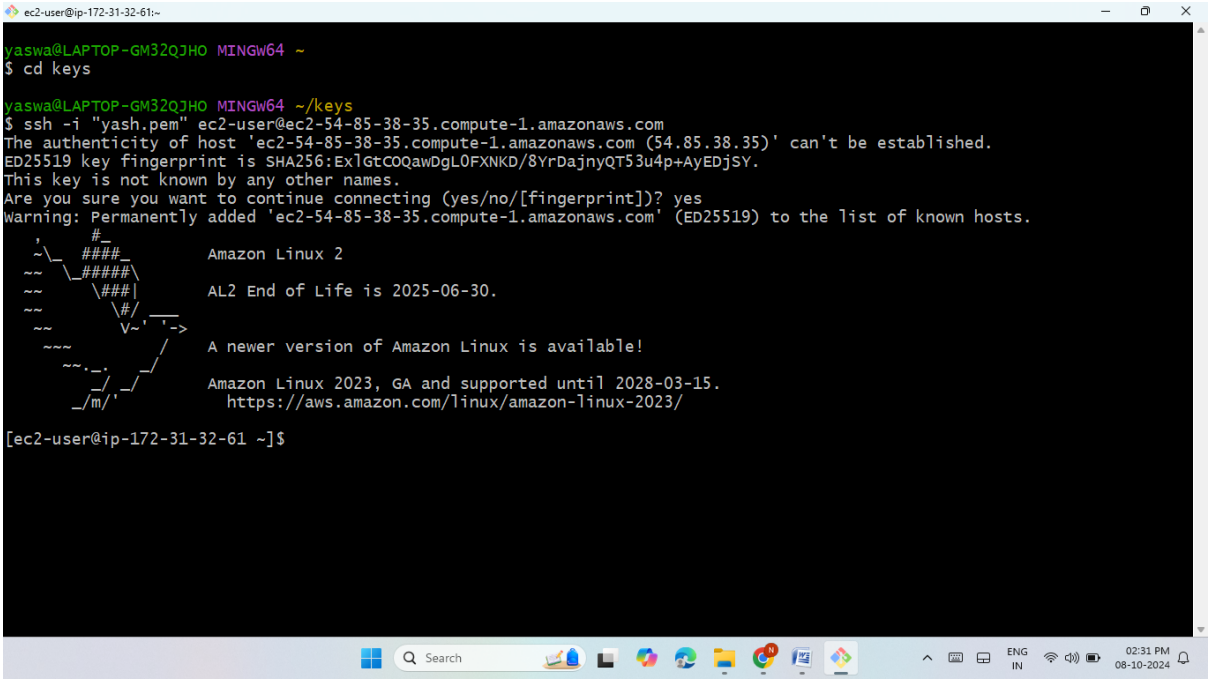
➢ Launch an instance with amazon-linux.



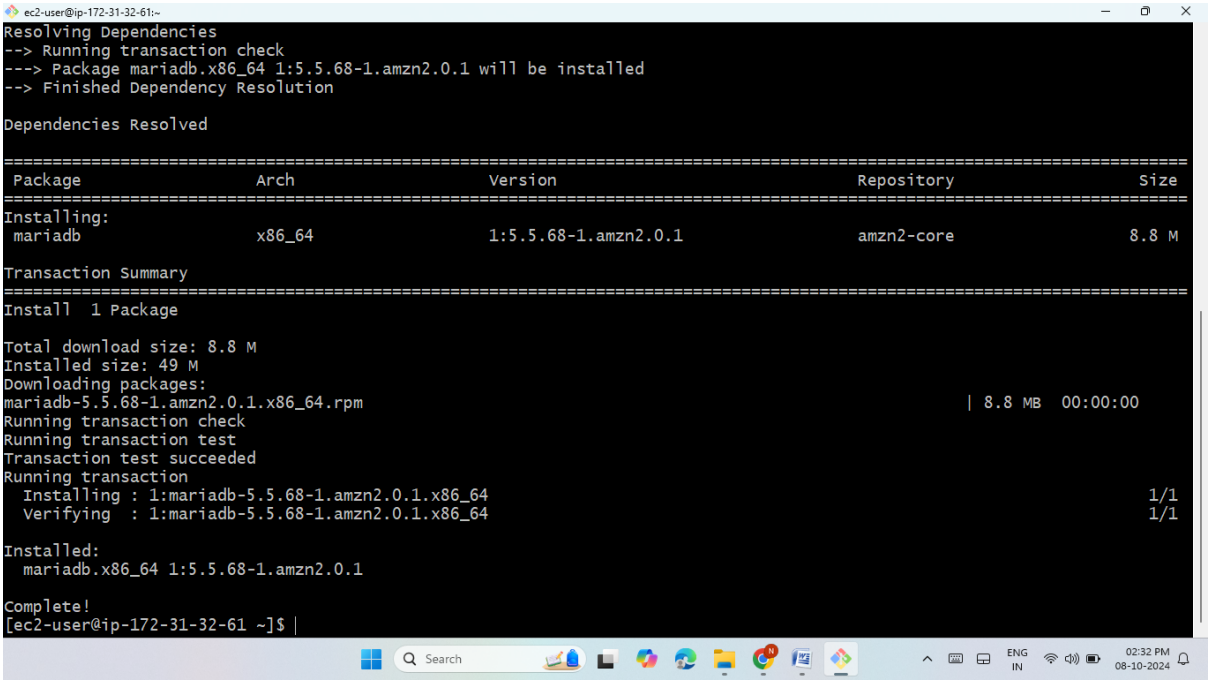
➢ Select the Security group with SSH(22) and HTTP(80).



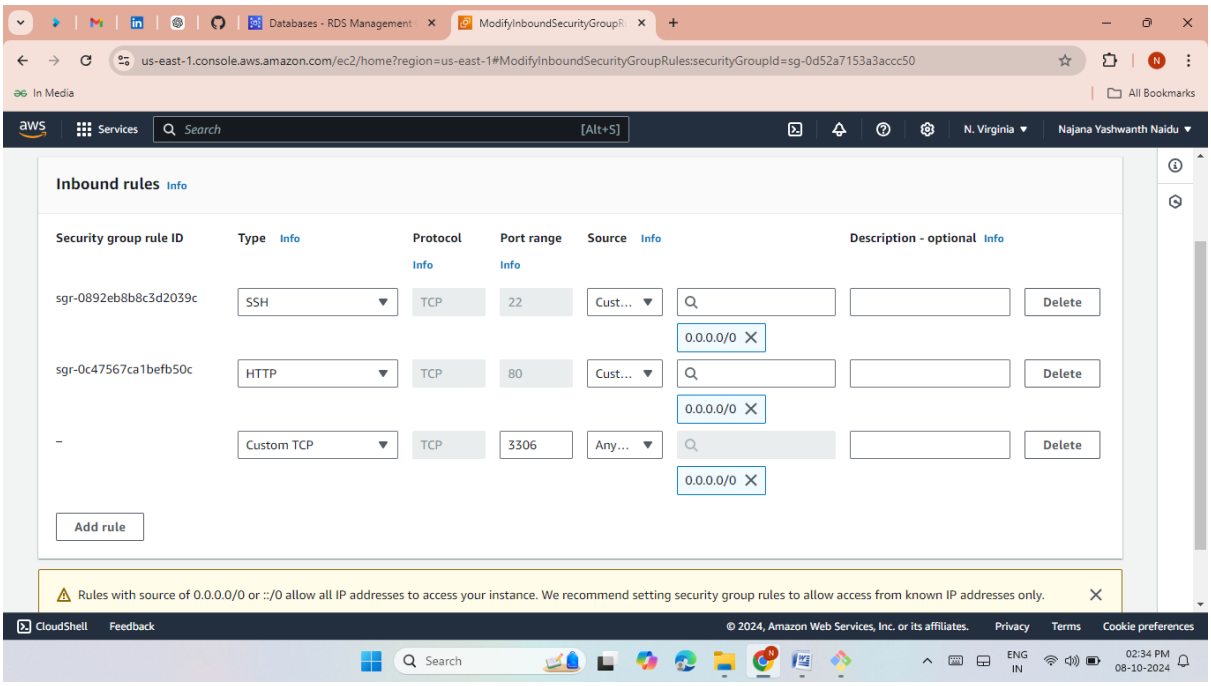
- Now Connect the server to locally (GitBash).



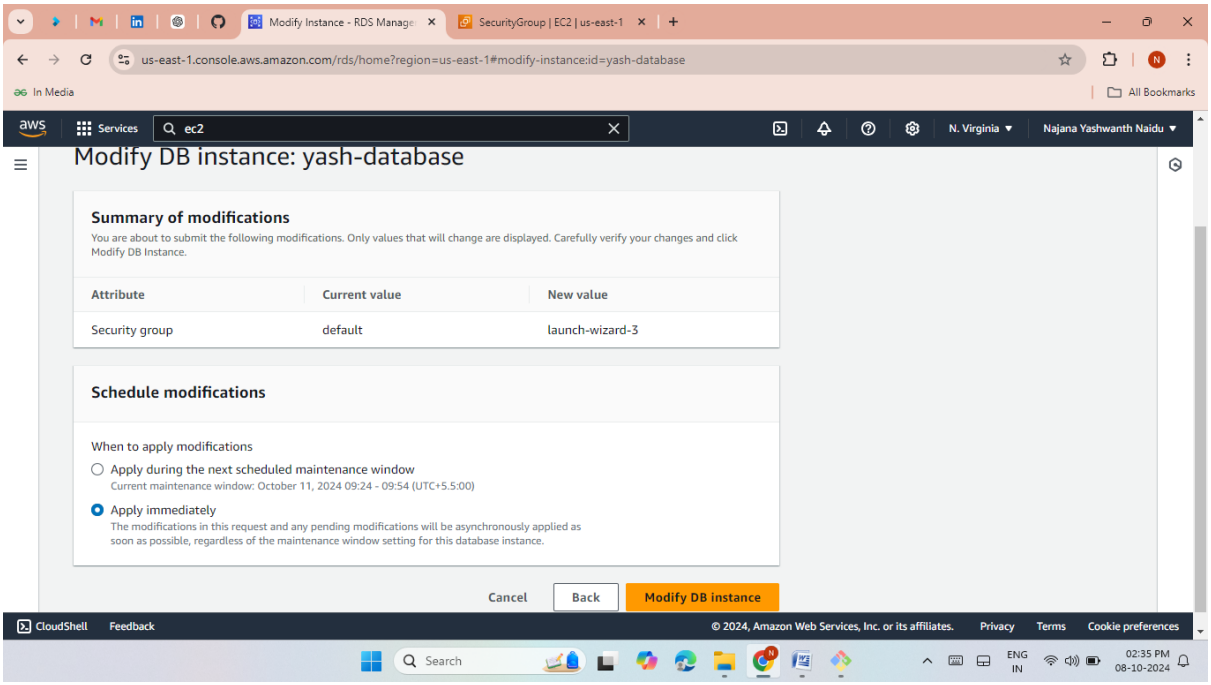
- Now update the linux version by using command as **sudo yum -y update** and install the MYSQL by using the command as **sudo yum –y install mysql**.



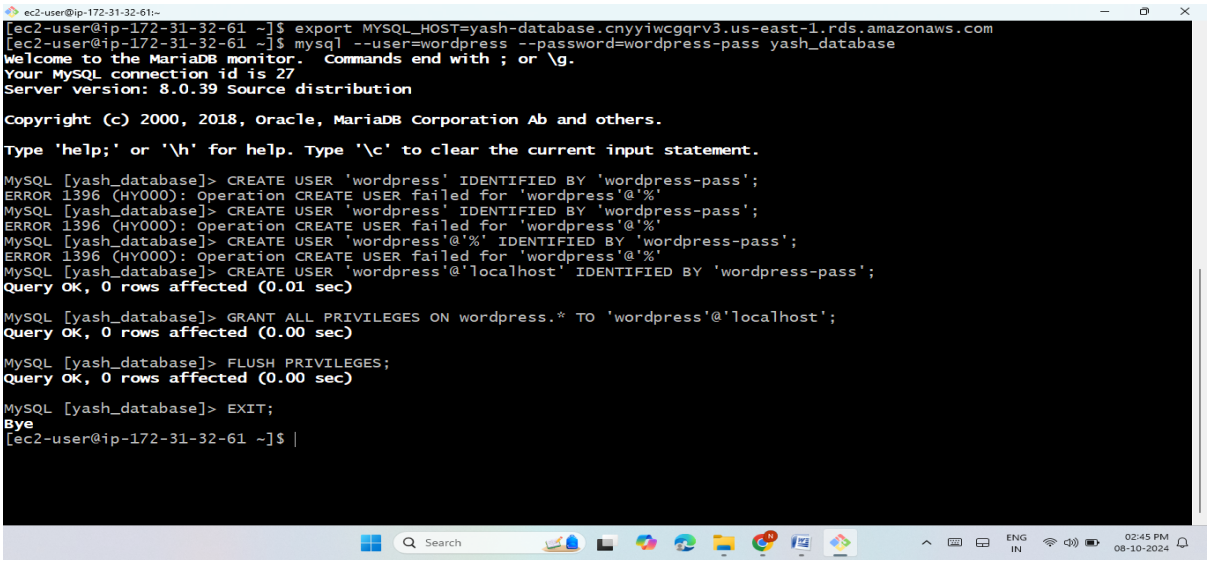
- Now go to Security Groups then Select Inbound rules and add 3306 port for MYSQL.



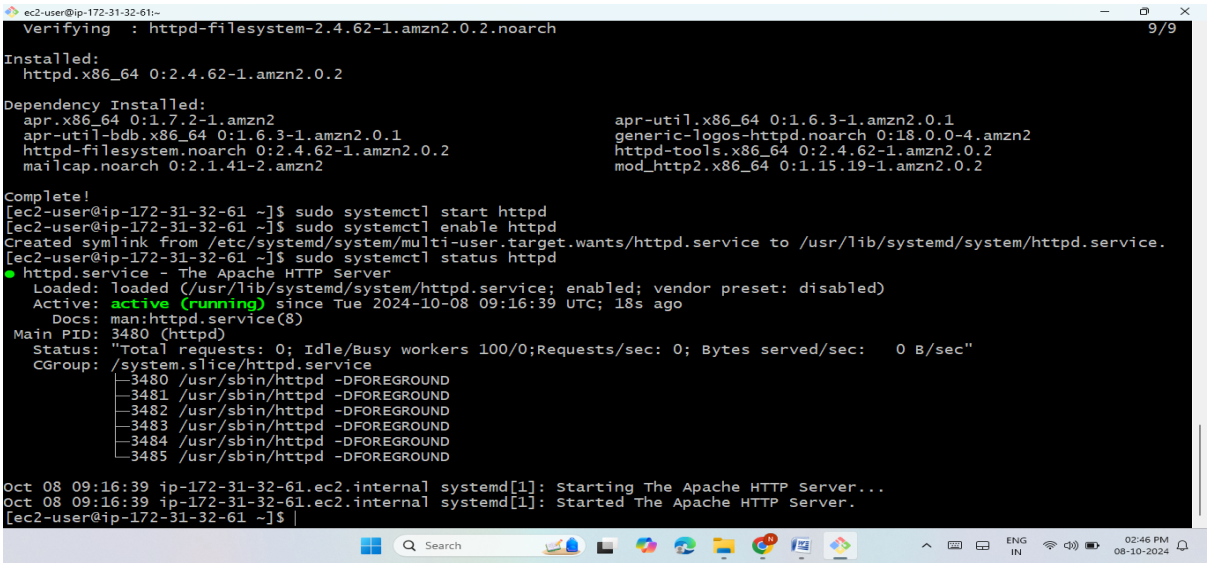
- Now Select the Database and click on modify then set the EC2 Security group to this Database.
- Click on modify DB instance.



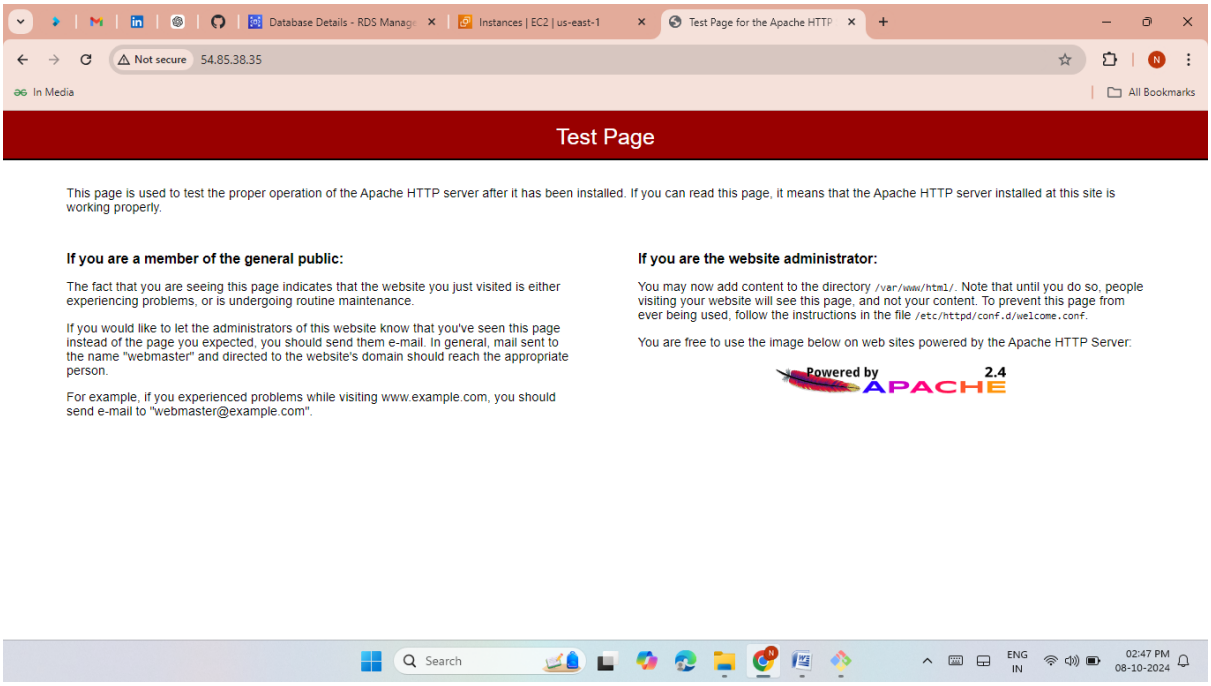
- Now access the MYSQL Database by using the command as **export MYSQL_HOST=<endpoint address>** for that go inside the created database and go to the endpoint and select and copy the endpoint address.
- Now give the credential of database by giving a command as "**mysql --user=<username> --password=<password> database-name"(yash_database).**
- Now create a database user for your wordpress application and give it permission to access the wordpress database. By using this command as.
    - **CREATE USER 'wordpress'@'localhost' IDENTIFIED BY 'wordpress-pass';**
    - **GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpress'@'localhost';**
    - **FLUSH PRIVILEGES;**
    - **EXIT;**



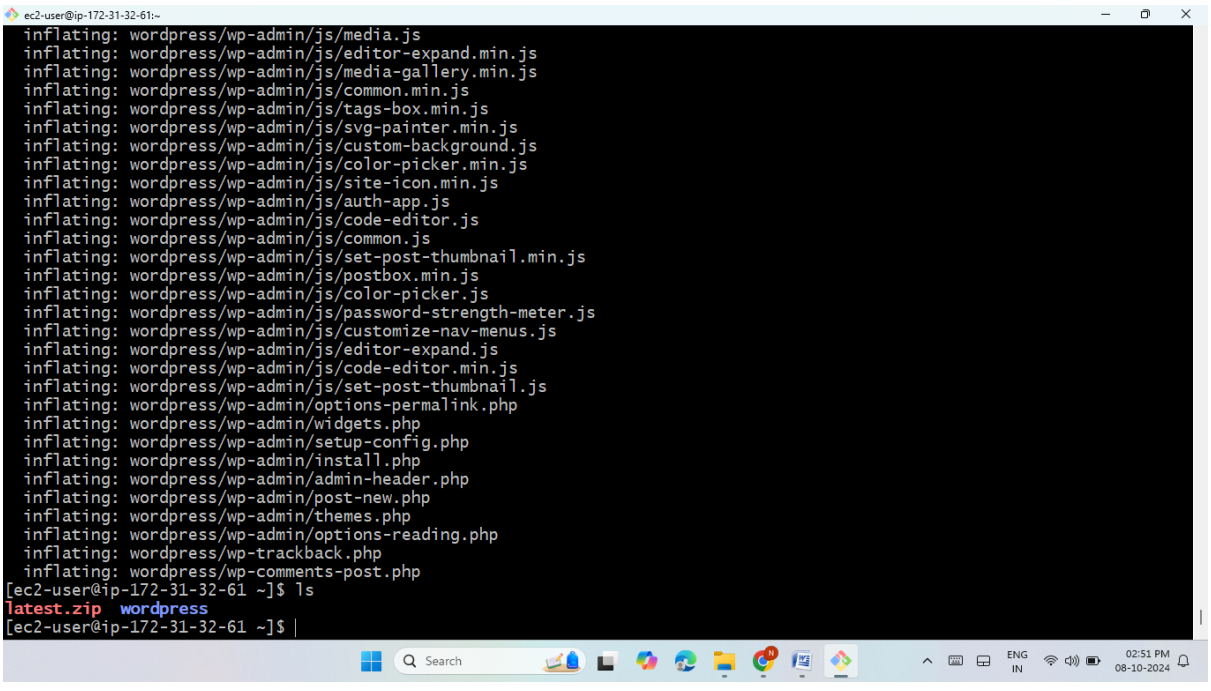
- Now Install HTTPD and then start and enable the HTTPD.



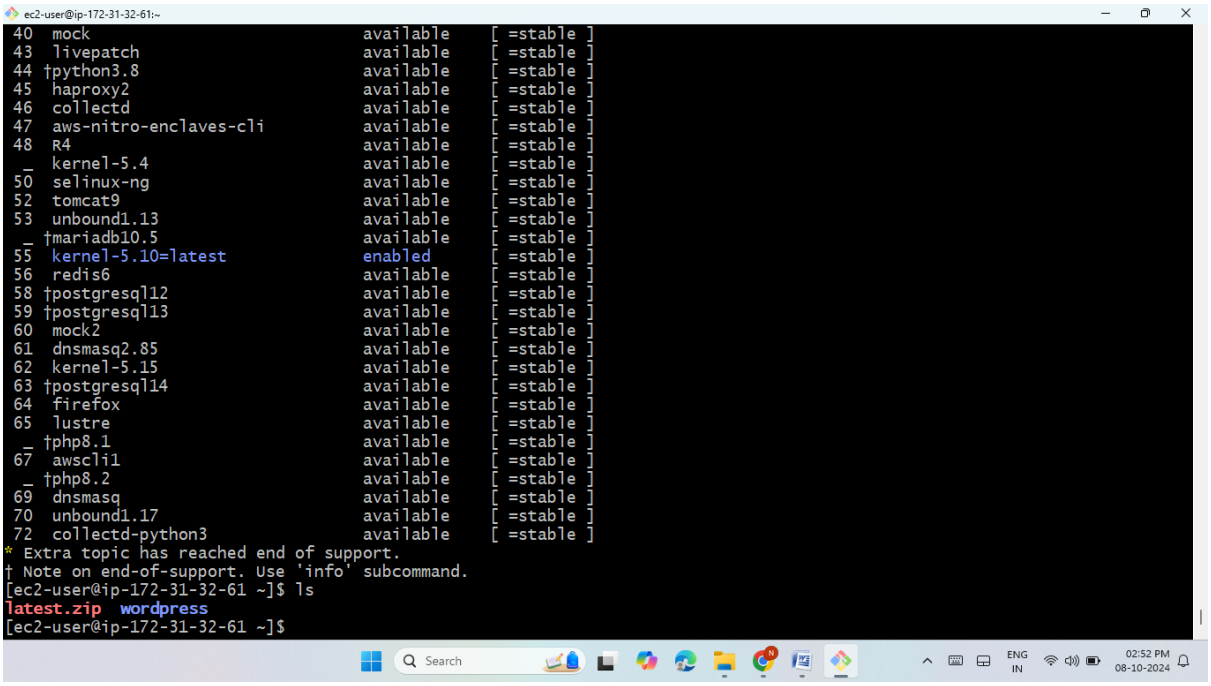
- Now go to EC2 instance and copy Public IP and Past it on Google browser it and check the official page of httpd is displays or not.



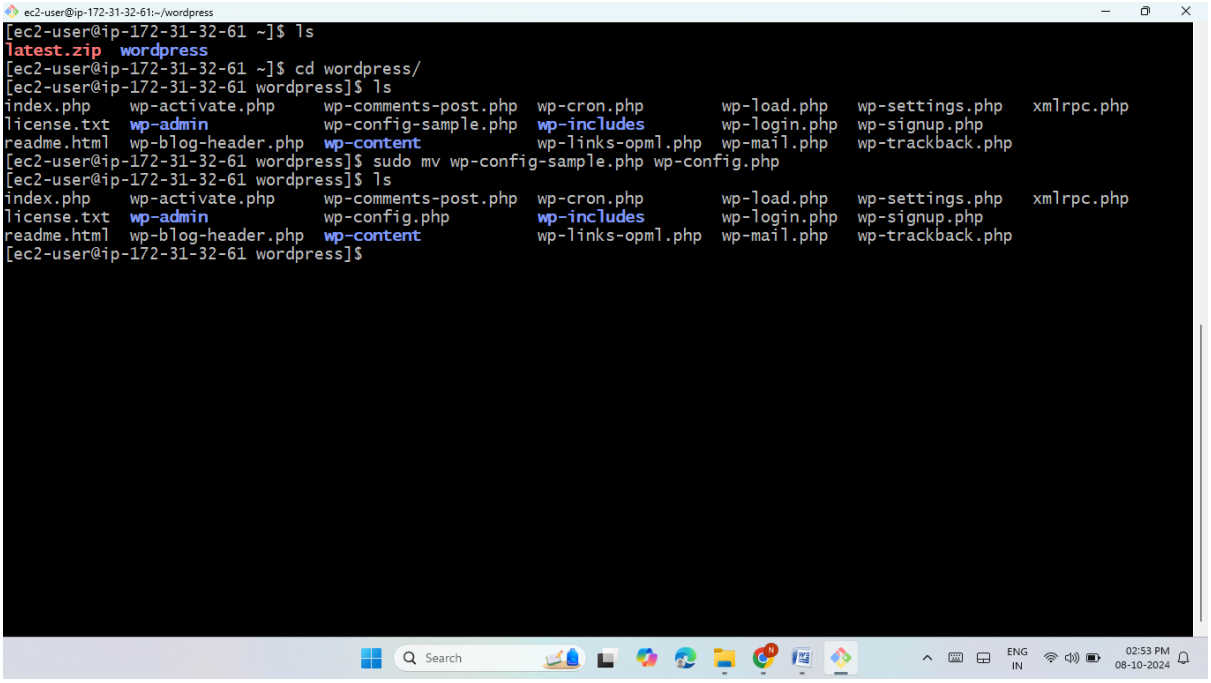
- Now go to browser and search as download wordpress and click on proper link and select ad copy the address link of wordpress download file and past that along with wget command in git bash as **“wget< >”**.
- It gives the zip file to unzip that file by using a command as “**unzip<zip file>**”.



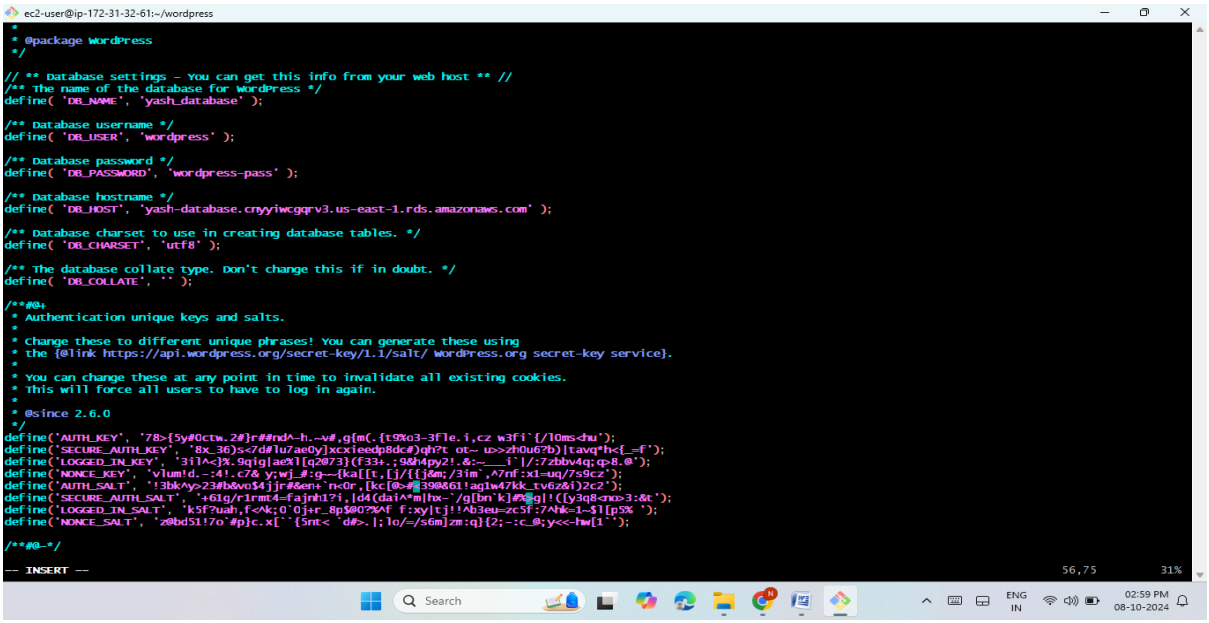
- To run wordpress web application, you have to install run time of wordpress web application as php laungage with the following command **sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2**



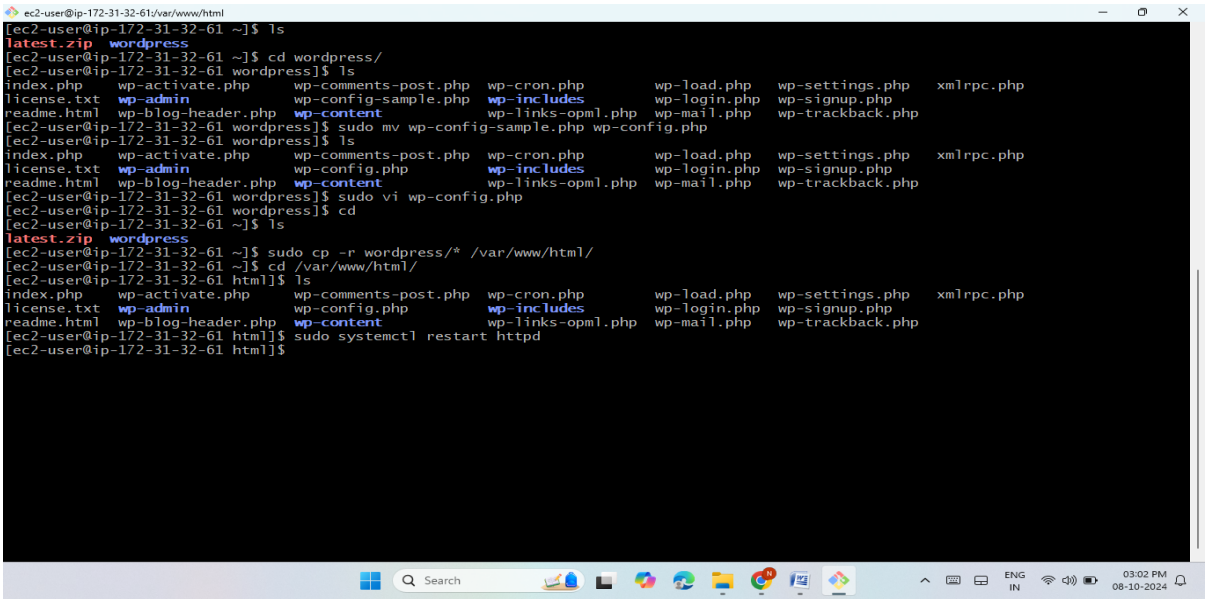
- Now go inside the unzip directory by using command as "**cd <unzip directory>**" and change the wordpress configuration file by giving command as "**sudo mv wp-config-sample.php wp-config.php**".



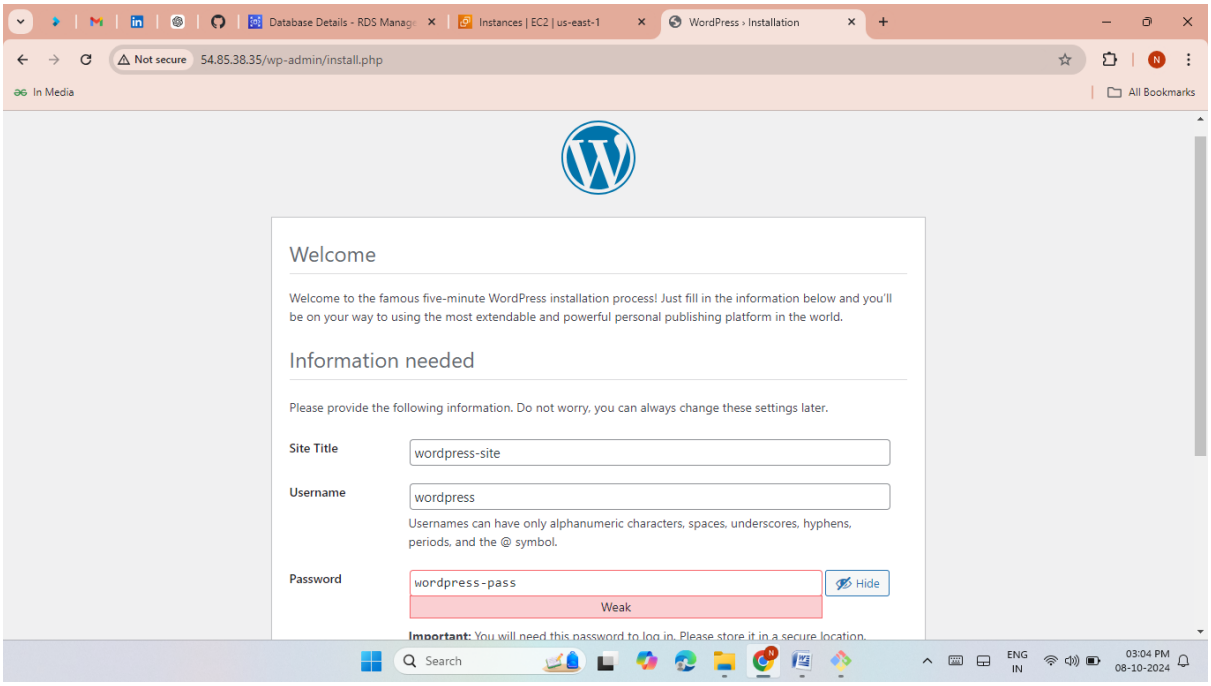
- Now do some configurations in wordpress configurations files as by giving database name, username, password and hostname for that execute a command as "**sudo vi wp-config.php**".
- Along with this change the wordpress security key to get the secret key browse in Google generate word press secret key.

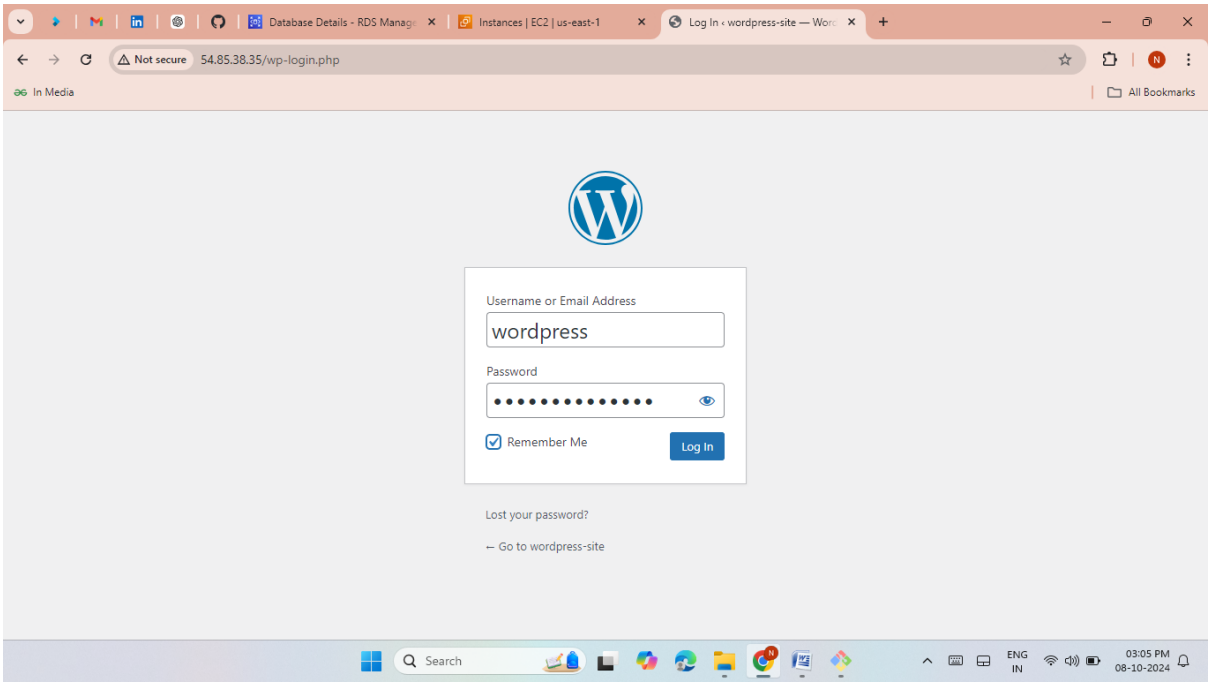


- Now copy this wordpress directory to the doucument root directory to host your web application of wordpress by giving a command as
  - **sudo cp –r <unzip file or wordpress directory>/* /var/www/html/**
  - **cd /var/www/html/**
- Restart the httpd **(sudo systemctl restart httpd").**

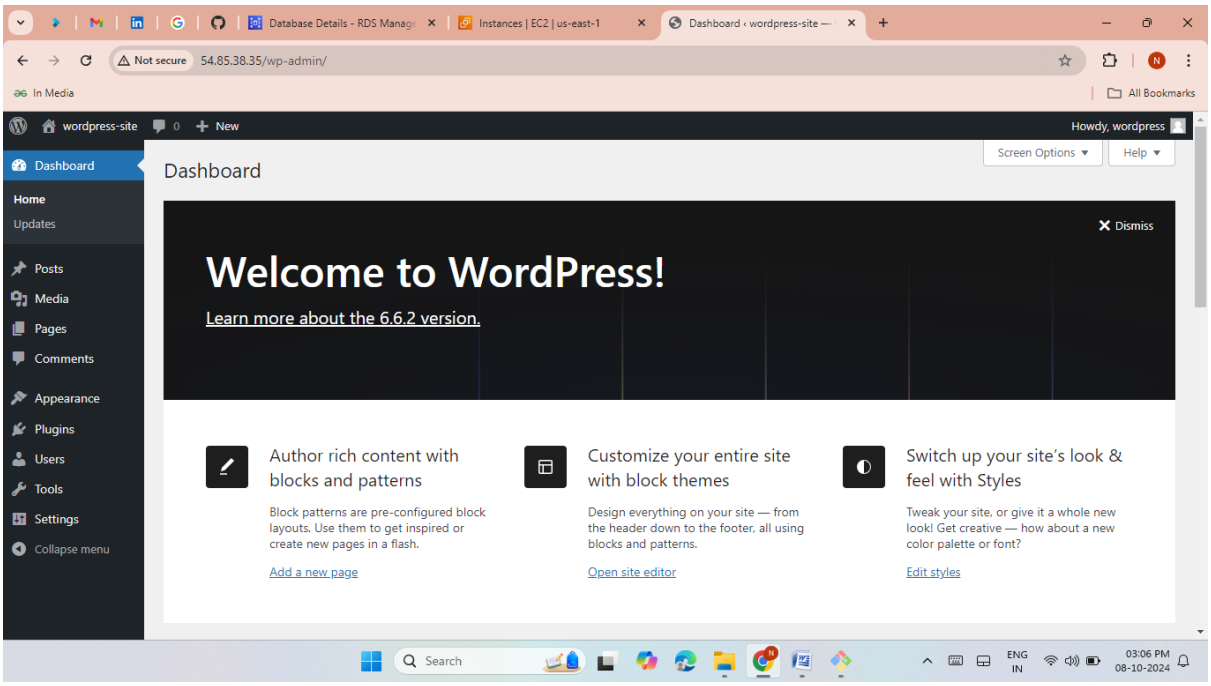


- Now go to EC2 instance and copy public IP and paste it on google browser and check the official page of wordpress is displays.





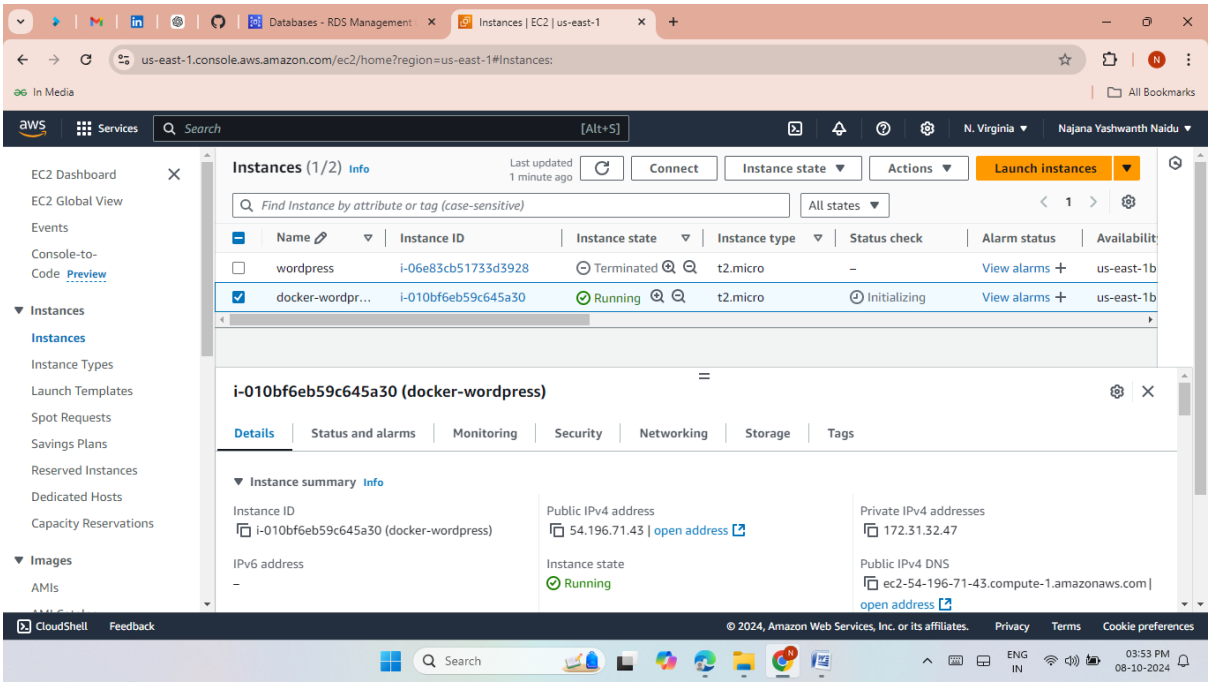
- Wordpress website was launched successfully .



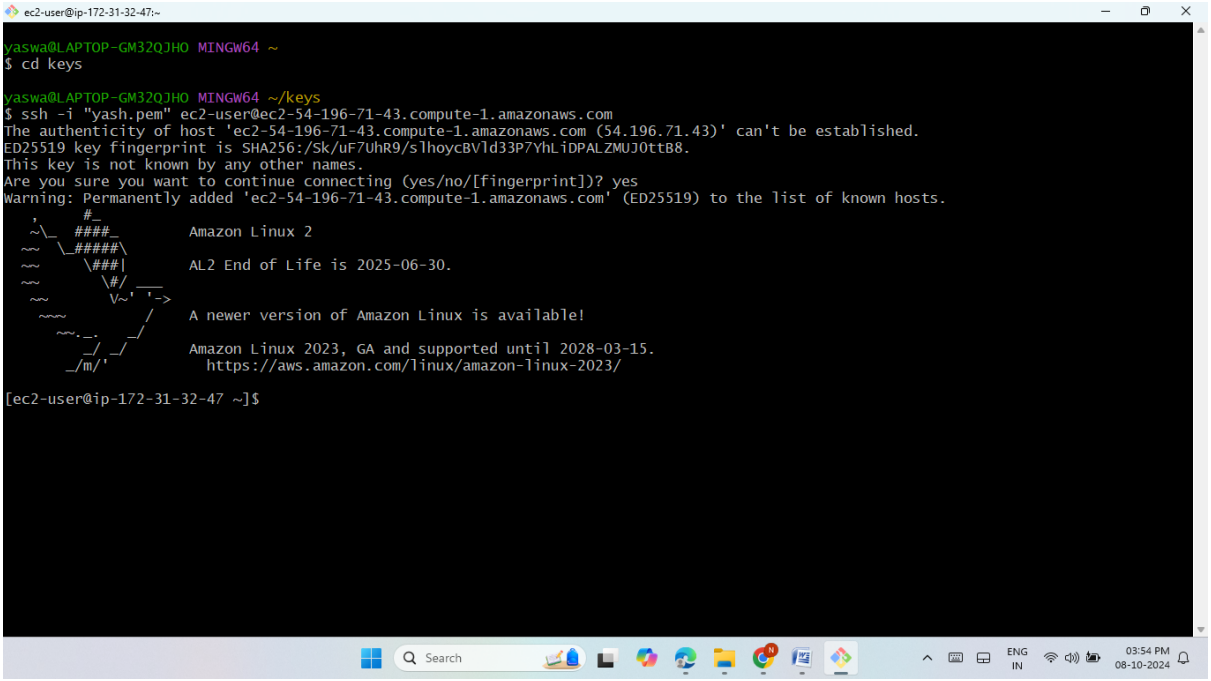
METHOD 2

Deploy WordPress web application by using docker compose file

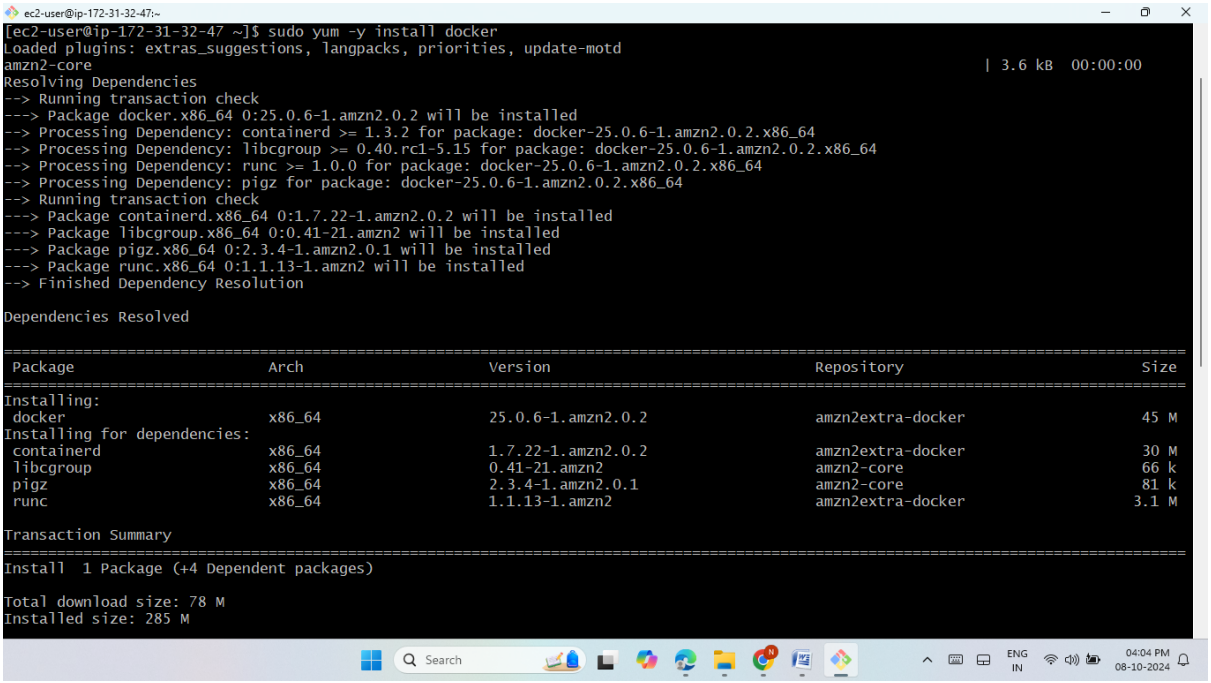
- Launch an EC2 Instnace.



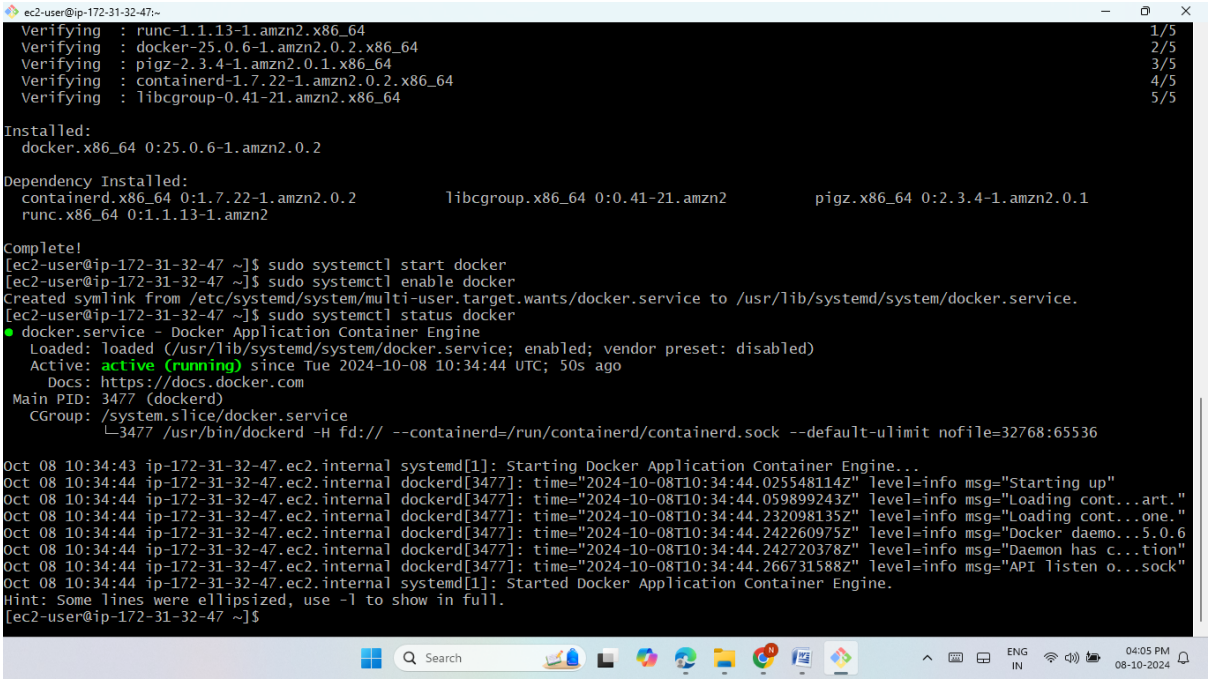
➢ Now Connect the server locally.



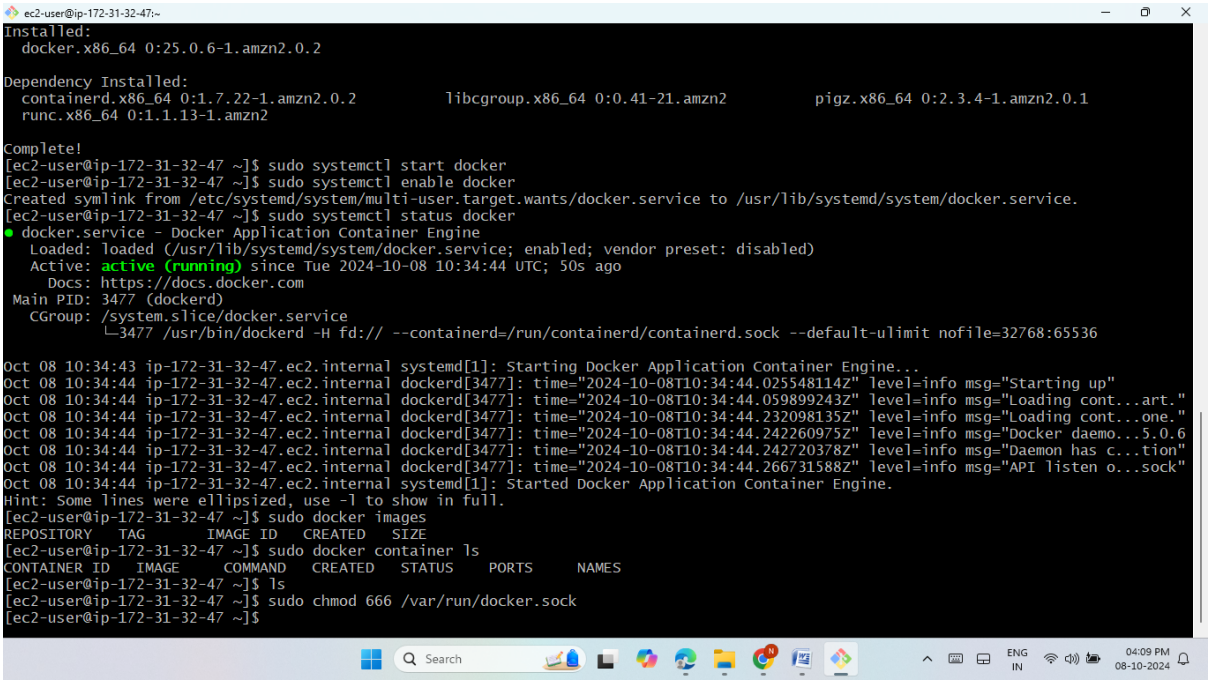
➢ Now install Docker with the command **sudo yum install –y docker**



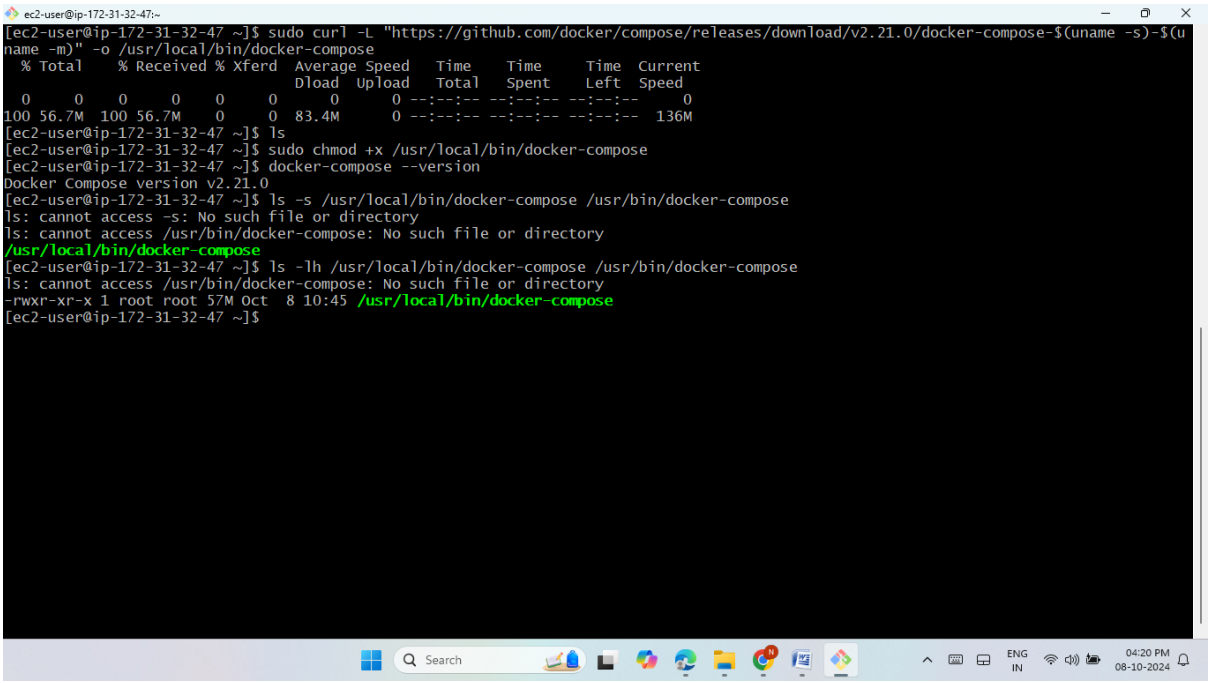
- Then start and enable the Docker.



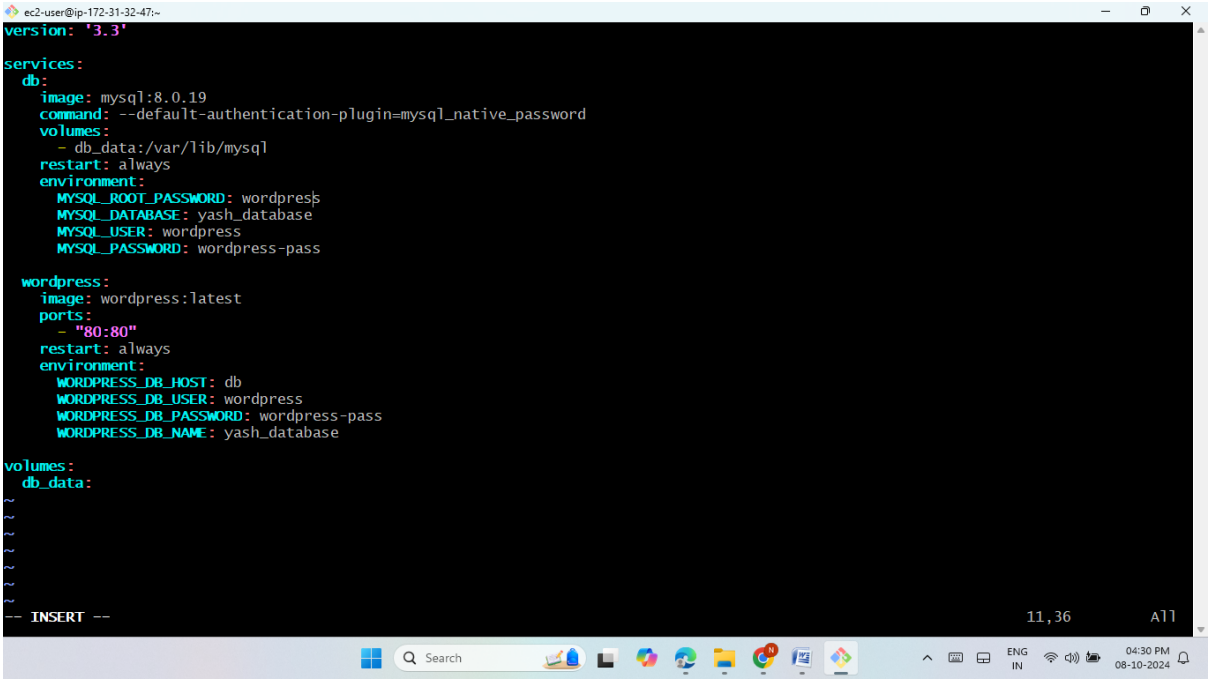
- Now give permissions to the docker with the command.
- **sudo chmod 666 /var/run/docker.sock**



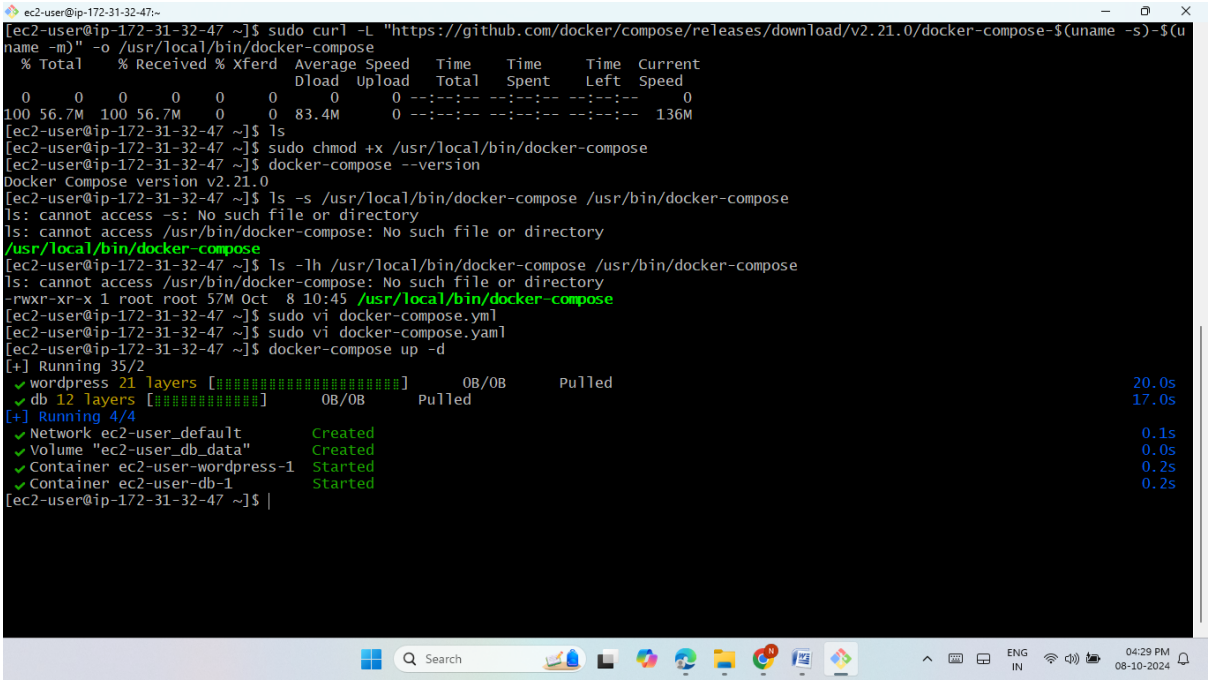
- Now install the Docker composer by using Google.
- Give permissions to the Docker compose with the command
- **sudo chmod +x /usr/local/bin/docker-compose**
- Now create the symbolic link by using command as **ln –s /usr/local/bin/docker-compose /usr/bin/docker-compose** (it works on backend to start).



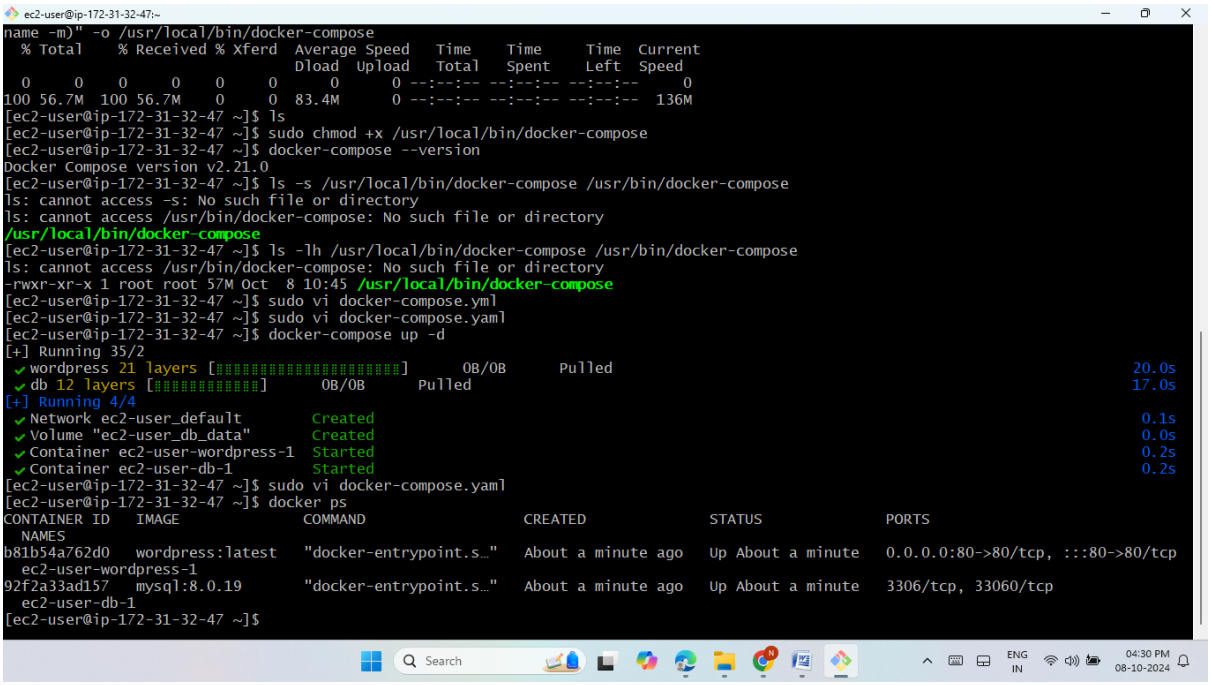
- Now to create a docker-compose.yml file in vi mode as **sudo vi docker-compose.yml**



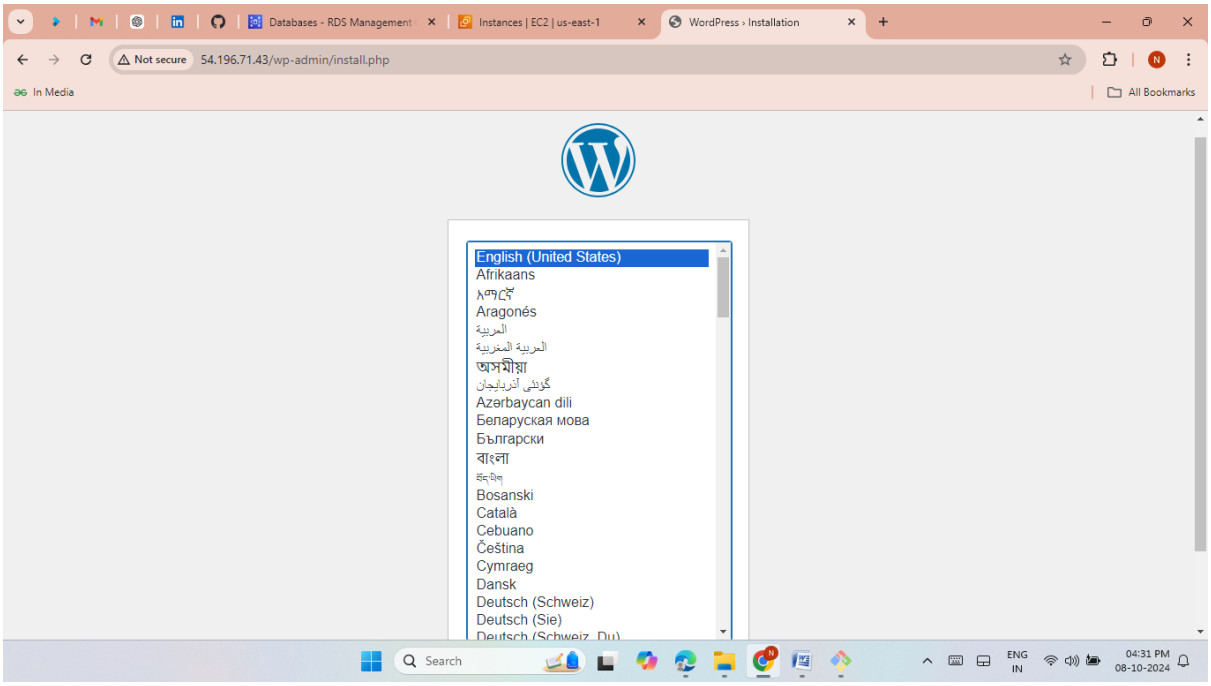
- Now to pull the images of MYSQL and WORDPRESS we have to execute this created **docker-compose.Yml** file by using the command as <**docker –compose up –d>.**

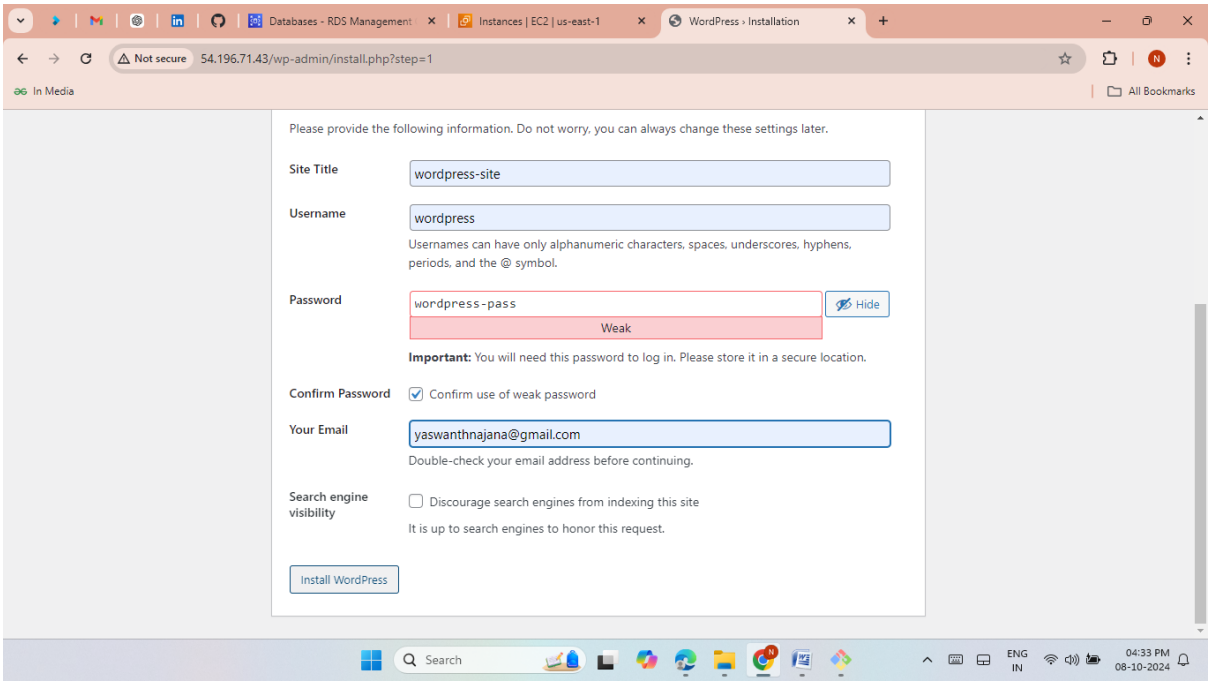


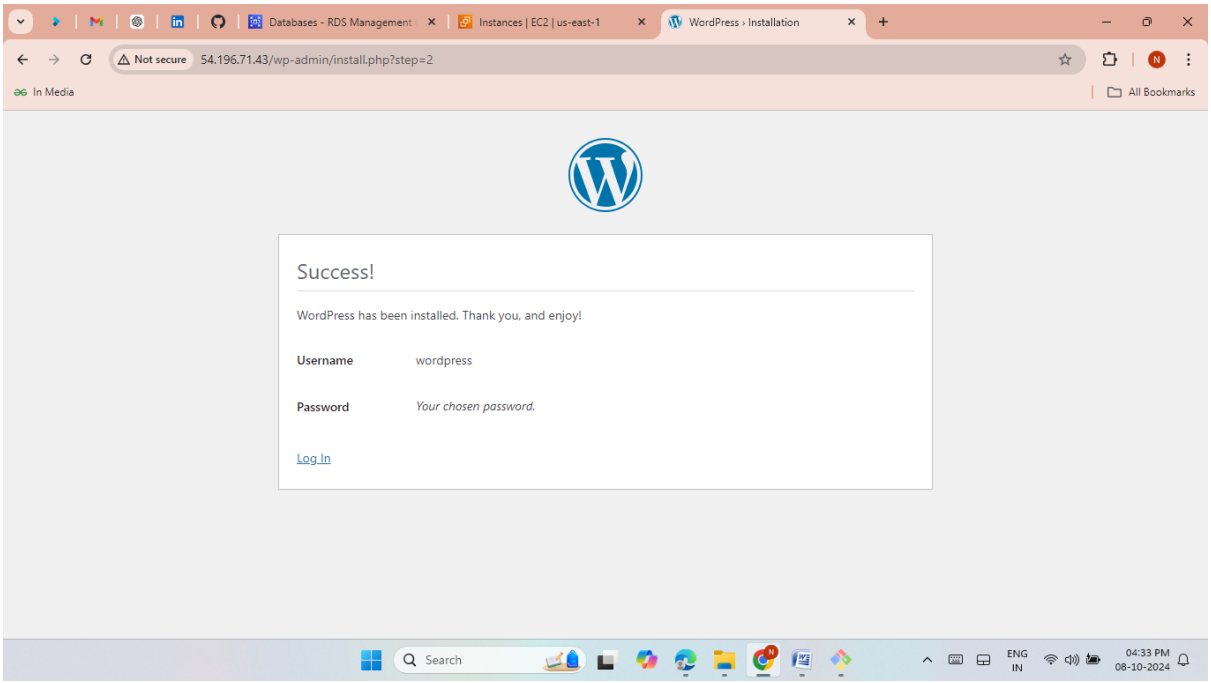
- Now see the containers of the pulled images of MYSQL and WORDPRESS by using a command as **docker ps**

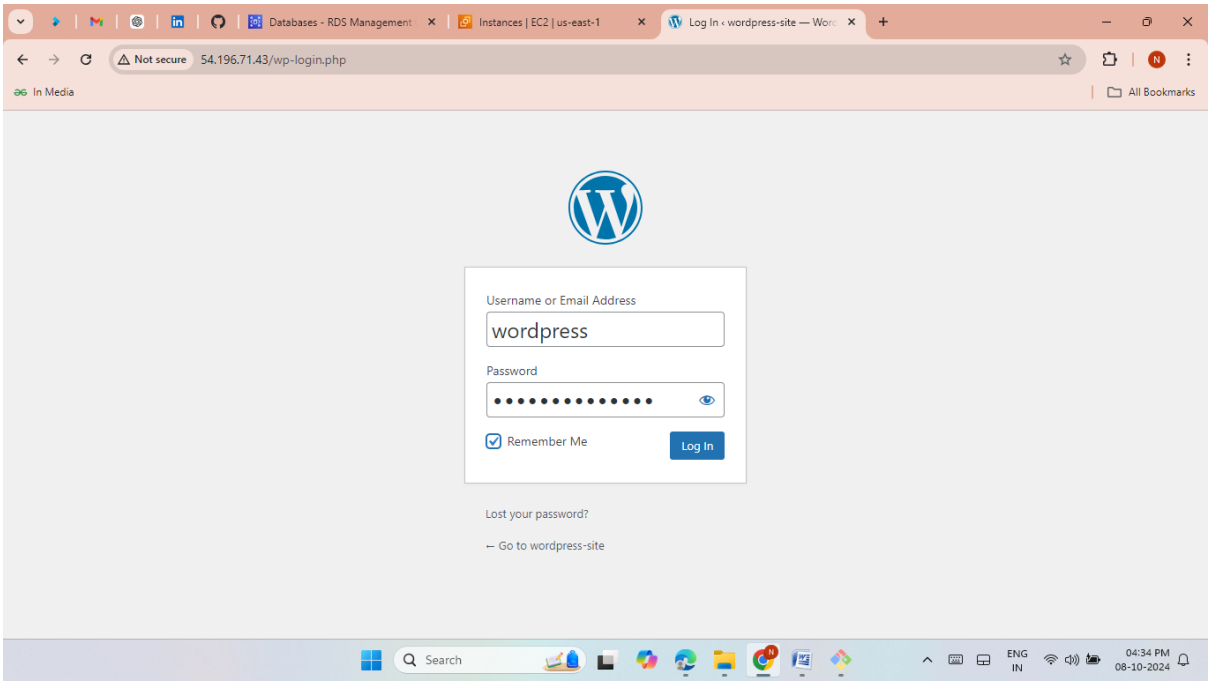


- Now copy that static Public IP and past it in Google browser then select the language and click on continue.

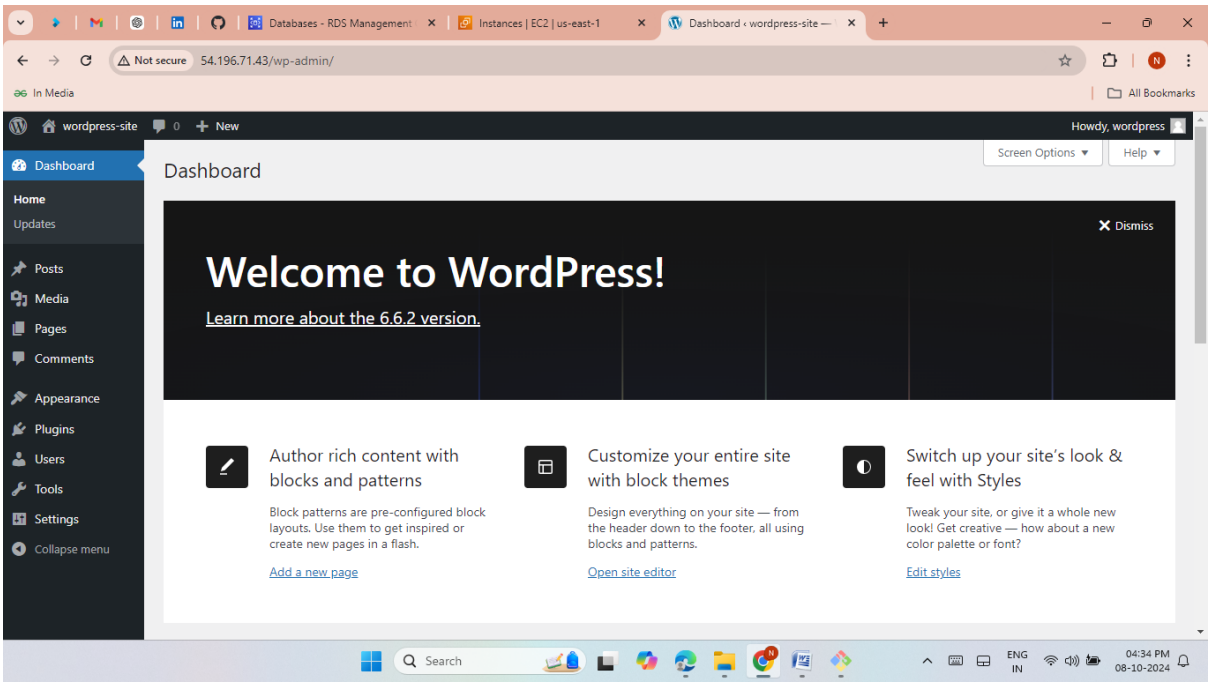




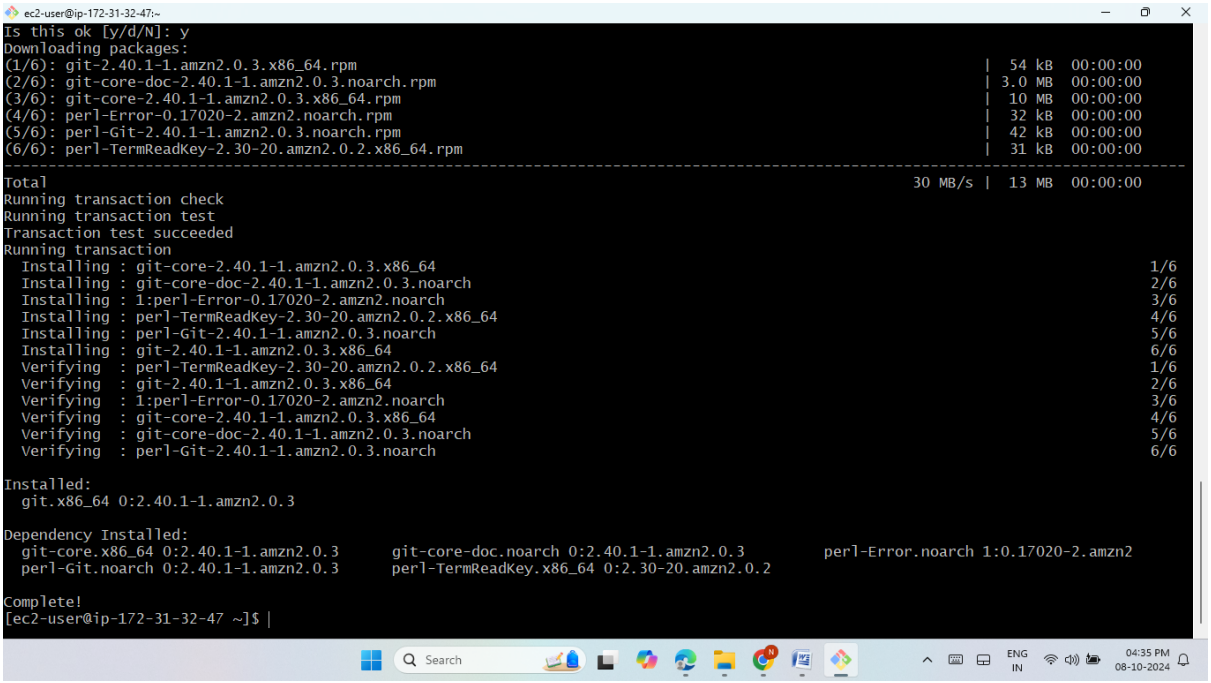

Databases - RDS Management
Instances | EC2 | us-east-1
WordPress › Installation
Not secure
54.196.71.43/wp-admin/install.php?step=2
In Media
All Bookmarks
Success!
WordPress has been installed. Thank you, and enjoy!
Username
wordpress
Password
Your chosen password.
Log In
Search
ENG
IN
04:33 PM
08-10-2024

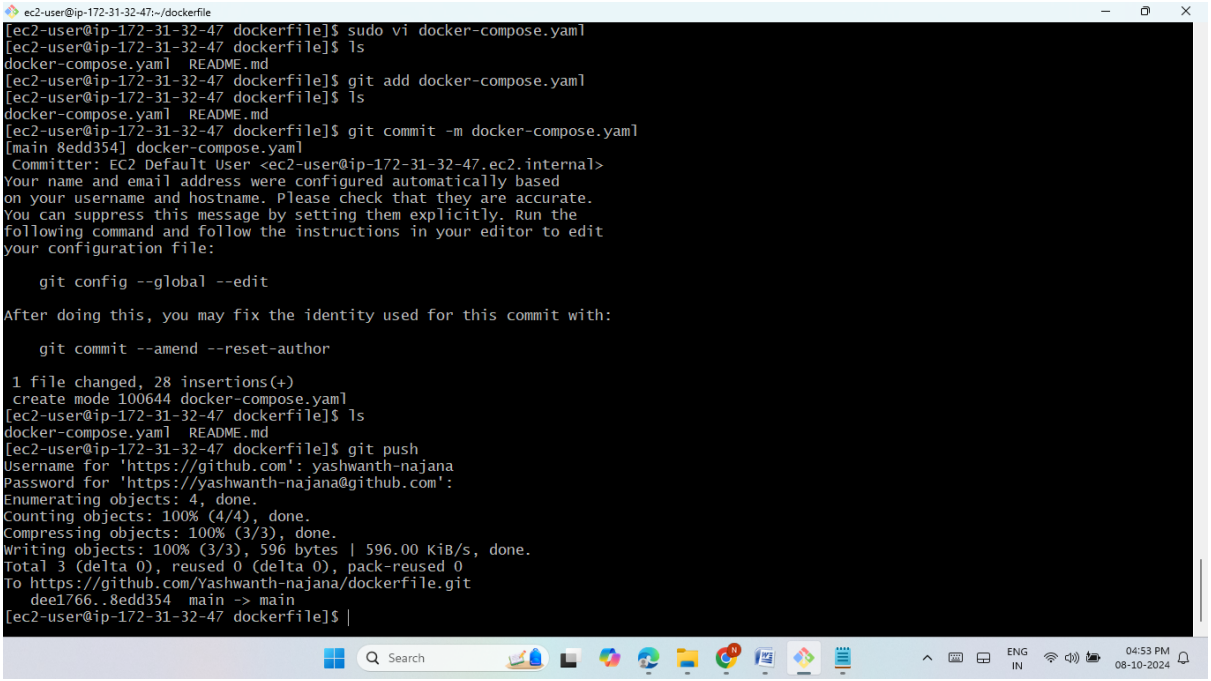

Databases - RDS Management
Instances | EC2 | us-east-1
Log In ‹ wordpress-site — Word
Not secure
54.196.71.43/wp-login.php
In Media
All Bookmarks
Username or Email Address
wordpress
Password
Remember Me
Log In
Lost your password?
← Go to wordpress-site
Search
ENG
IN
04:34 PM
08-10-2024

- Here the wordpress is successfully launched.

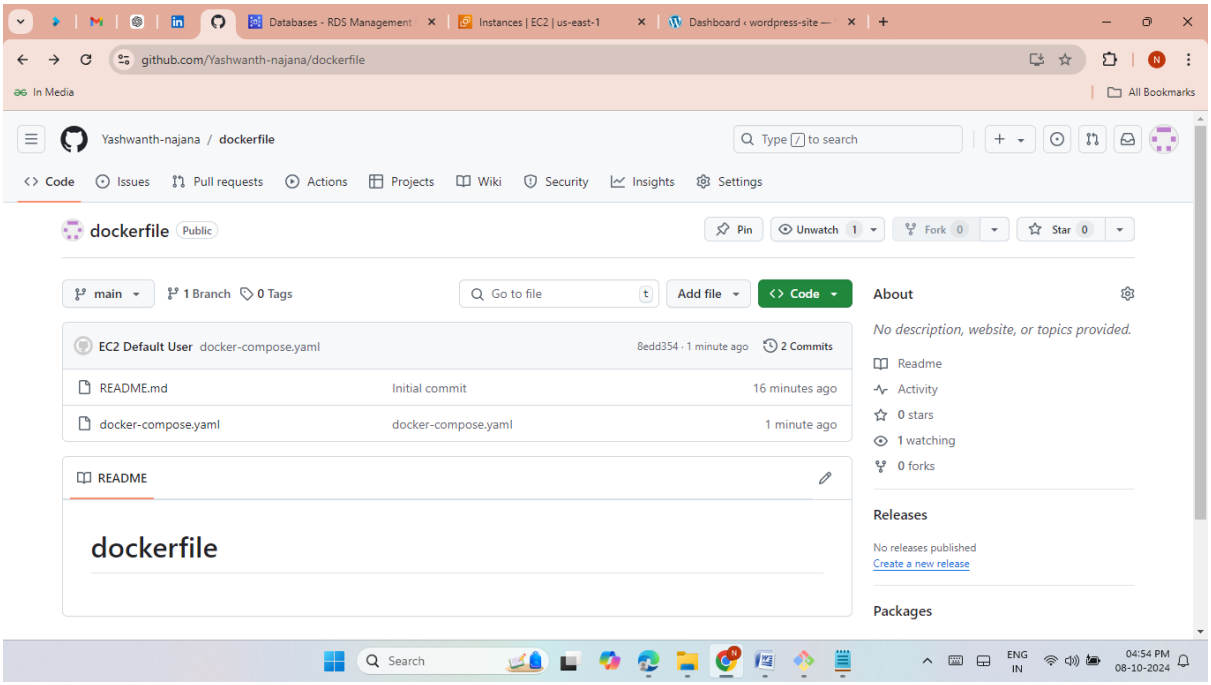


- Now install git then push the docker-compose file in github.





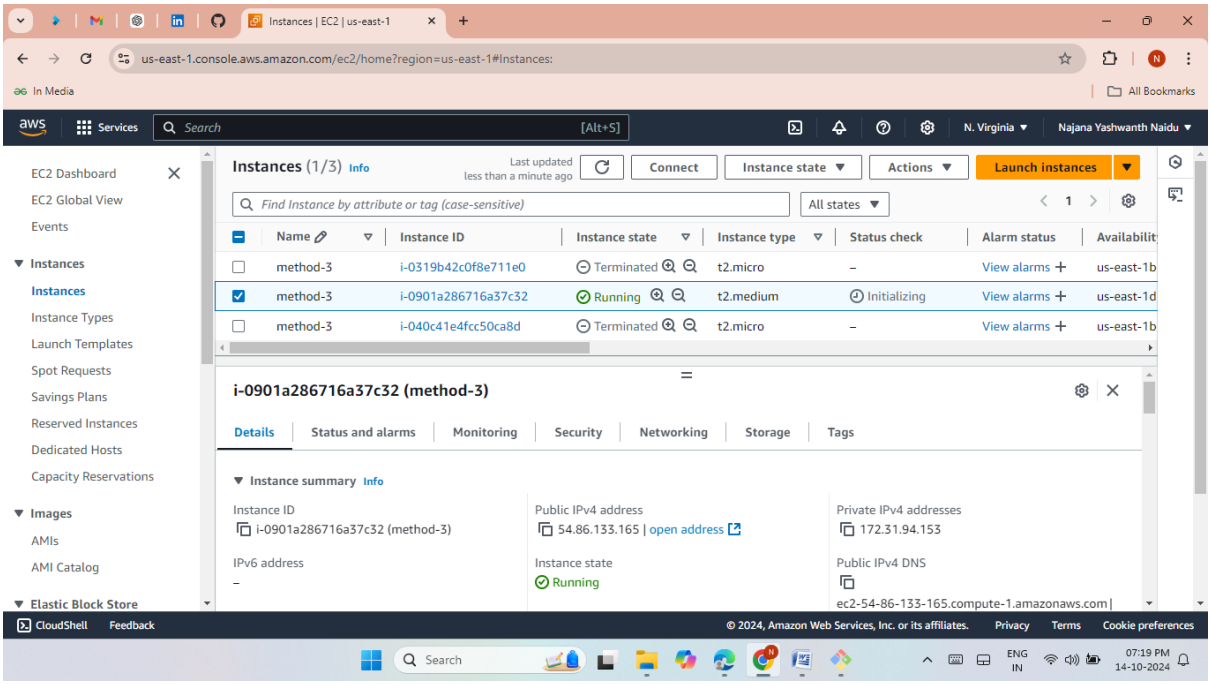
- Here the docker-compose file is pushed.



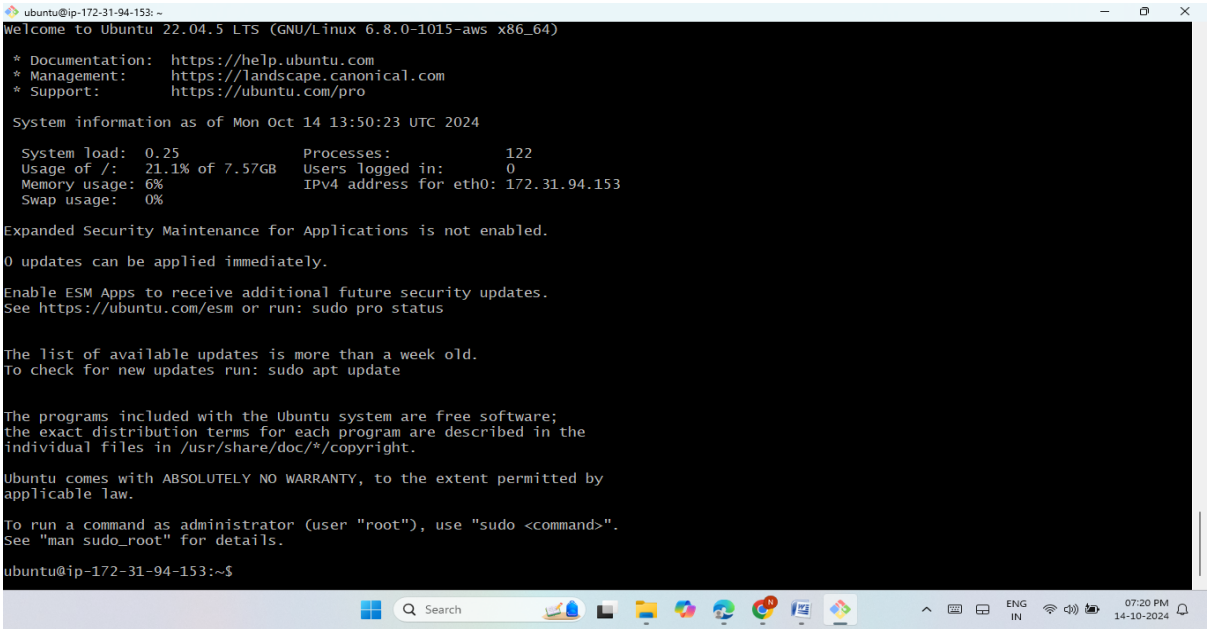
METHOD 3

Deploy WordPress web application by using git and jenkins

- Launch EC2 instance with Jenkins port number(8080).
- Here I have choose ubuntu.



- Now connect the remote server to local server.



- Now install Jenkins with the following commands

Sudo apt update

sudo apt install openjdk-17-jre-headless

**sudo wget -O /usr/share/keyrings/jenkins-keyring.asc **

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

**echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" **

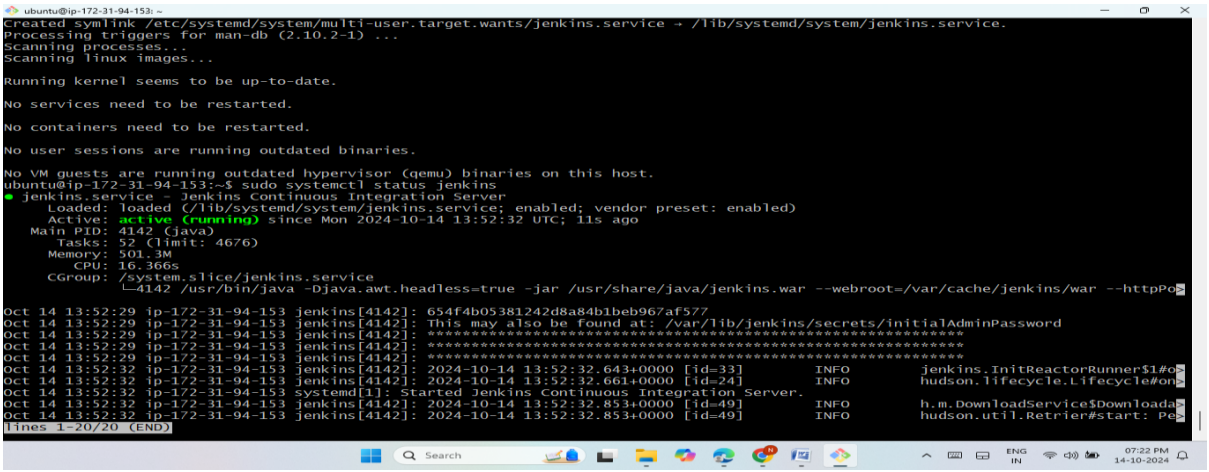
**https://pkg.jenkins.io/debian-stable binary/ | sudo tee **

/etc/apt/sources.list.d/jenkins.list > /dev/null

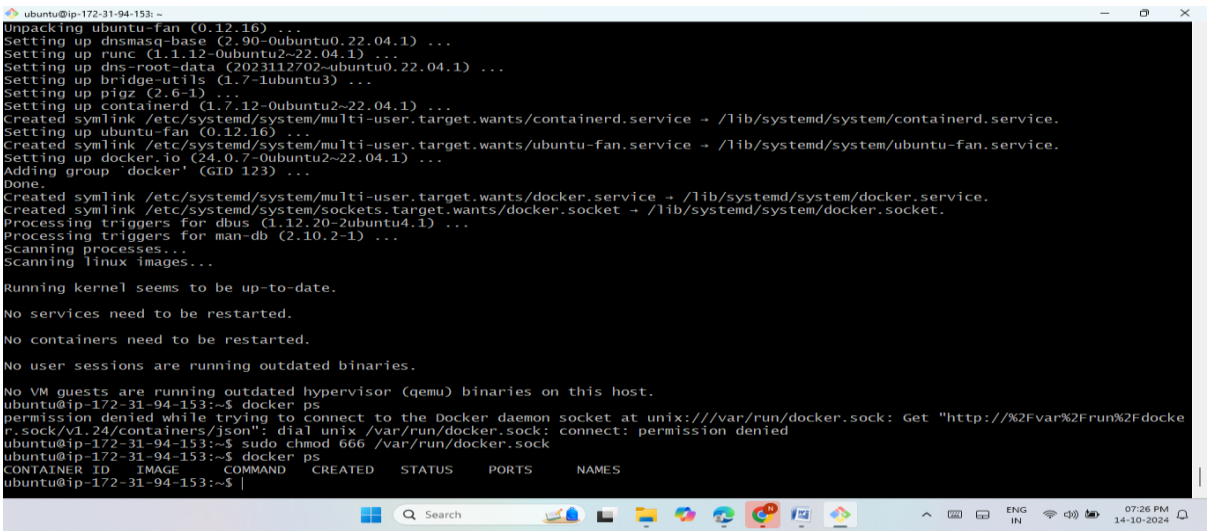
sudo apt-get update

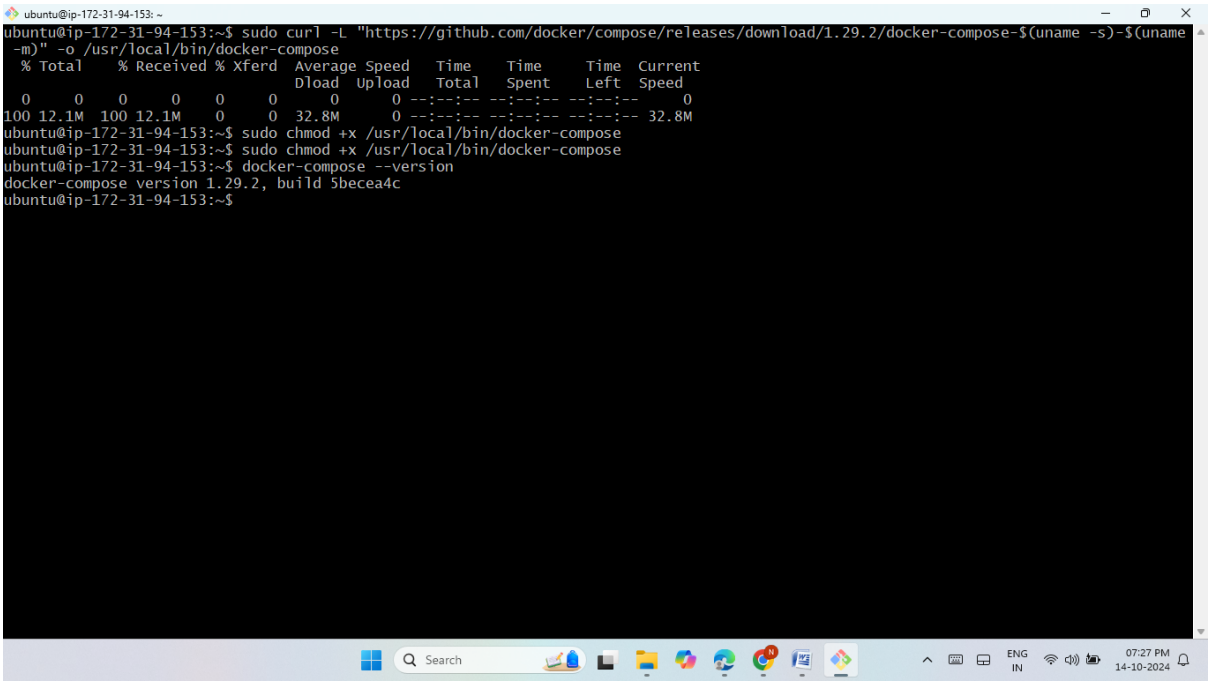
sudo apt-get install Jenkins

- No need to start and enable manually by choosing an ubuntu.

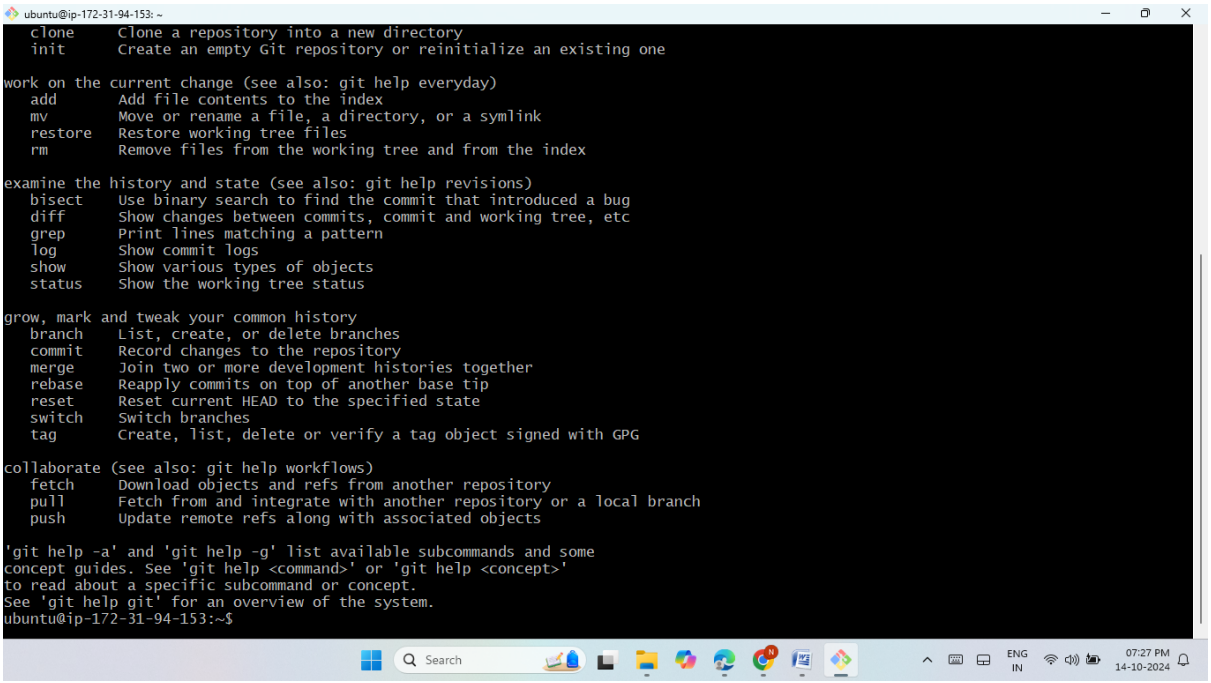


- Now install docker and docker-compose and then give permissions.

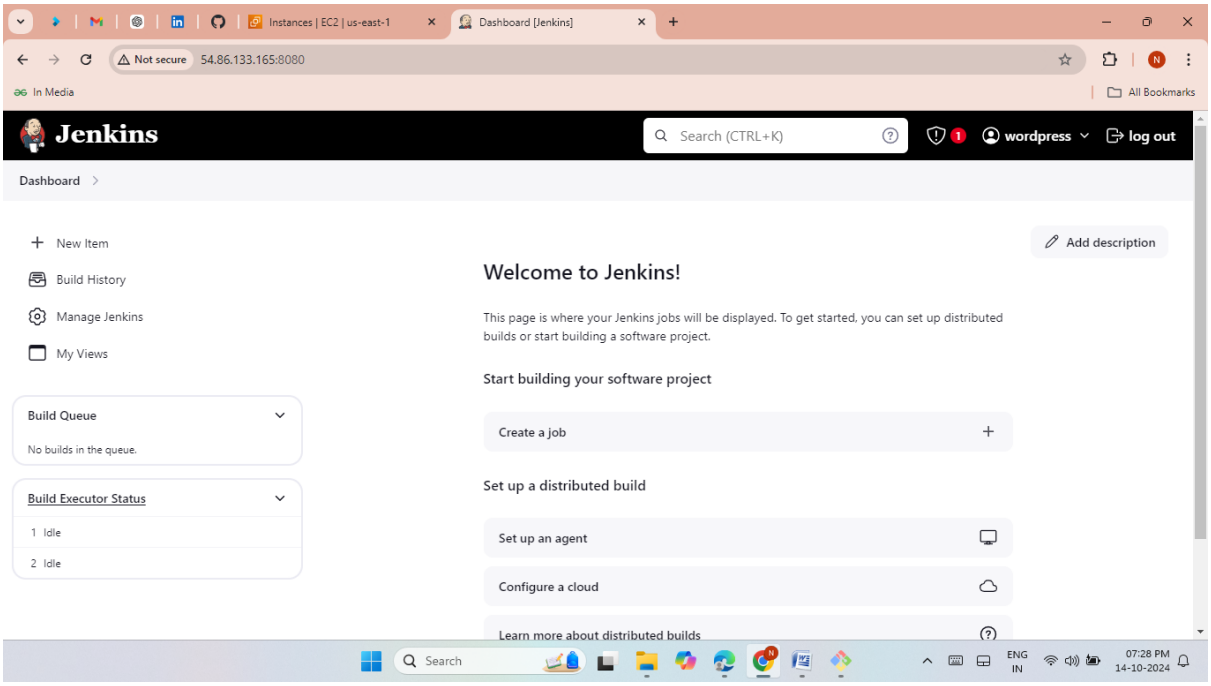




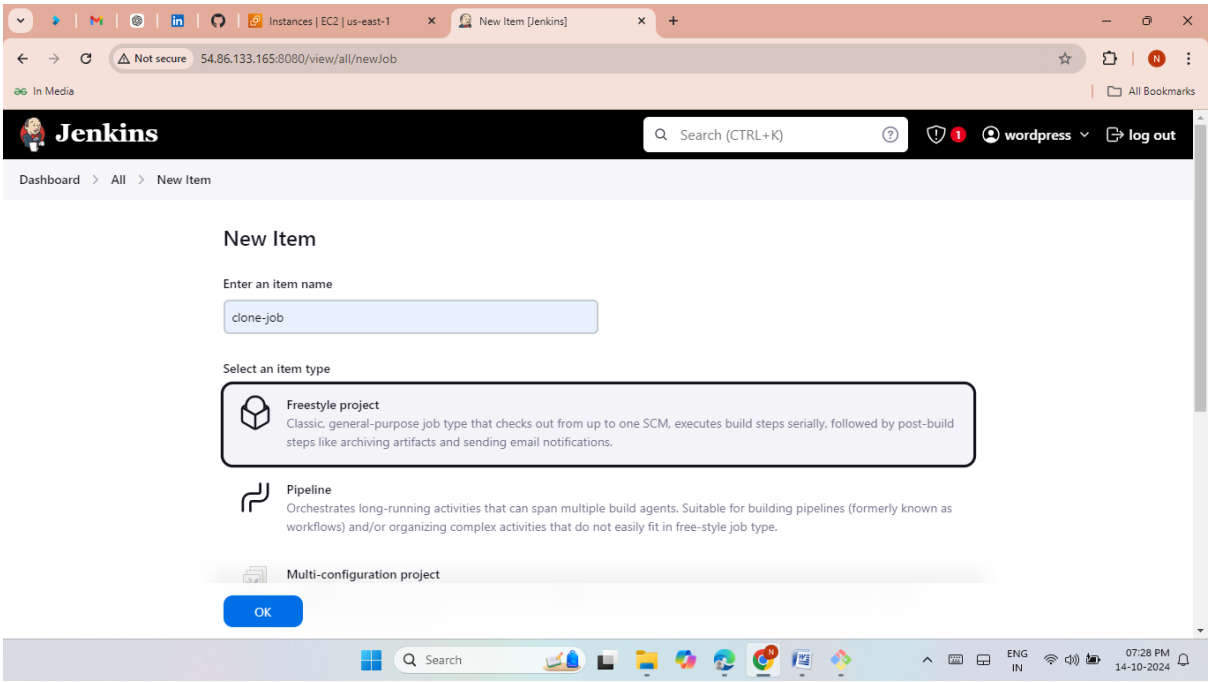
- No need to install git manully by choosing ubuntu.



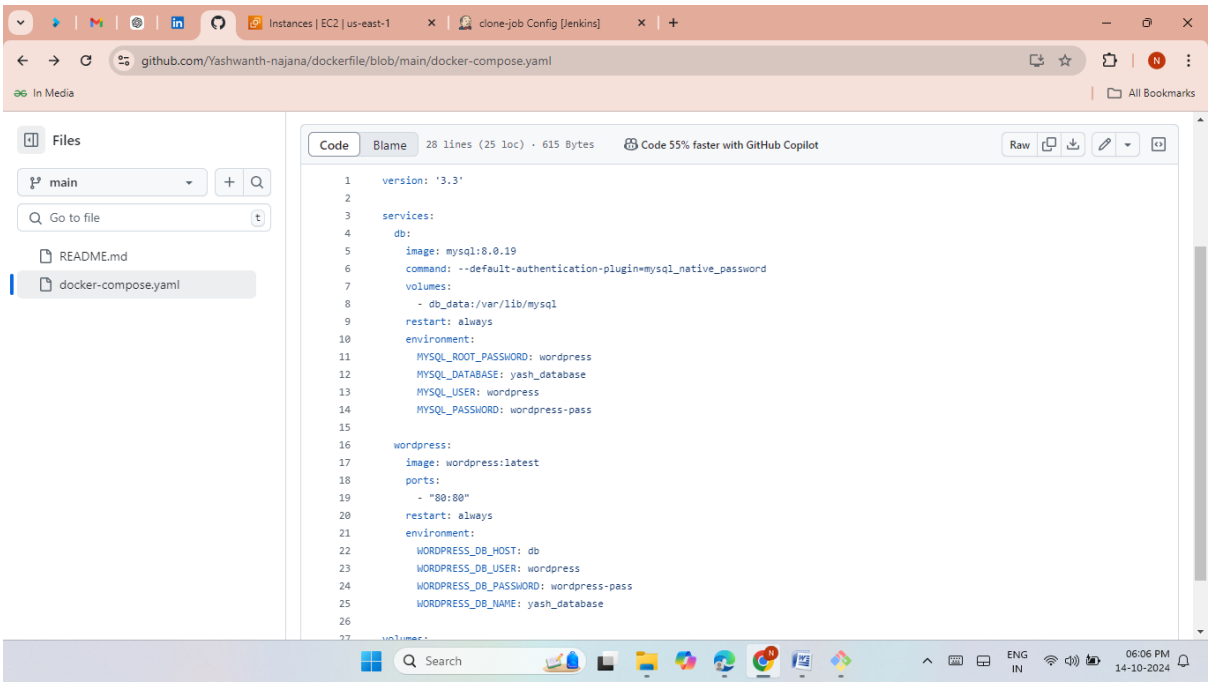
- Now copy that static Public IP and past it in Google browser.
- We can get the Jenkins.



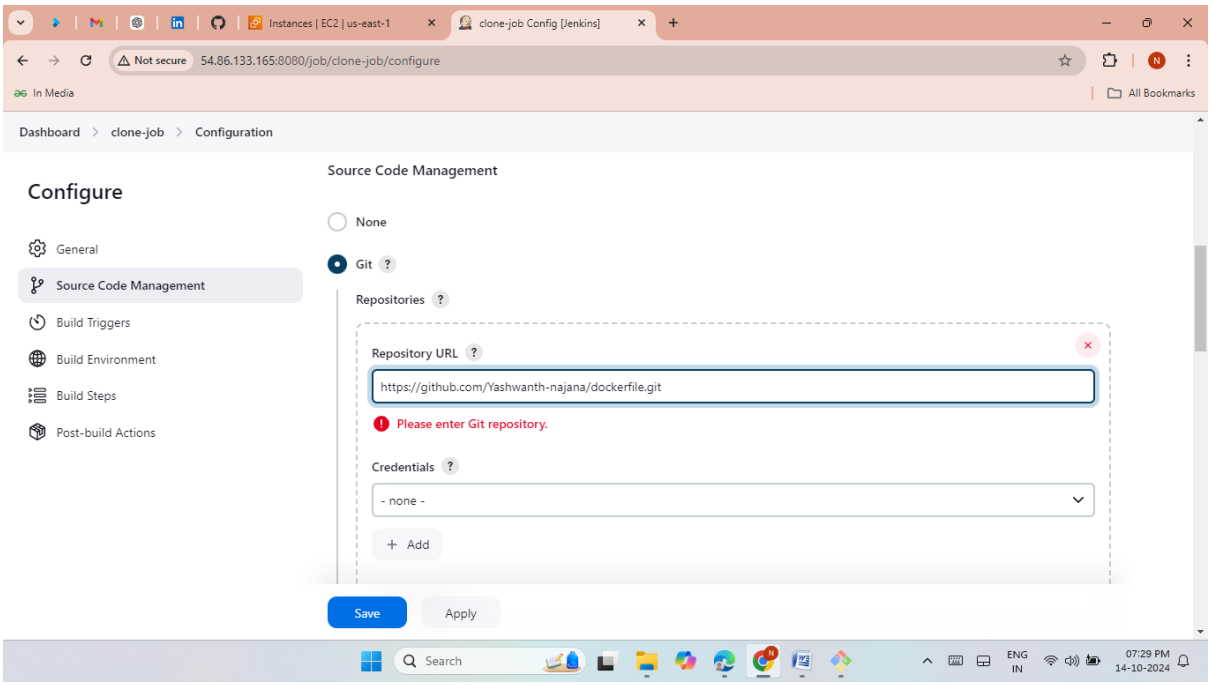
- Now create a clone-job for cloning the repo from github.



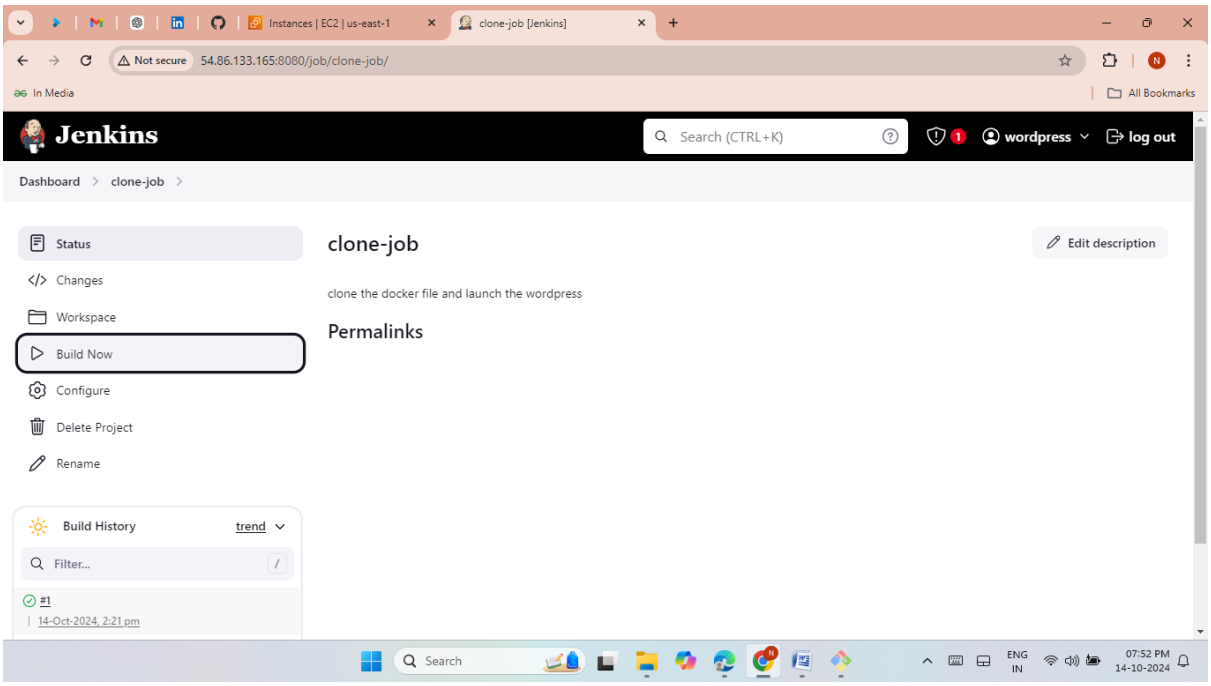
- This is the docker-compose file from github.



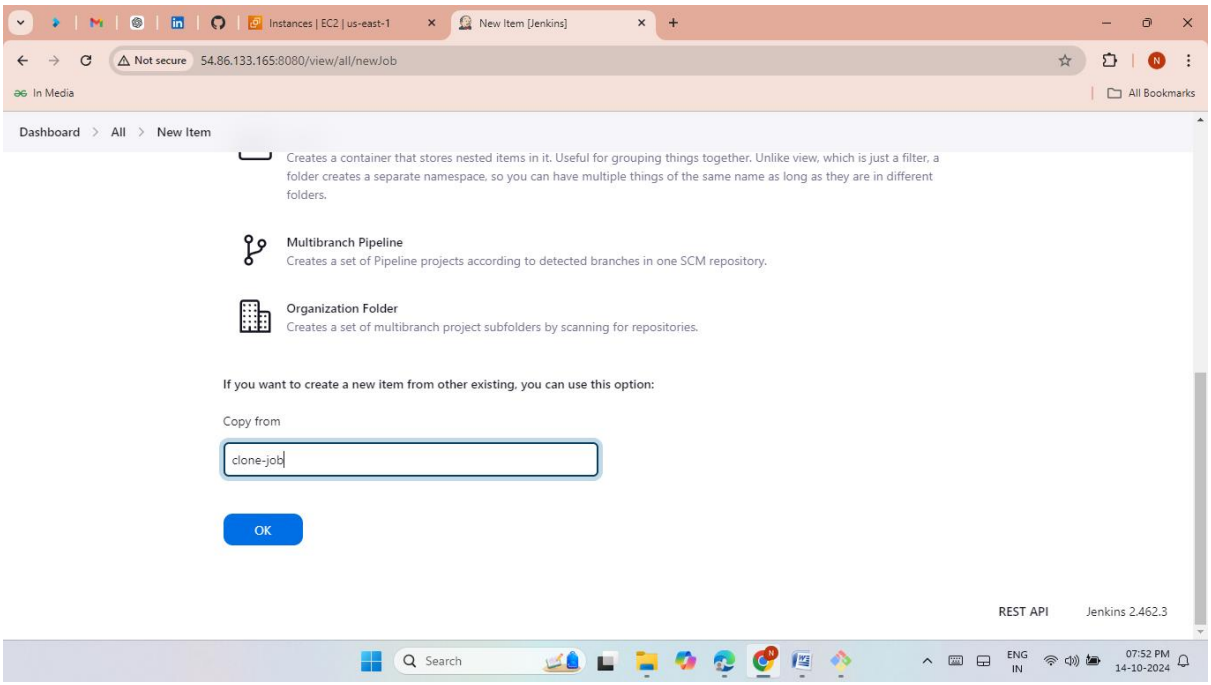
- Now Clone the repository URL.



- Now click on build now. it shows success.

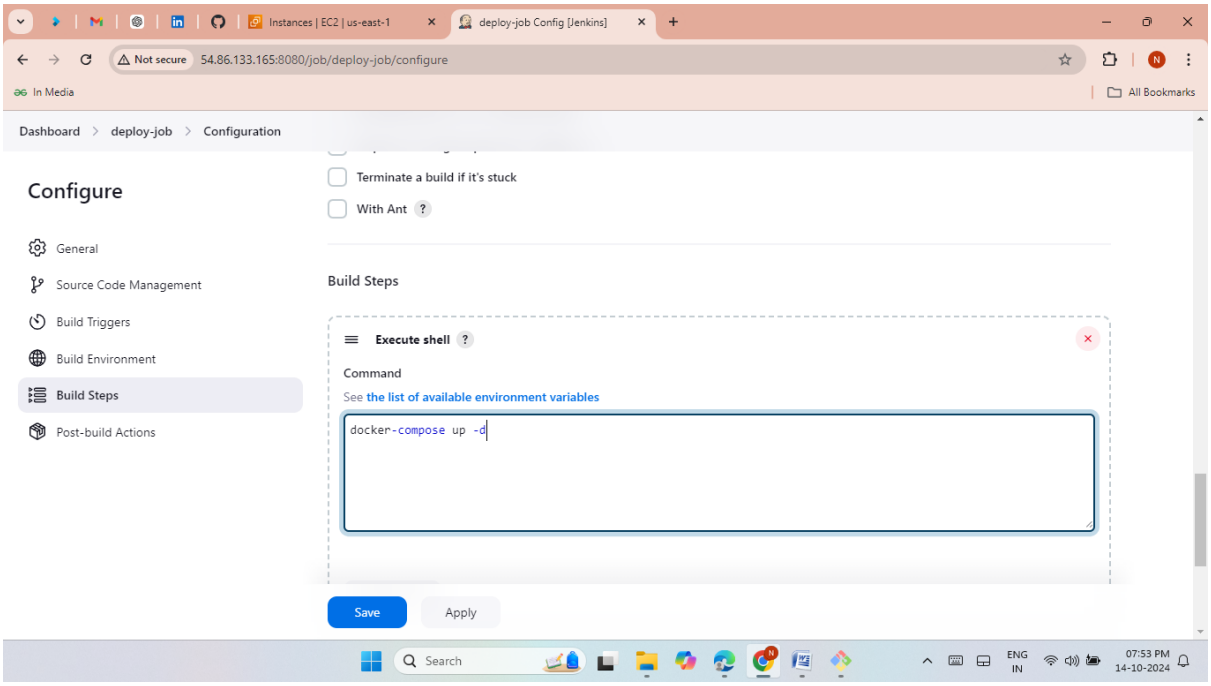


- Now create a deploy job and add existing clone-job.

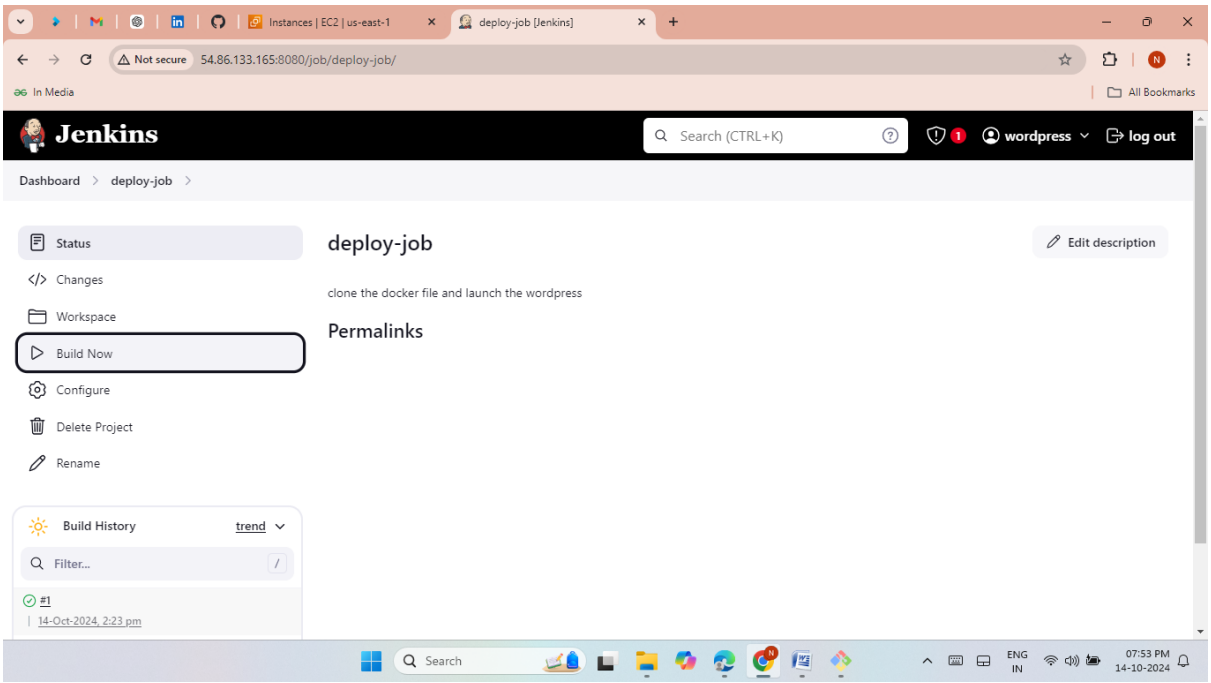


- Now write the script in execute shell.

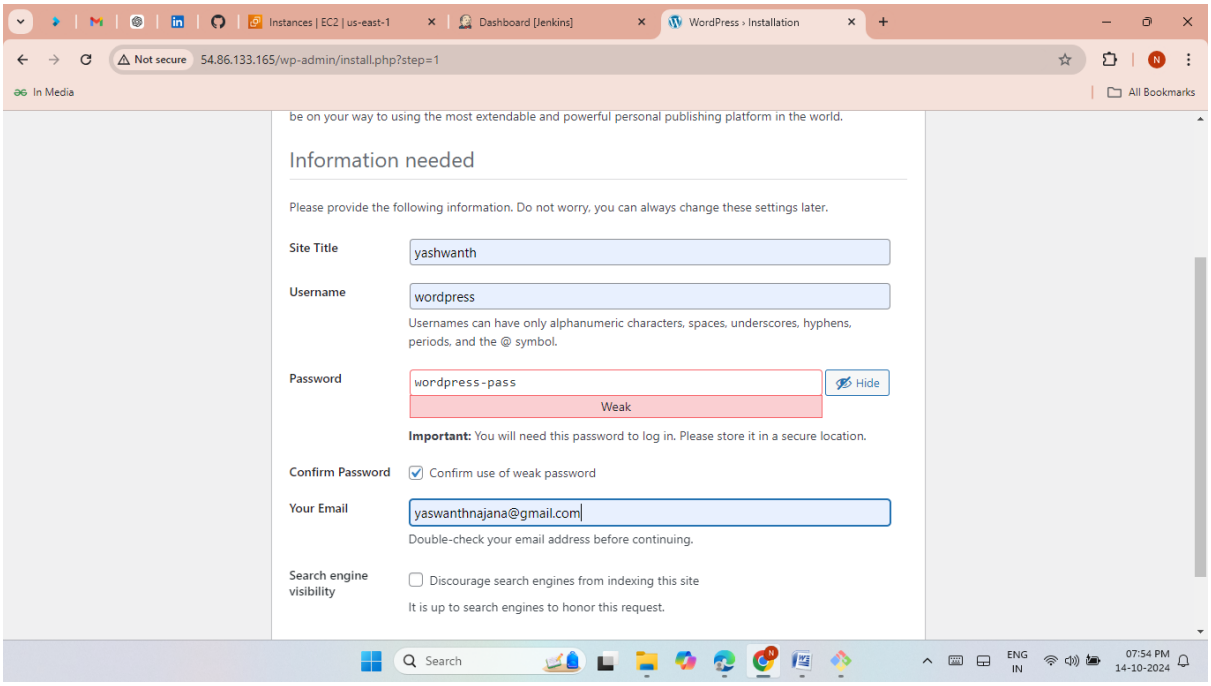
  docker-compose up –d

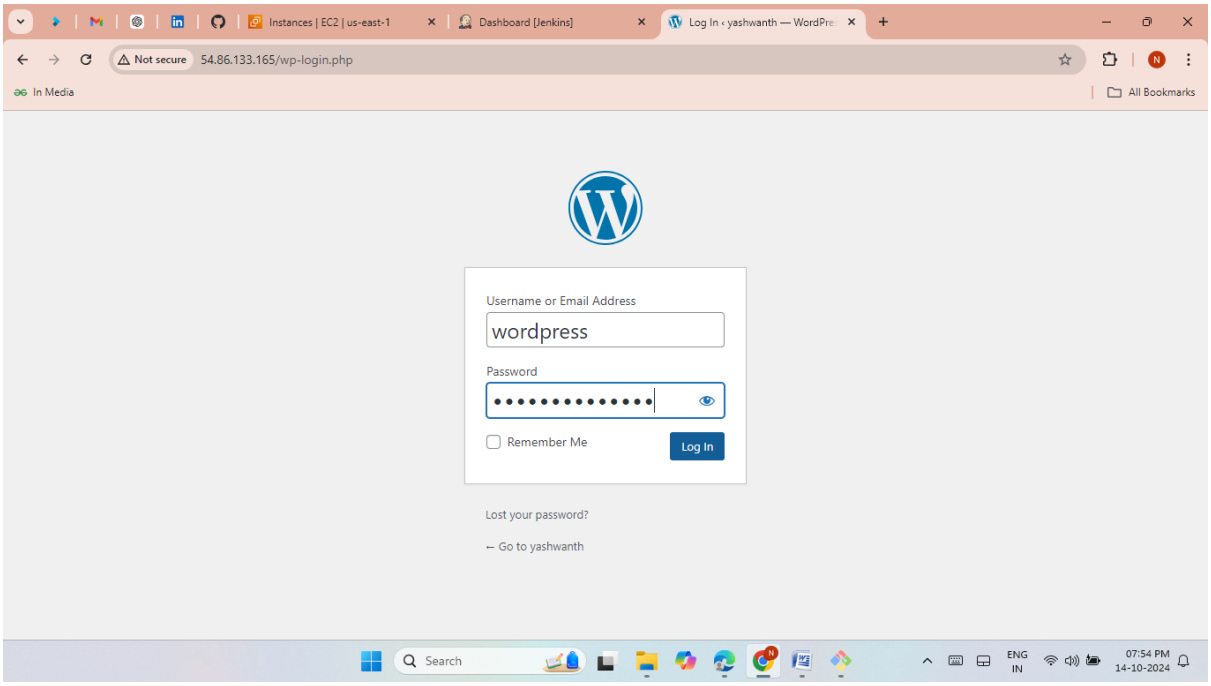


- Now it shows job is success.

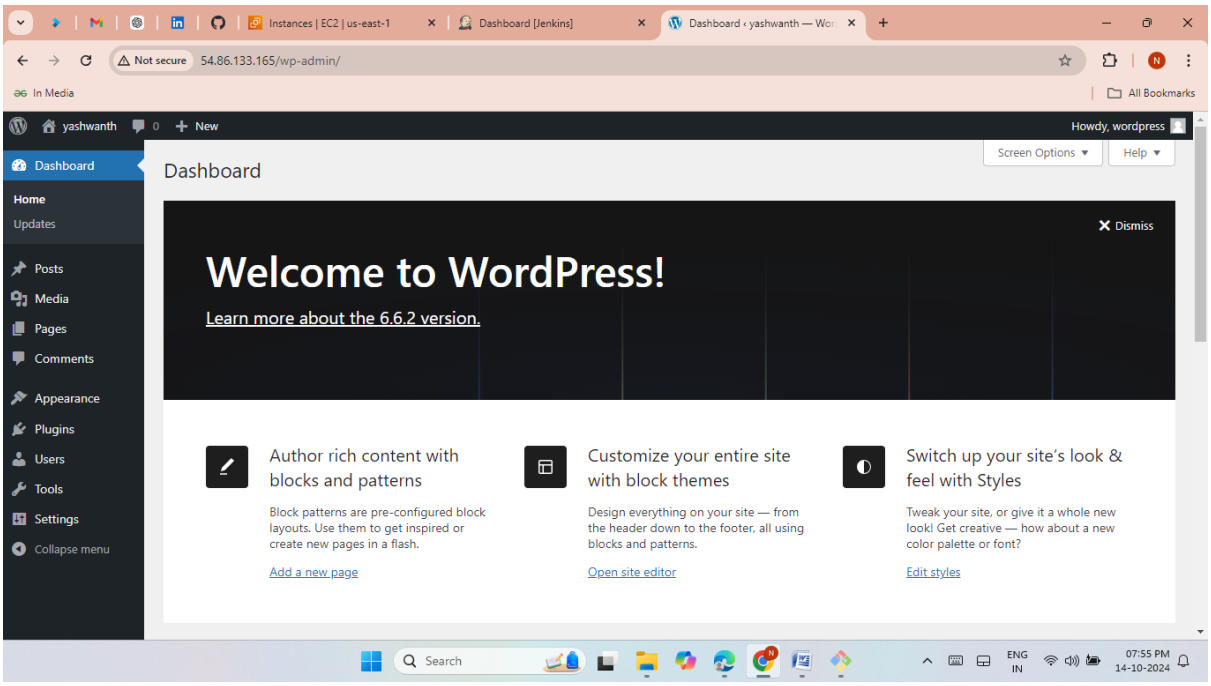


- Now copy that Static Public IP and past it in Google browser then select the language and click on continue.





- Here the wordpress launched successfully.



METHOD-4

Deploy WordPress web application by using userdata of EC2 instance

- Write the script in ec2 instance.

```
#!/bin/bash
yum update -y
yum install -y git
git clone https://github.com/Yashwanth-najana/dockerfile.git
cd dockerfile/
yum install -y docker
systemctl start docker
systemctl enable docker
chmod 666 /var/run/docker.sock
DOCKER_COMPOSE_VERSION="v2.20.0"
```

curl -L

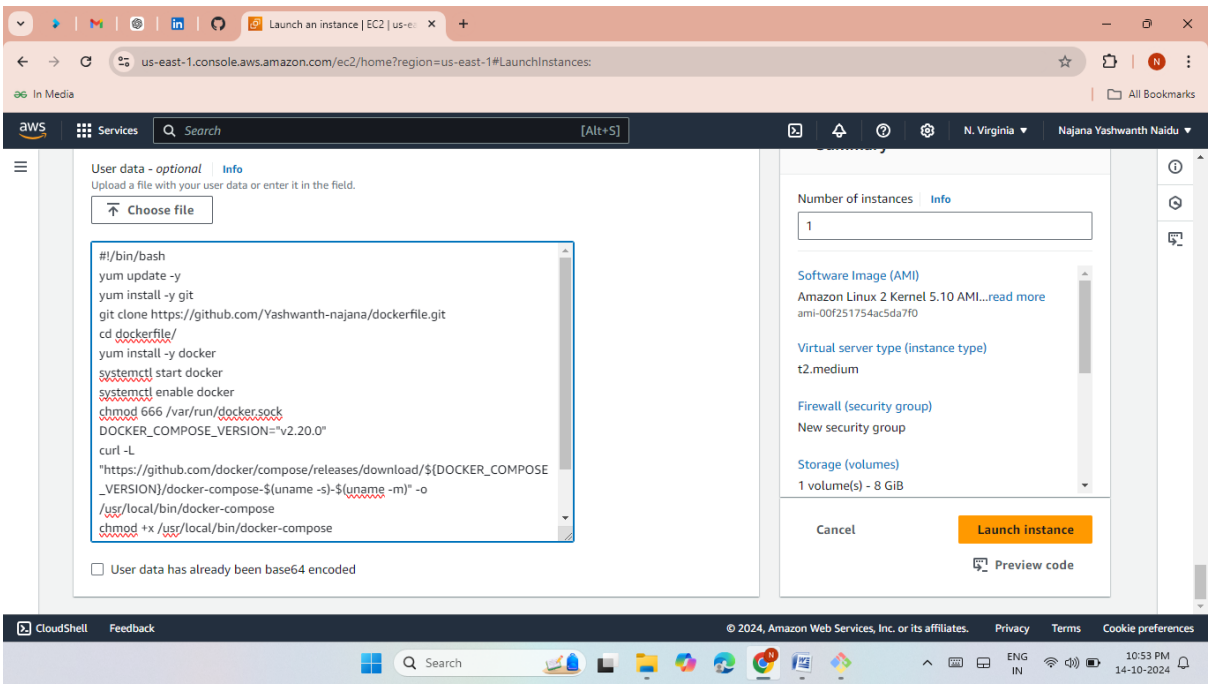
"https://github.com/docker/compose/releases/download/${DOCKER_COMPOSE_VERSION}/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

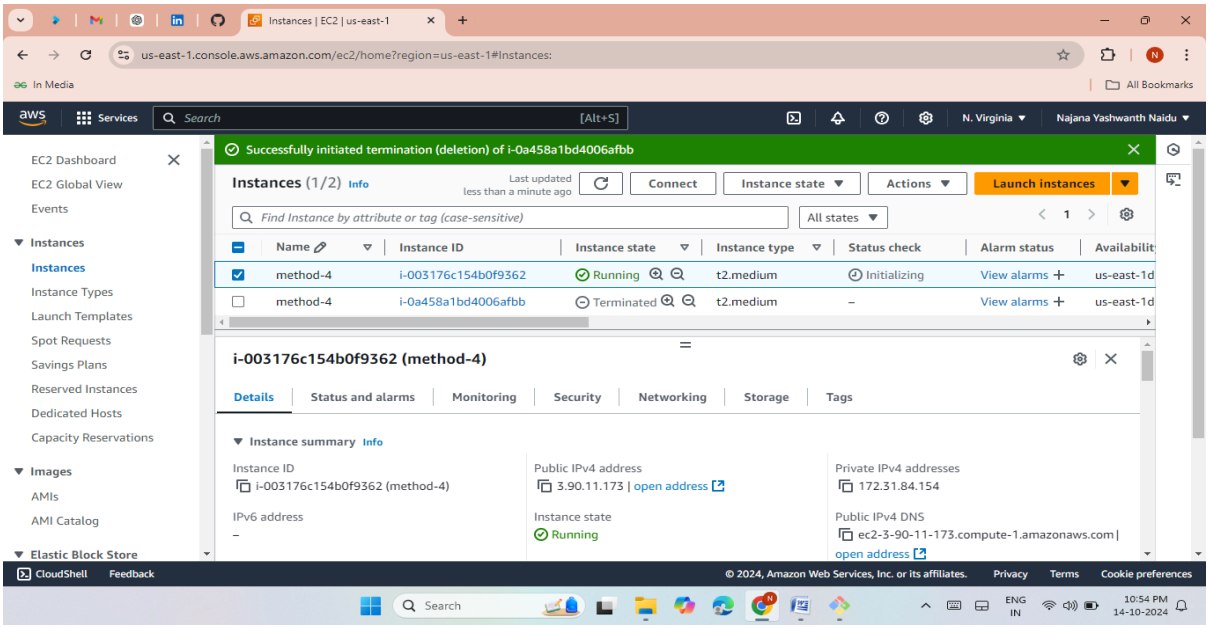
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

docker-compose up -d

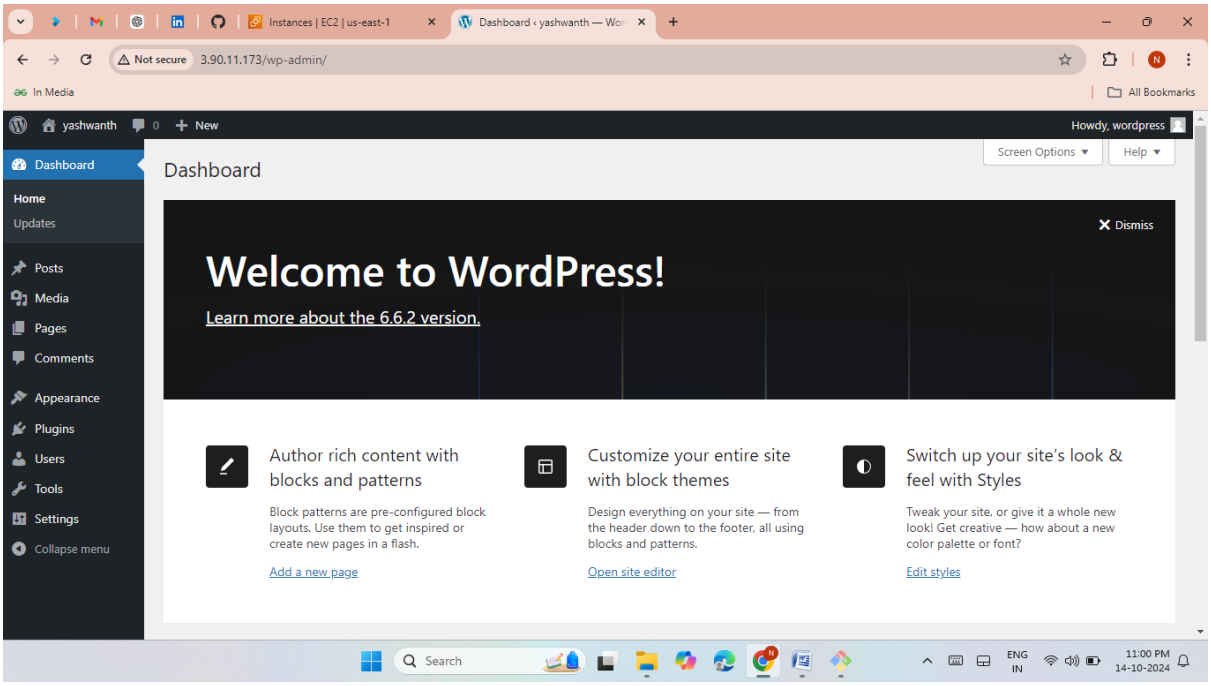
docker ps

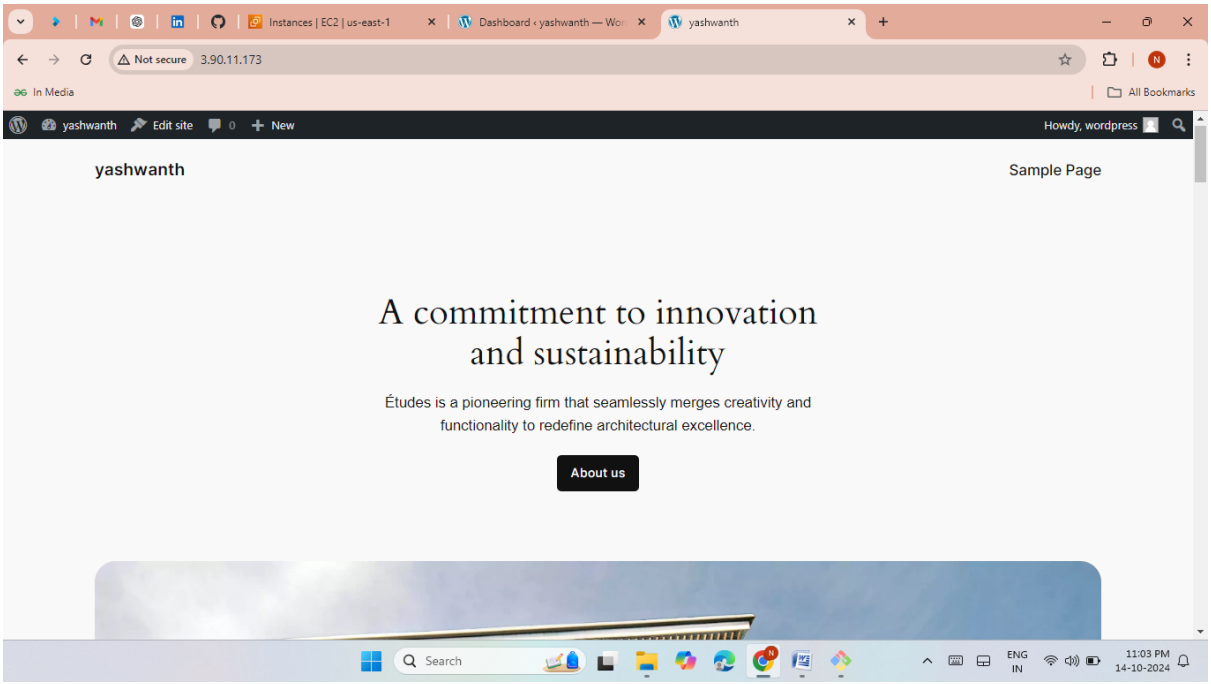


➢ Here the instance was launched with the user data.



- Now copy that Static Public IP and past it in browser.
- Here the wordpress was launched successfully.

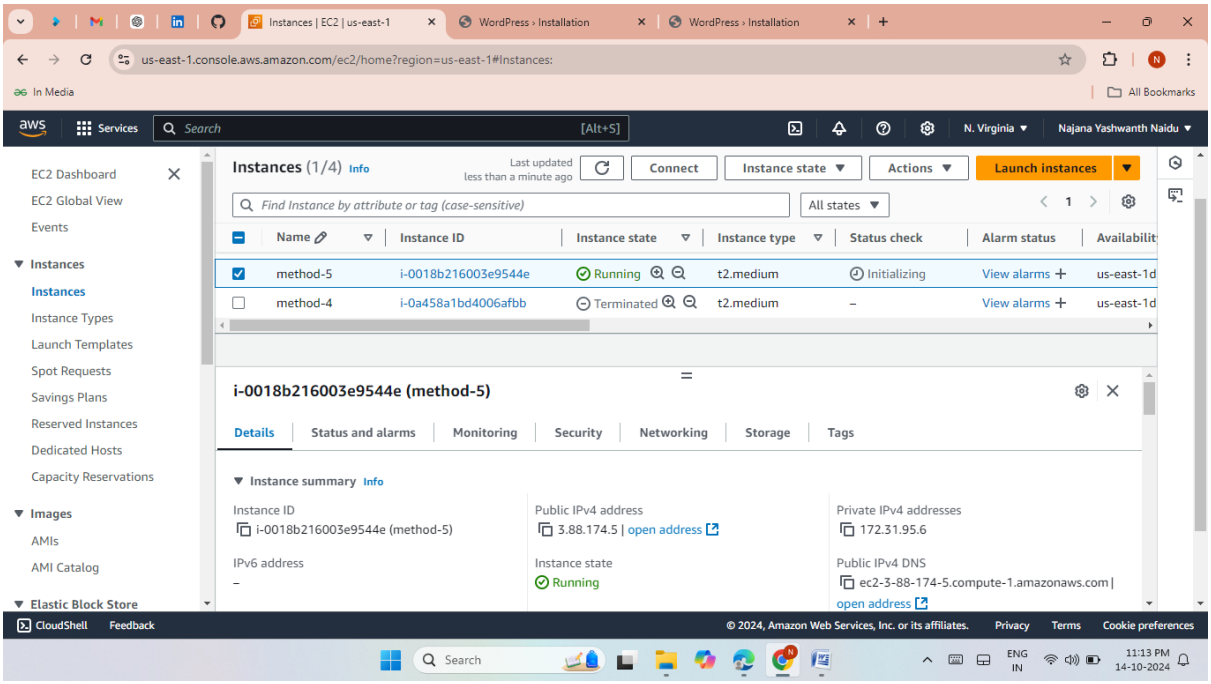




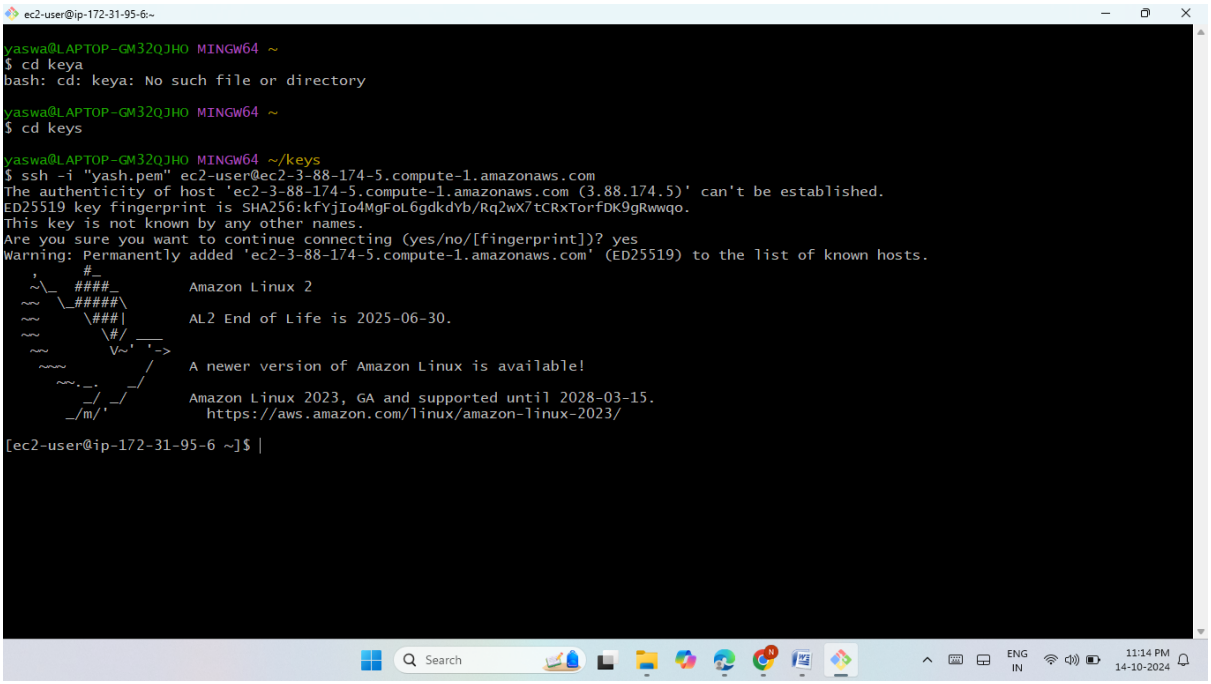
METHOD-5

Deploy WordPress web application by using git and jenkins execute shell (bash script)

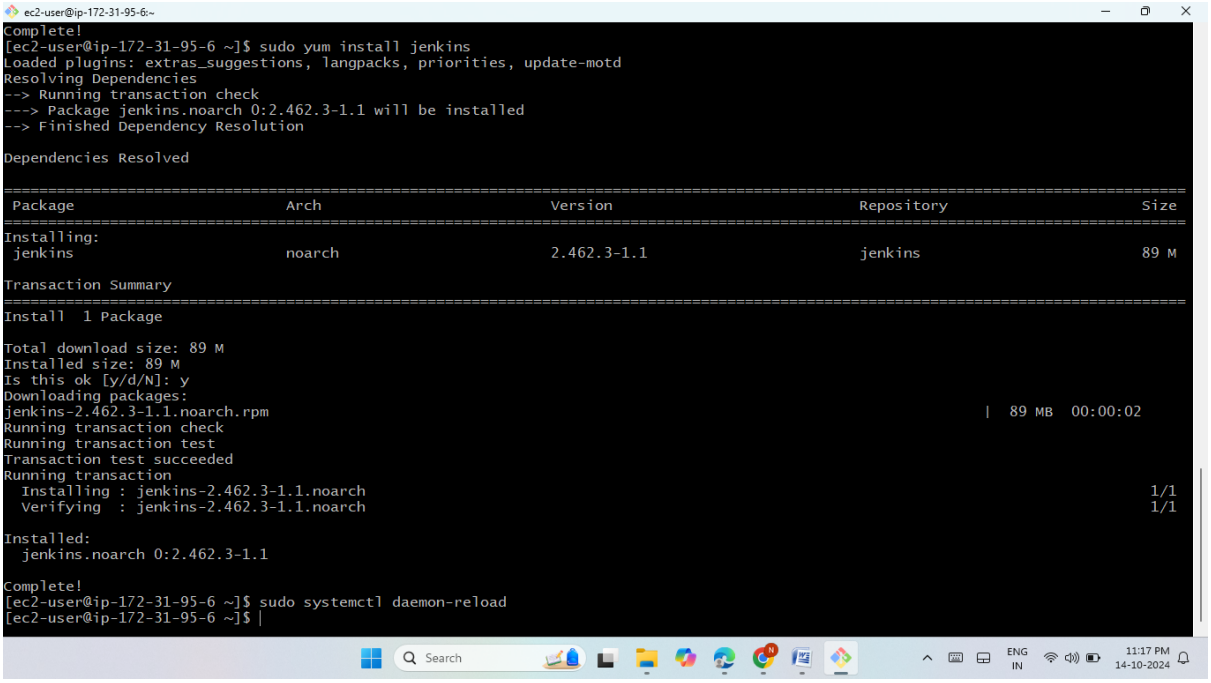
- Launch an EC2 instance with the jenkins port number.

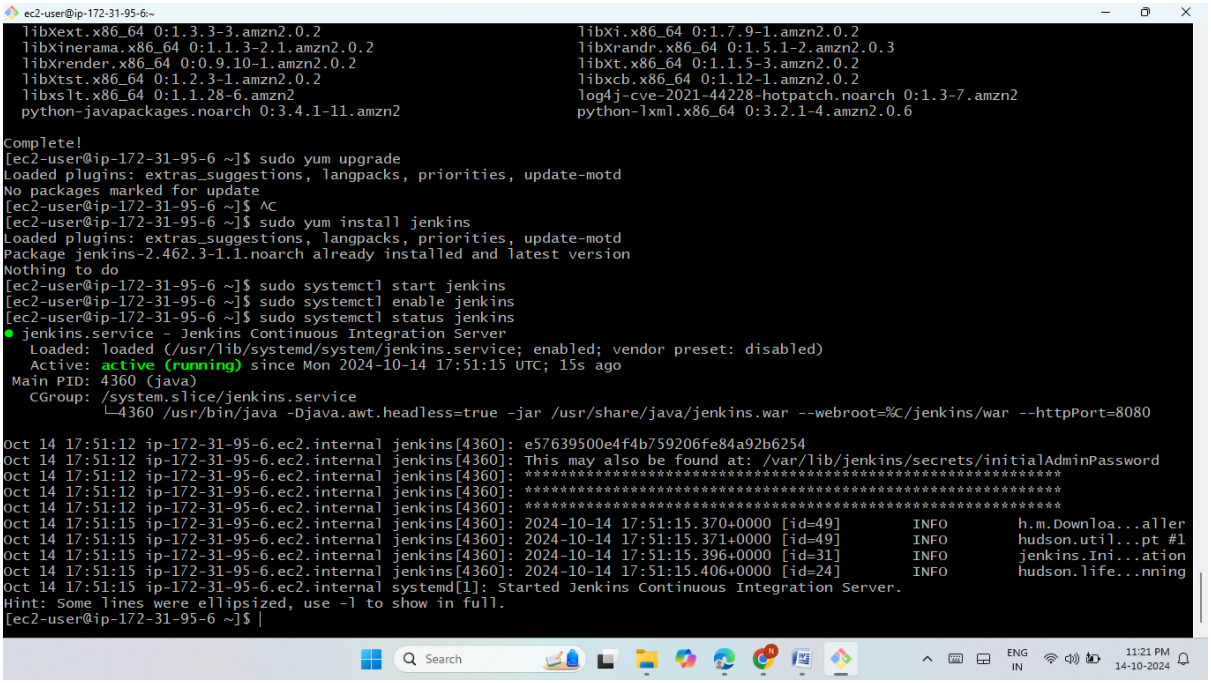


- Now connect the server locally.

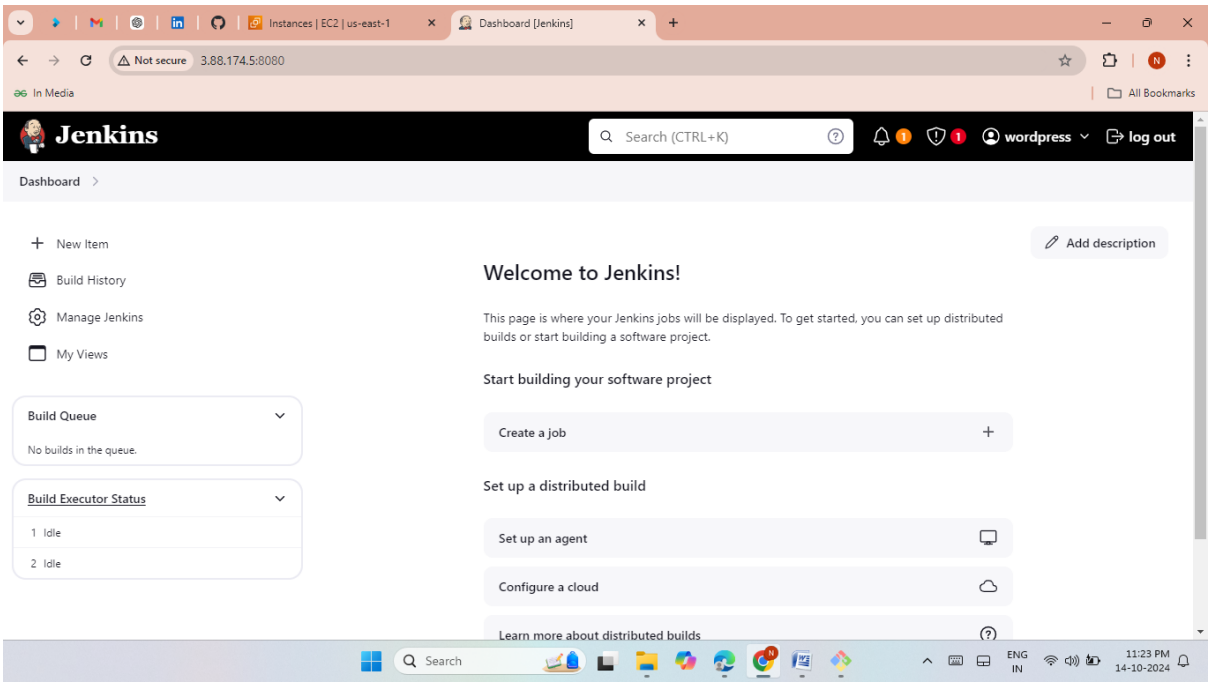


- Now install Jenkins and then start and enable.

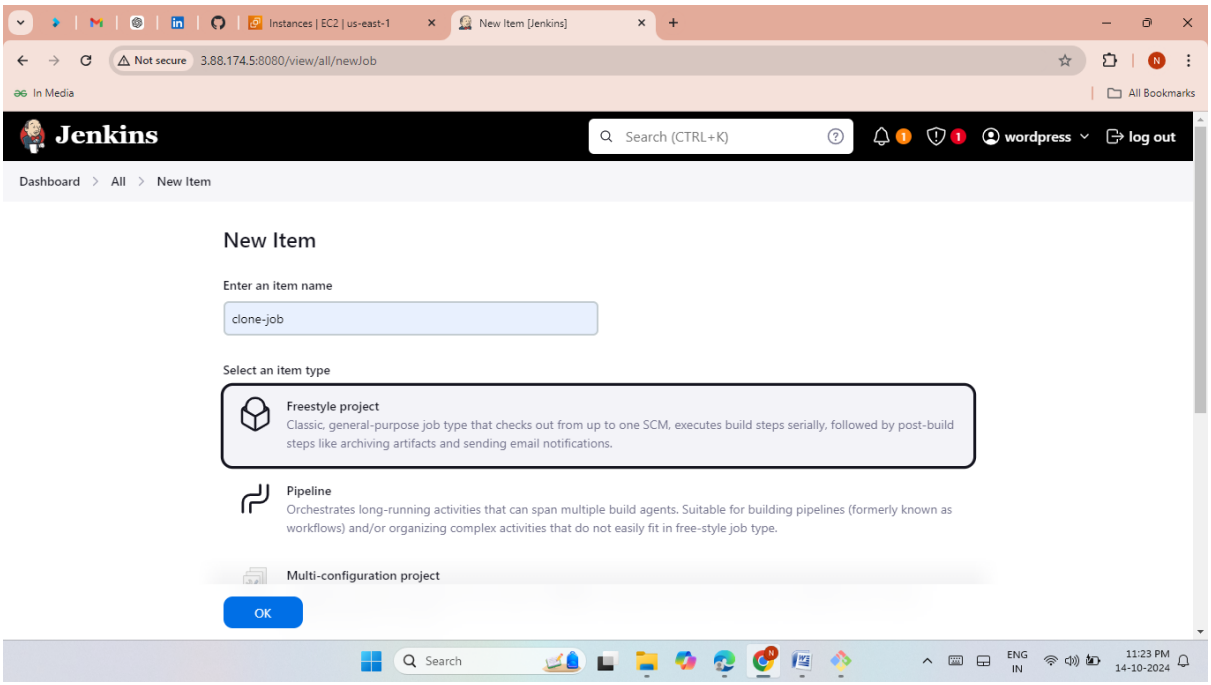




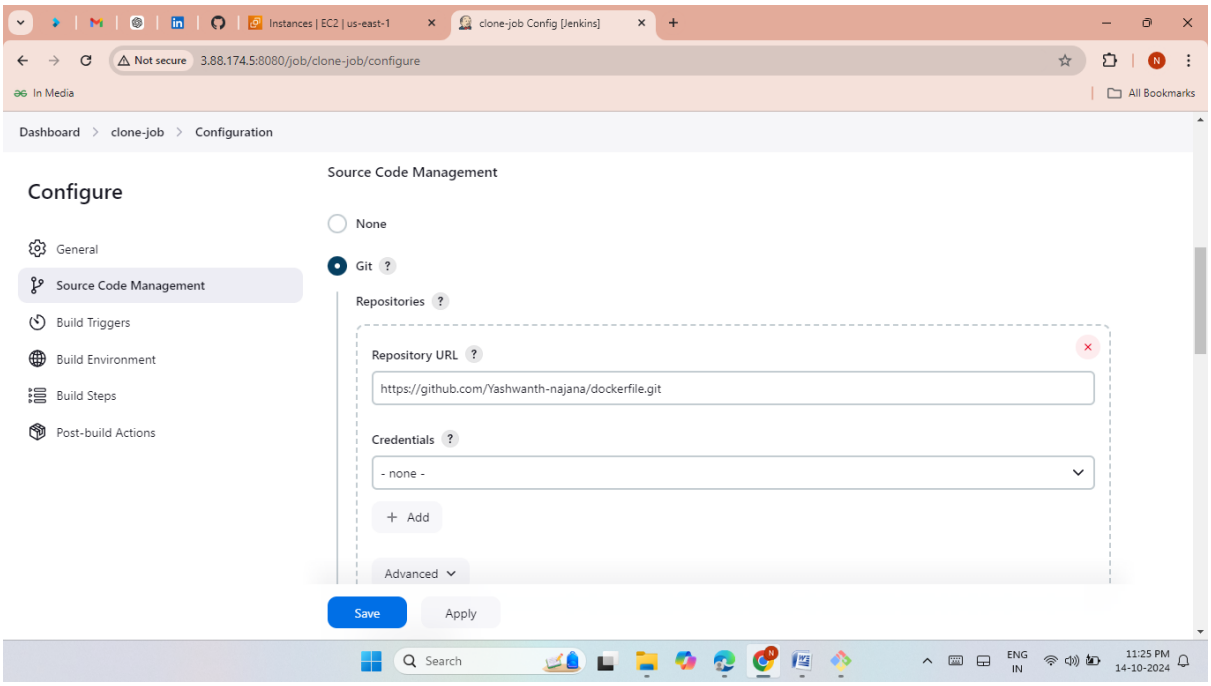
➢ Here the Jenkins was launched.



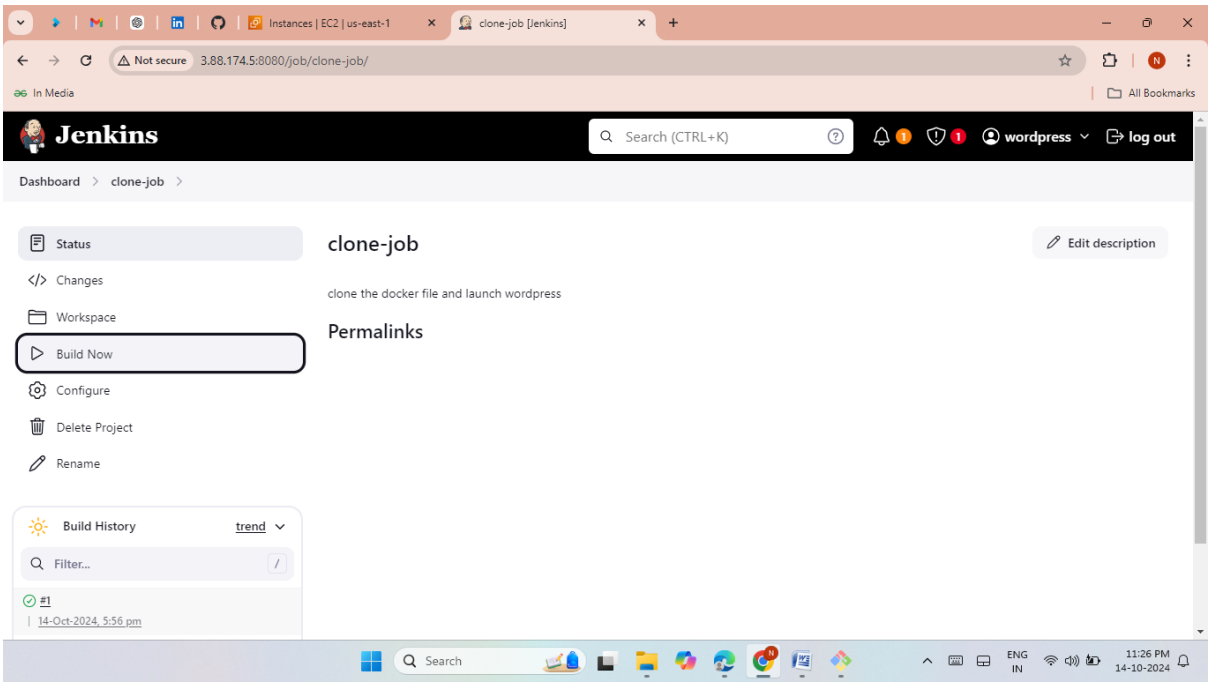
➢ Create a clone job for cloned the repo.



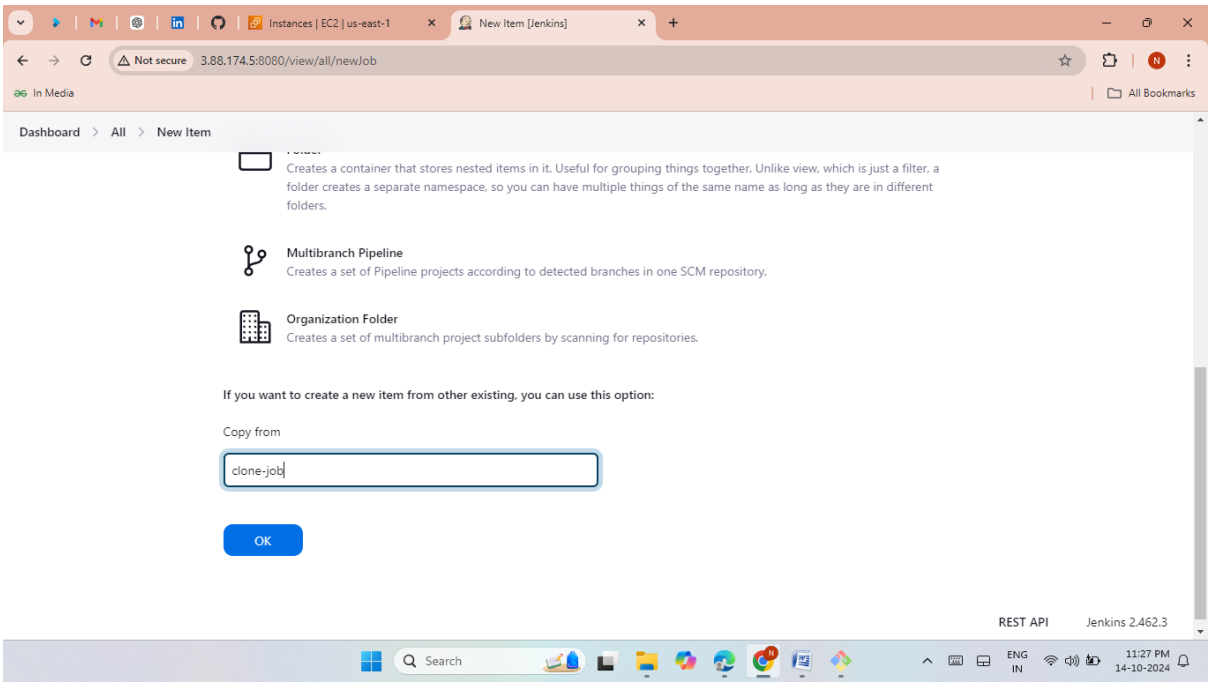
➢ Now Clone the repository URL.



➢ Here the build is success.



➢ Now Create the deploy job and add it to the clone job.



➢ Now write the script in execute shell.

#!/bin/bash

set -e

sudo yum update -y

sudo yum install -y docker

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker jenkins

sudo chmod 666 /var/run/docker.sock

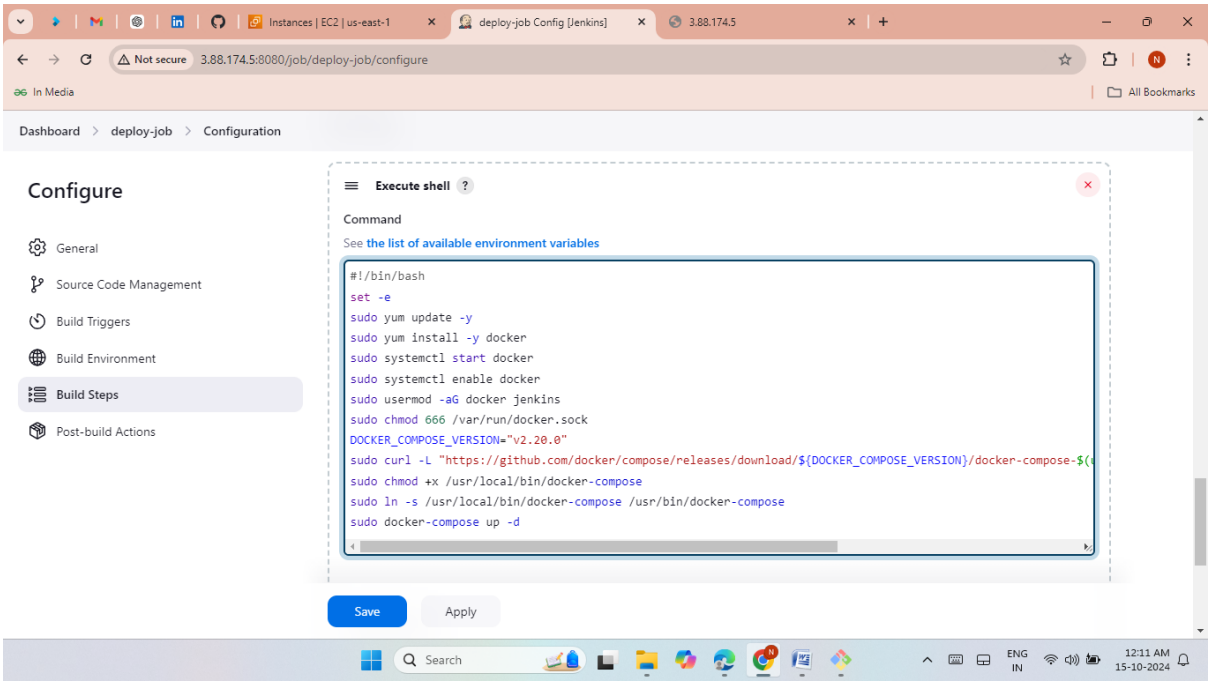
DOCKER_COMPOSE_VERSION="v2.20.0"

sudo curl -L "https://github.com/docker/compose/releases/download/${DOCKER_COMPOSE_VERSION}/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

sudo docker-compose up –d



- Edit the **sudoers** File:
- You need to add the Jenkins user to the **sudoers** file to allow it to execute commands without a password.

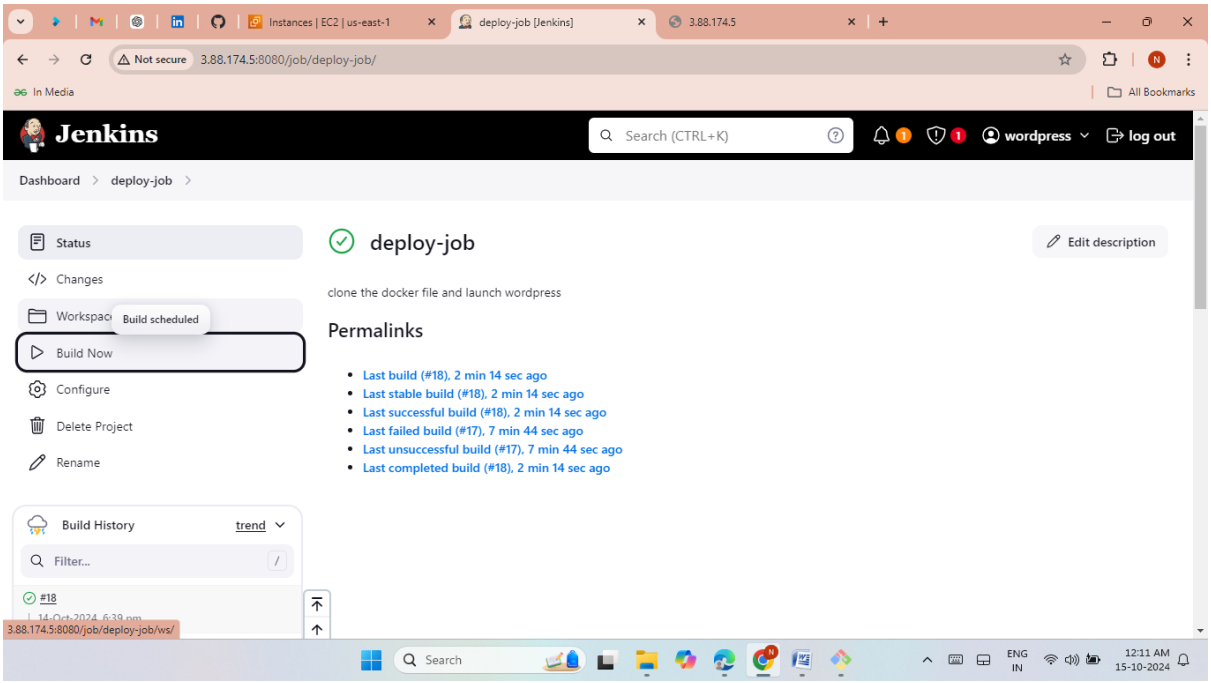
  Sudo visudo
- Add the Following Line->At the end of the file

  jenkins ALL=(ALL) NOPASSWD: ALL

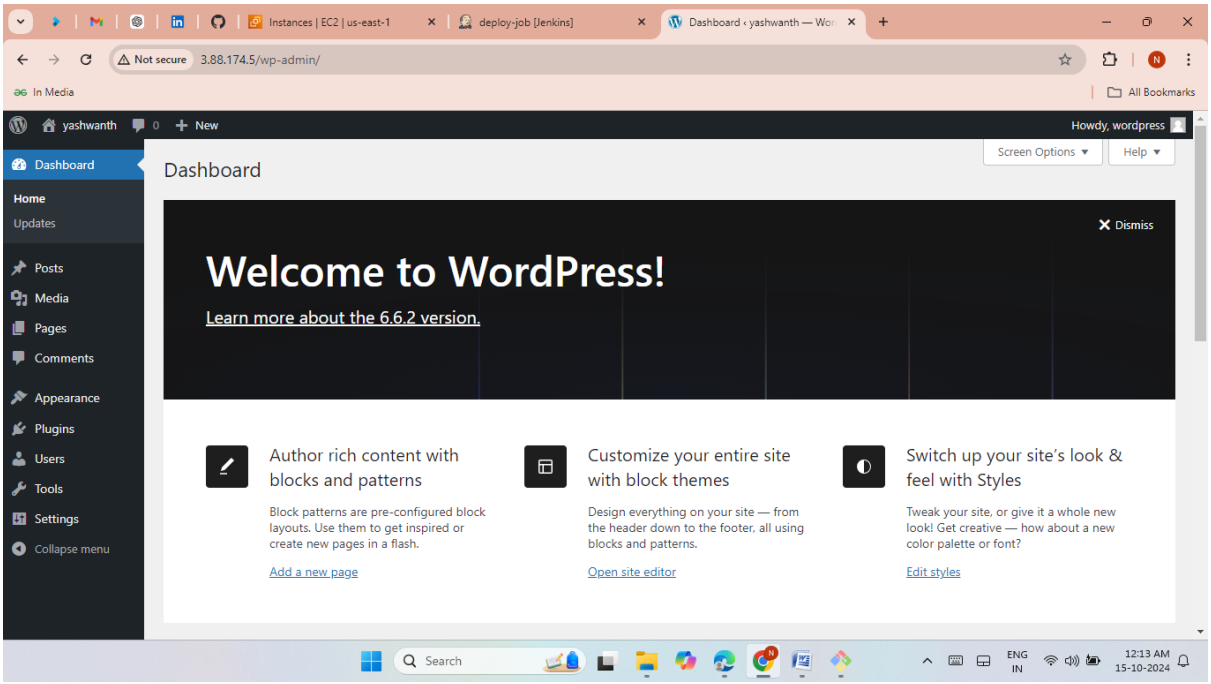
- you want to give the Jenkins user elevated privileges for Docker-related commands. you can add the Jenkins user to **the docker** group.
- Run the following command**s.**

  sudo usermod -aG docker Jenkins

  sudo systemctl restart Jenkins
- Now it shows build is success.



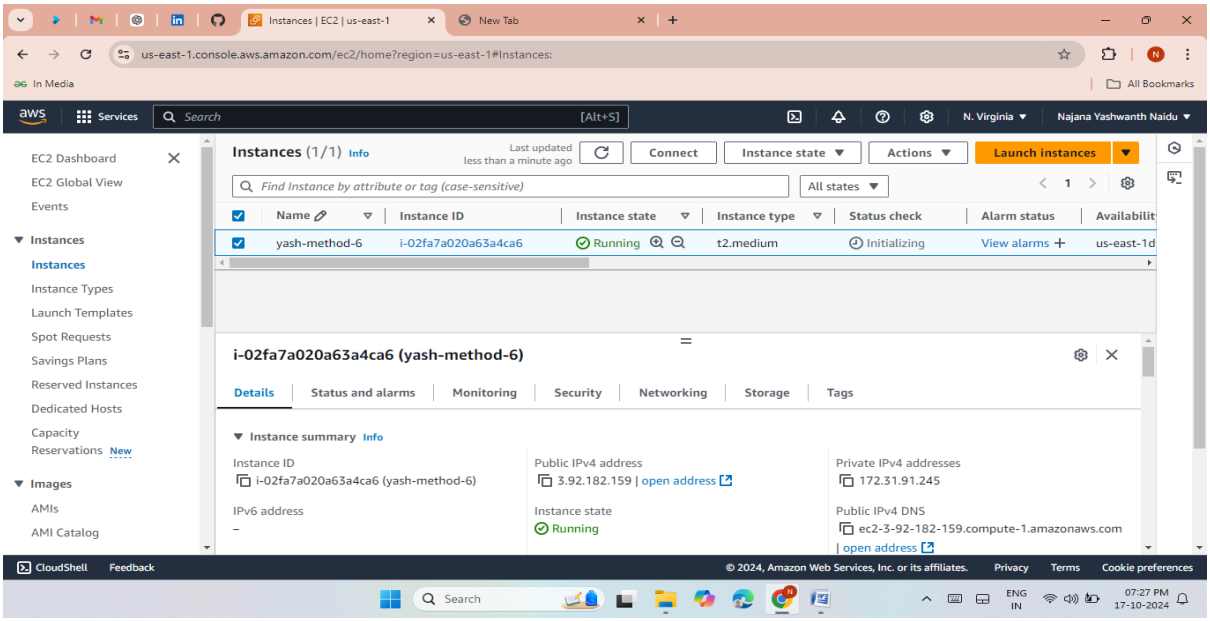
- Now copy that static Public IP and past it in browser.
- Here the wordpress was launched successfully.



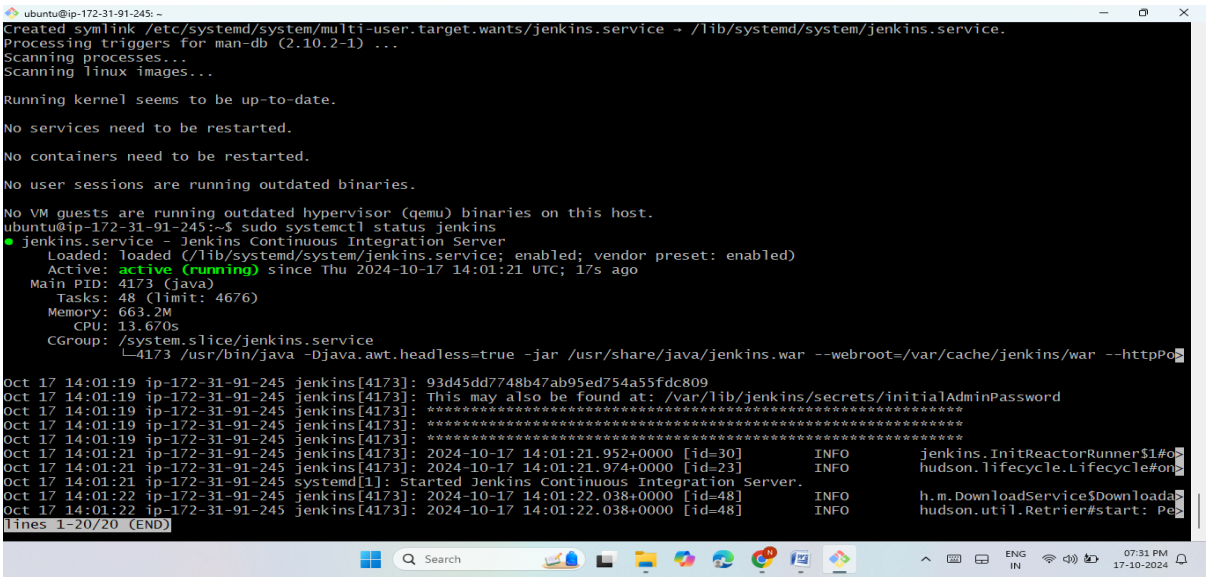
METHOD-6

Deploy WordPress web application by using git and jenkins execute shell (bash script) create jenkins pipeline add build periodically and poll scm to initial job of pipeline and check the changes happened or not which are made in github repo.

- Now launch an EC2 instances with the Jenkins port number and ubuntu AMI.



- Here no need to Install git in ubuntu.
- Now install Jenkins then no need to start and enable manually in ubuntu.

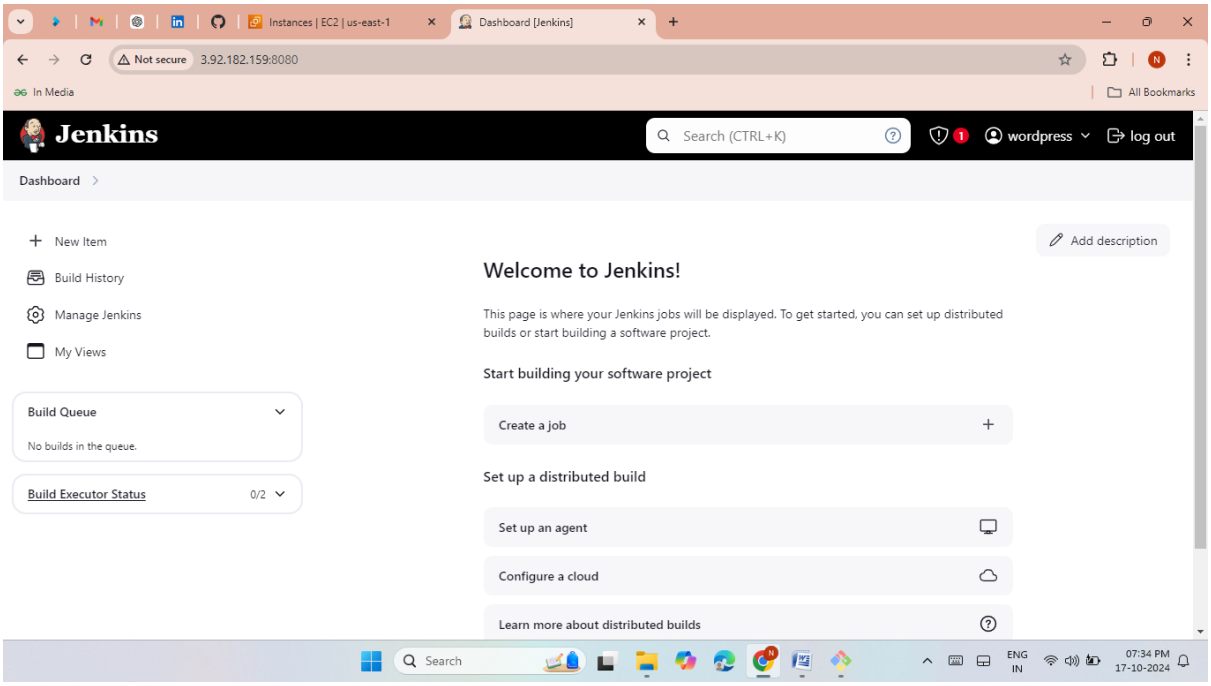


- Give permissions to Jenkins.

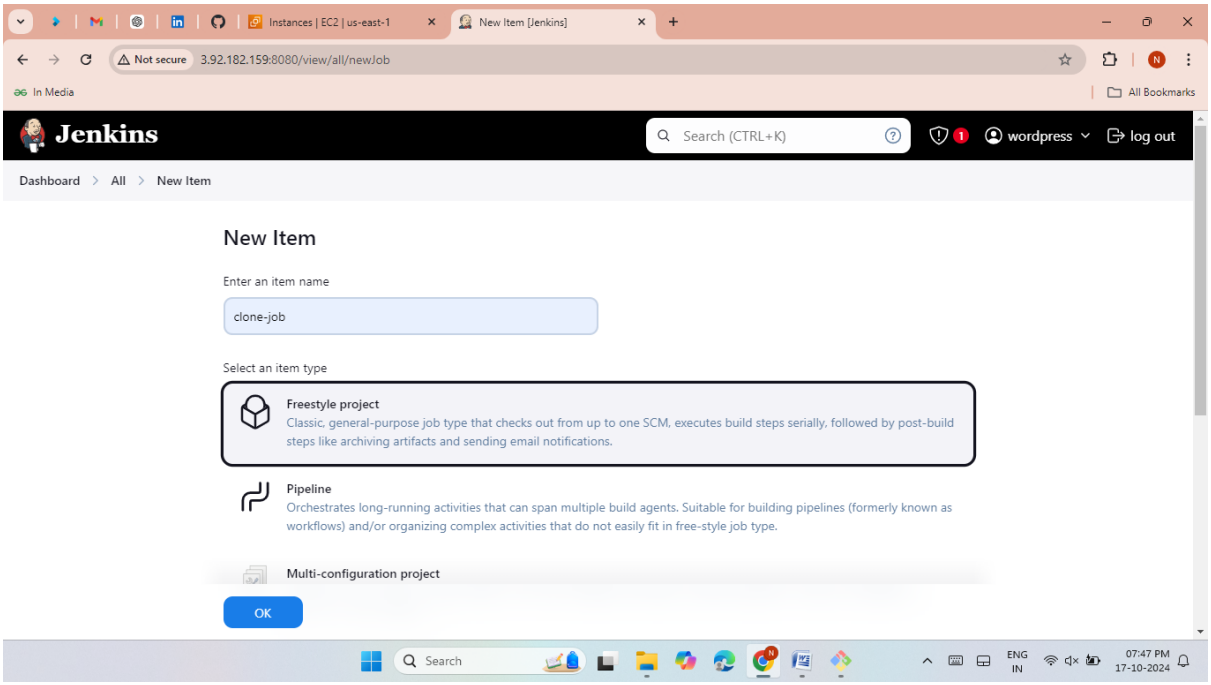
  sudo visudo

  jenkins ALL=(ALL) NOPASSWD: ALL

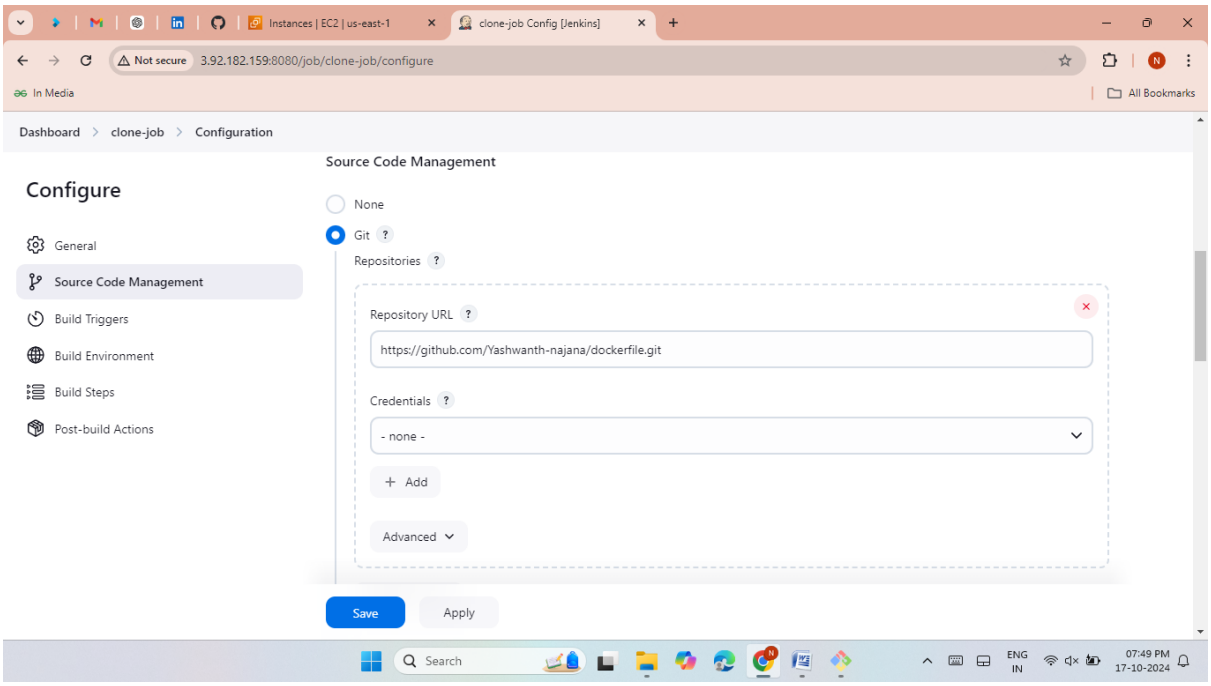
- Here the Jenkins was launched.



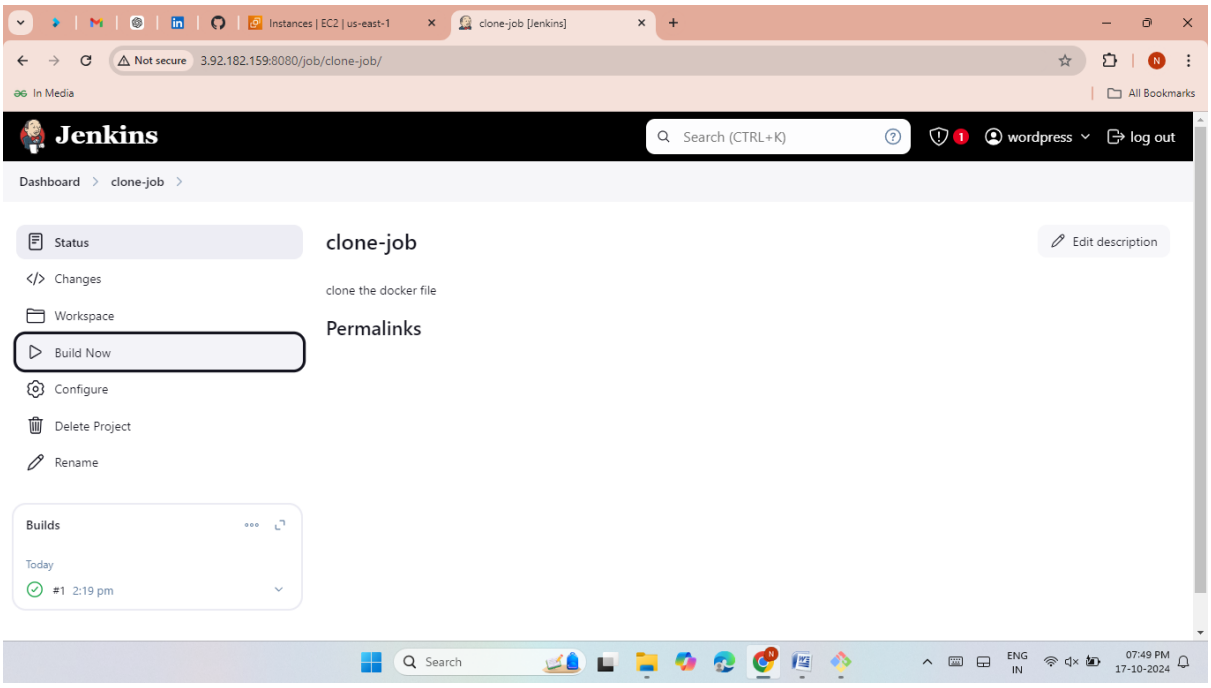
- Create a clone-job for clone the repo from github.



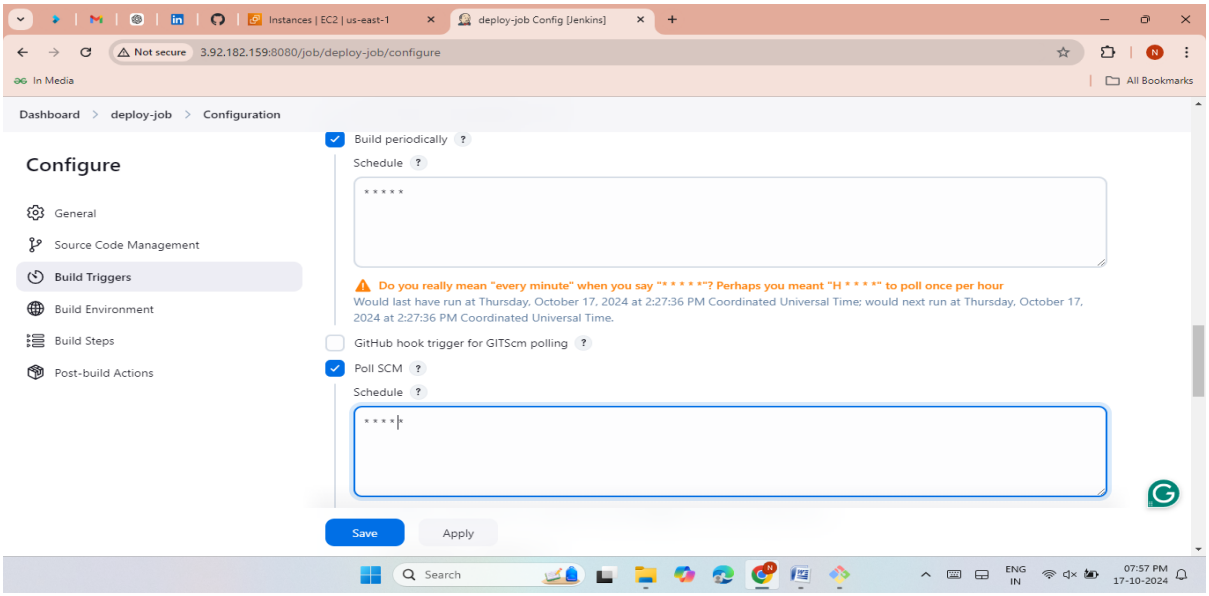
➢ Now past the docker-compose repository.



➢ Here the build was success.



- Now create the deploy job and add the clone-job.
- Now add the build periodically and poll SCM Schedules.



- Now write the script in execute shell.

sudo apt update

sudo apt install -y docker.io

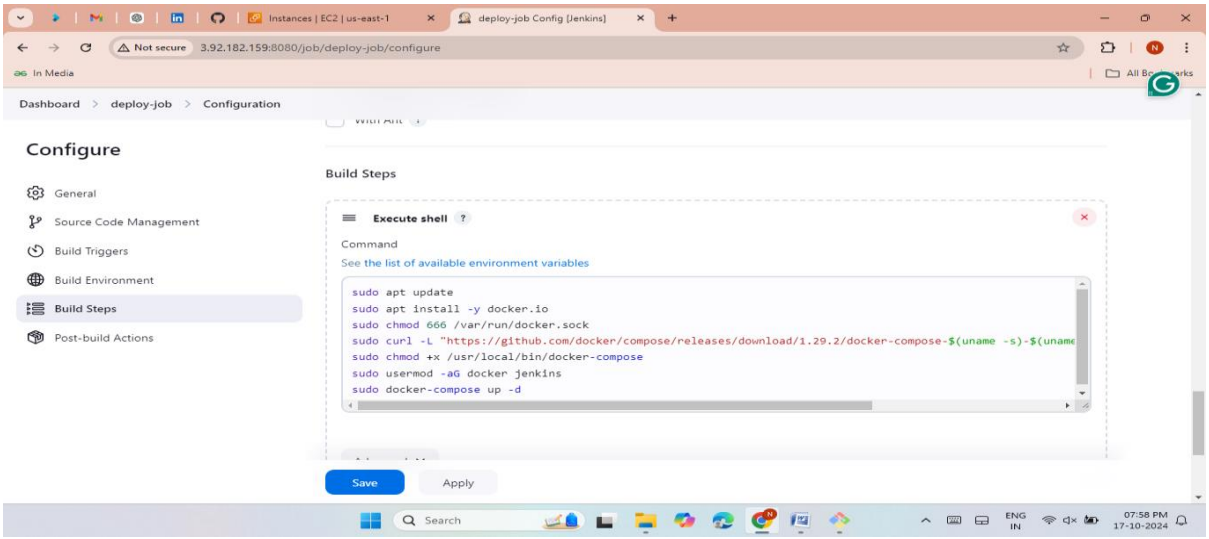
sudo chmod 666 /var/run/docker.sock

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

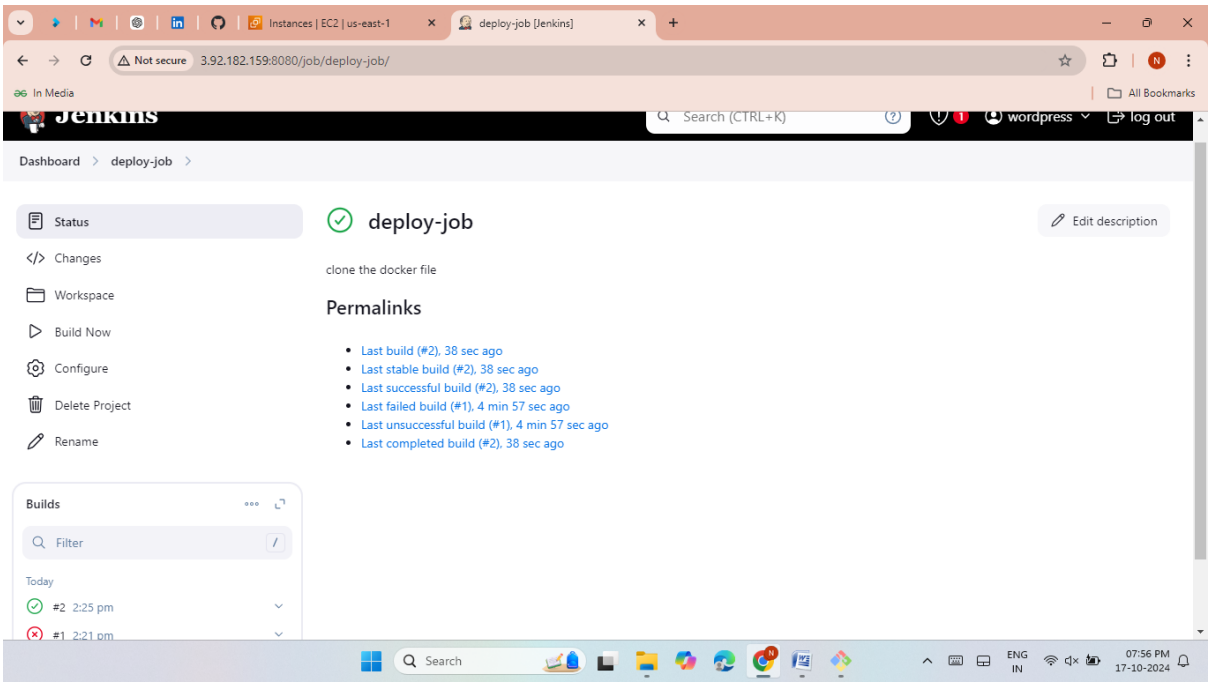
sudo chmod +x /usr/local/bin/docker-compose

sudo usermod -aG docker jenkins

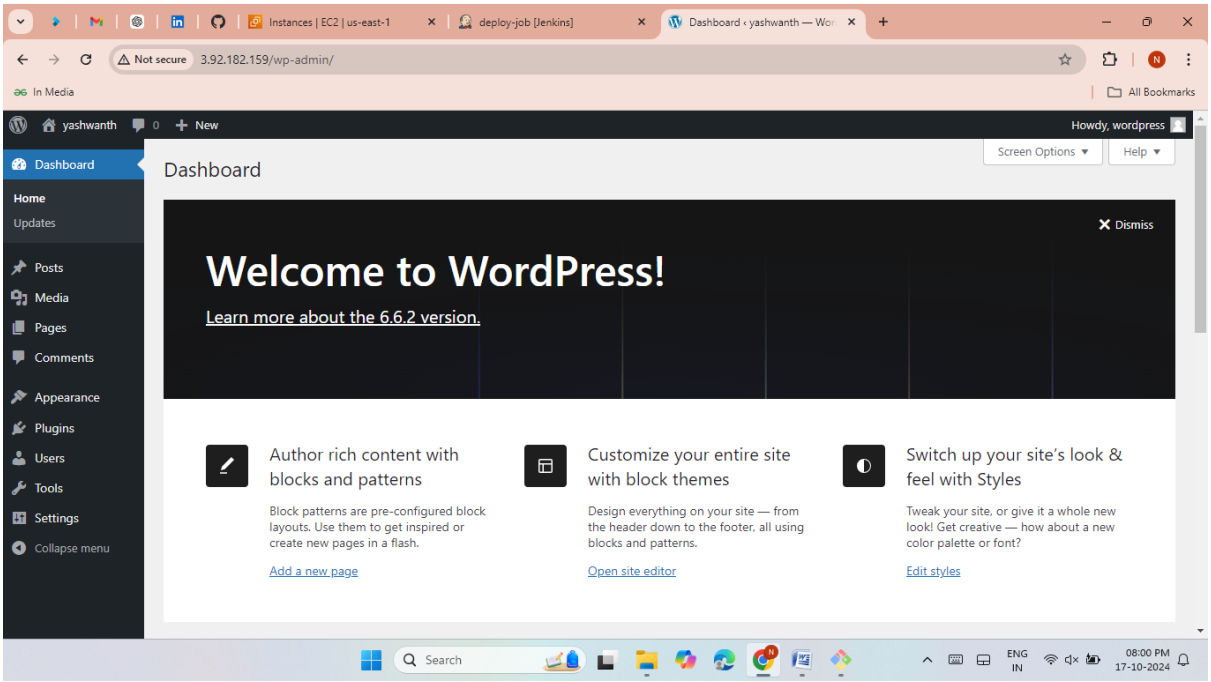
sudo docker-compose up –d



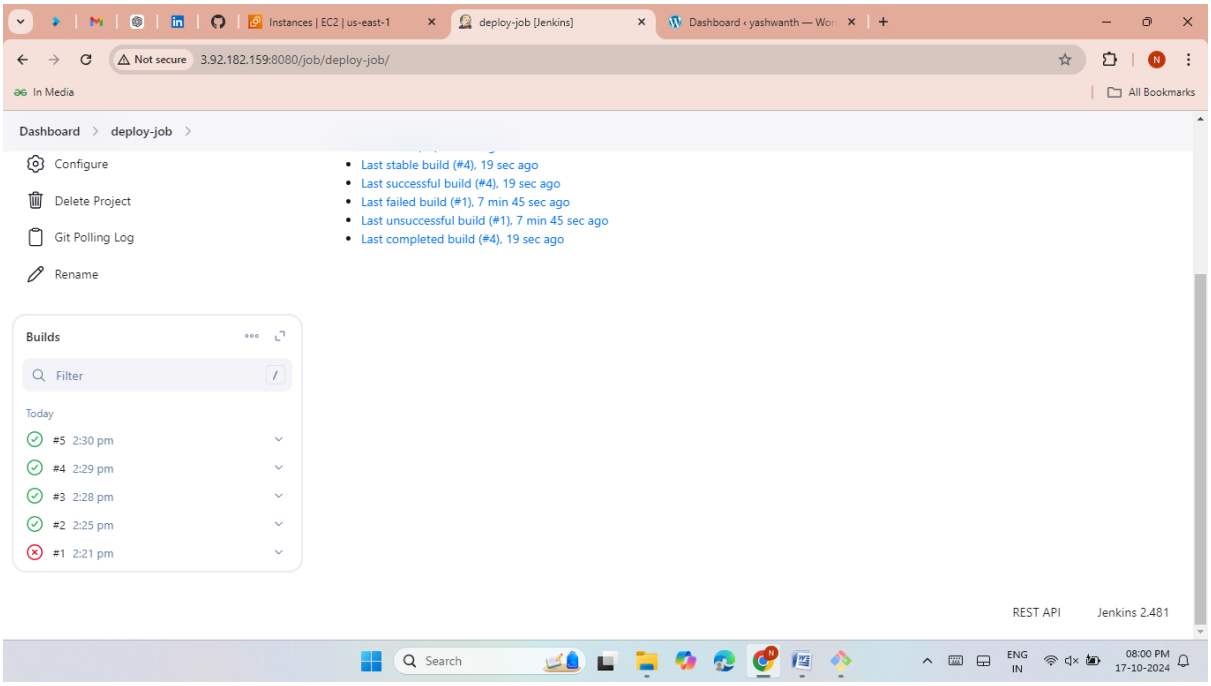
- Here the job was success.



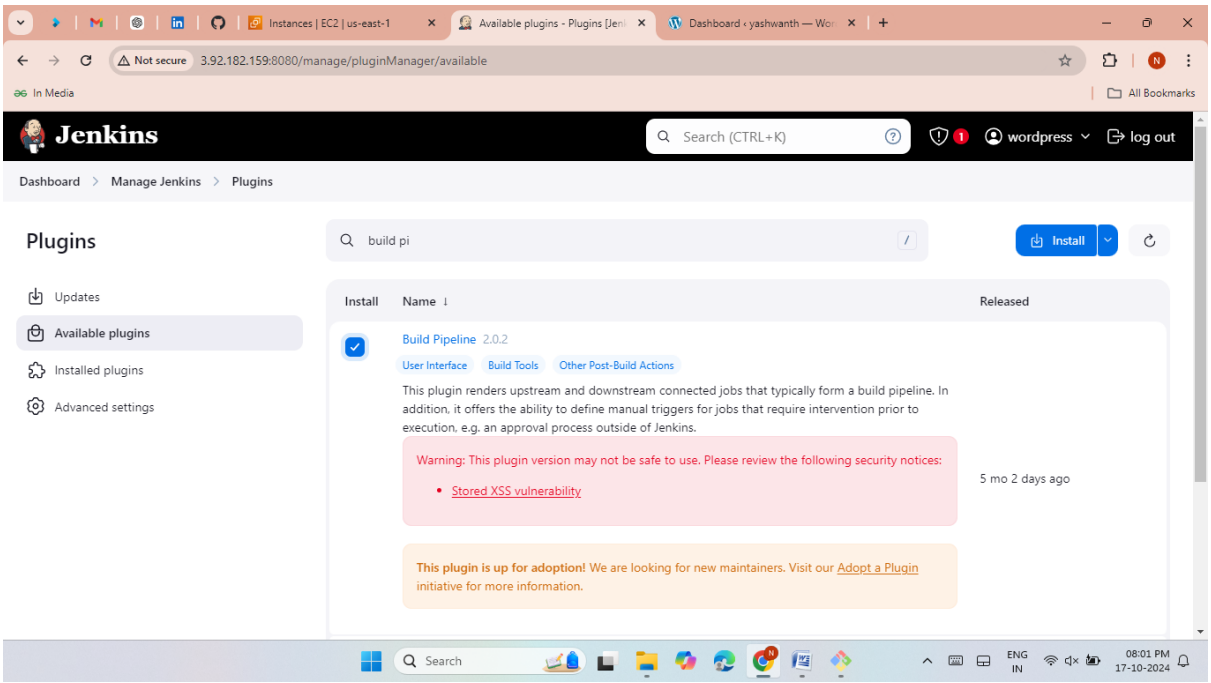
- Now copy that static Public IP and past it in browser.
- Wordpress was launched successfully.



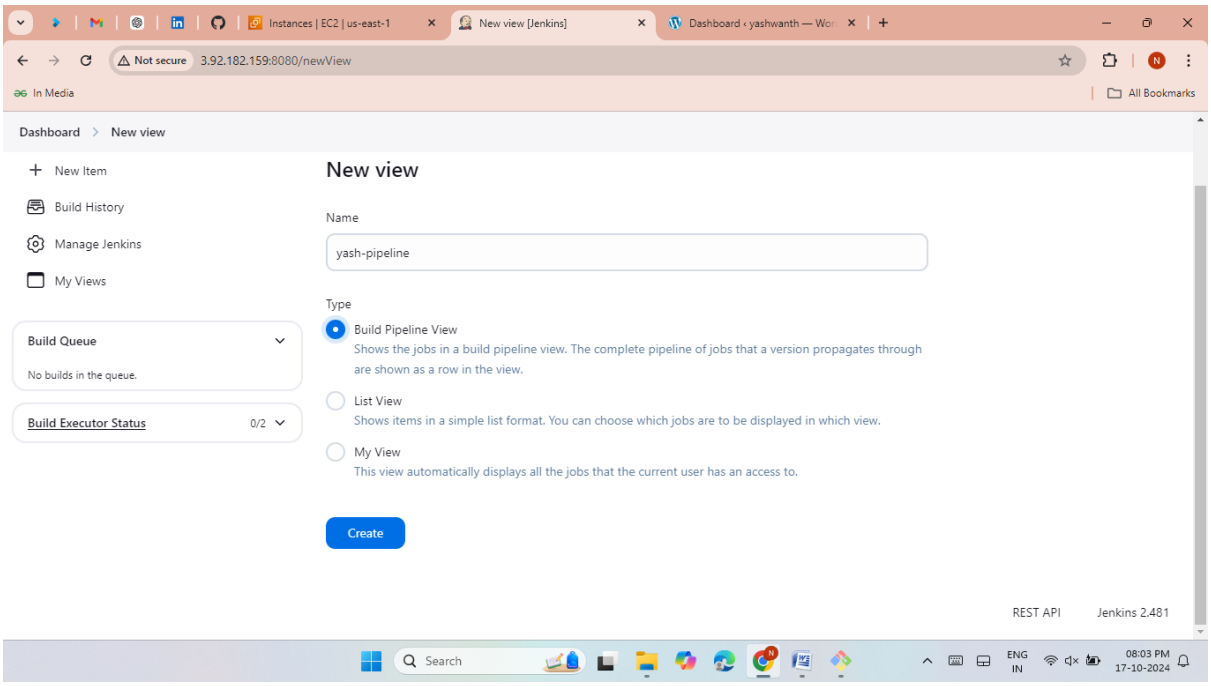
- It runs every MINUTE HOUR DOM(DAY OF MONTH) MONTH DOW(DAY OF WEEK).
- Because I have given (* * * * *).



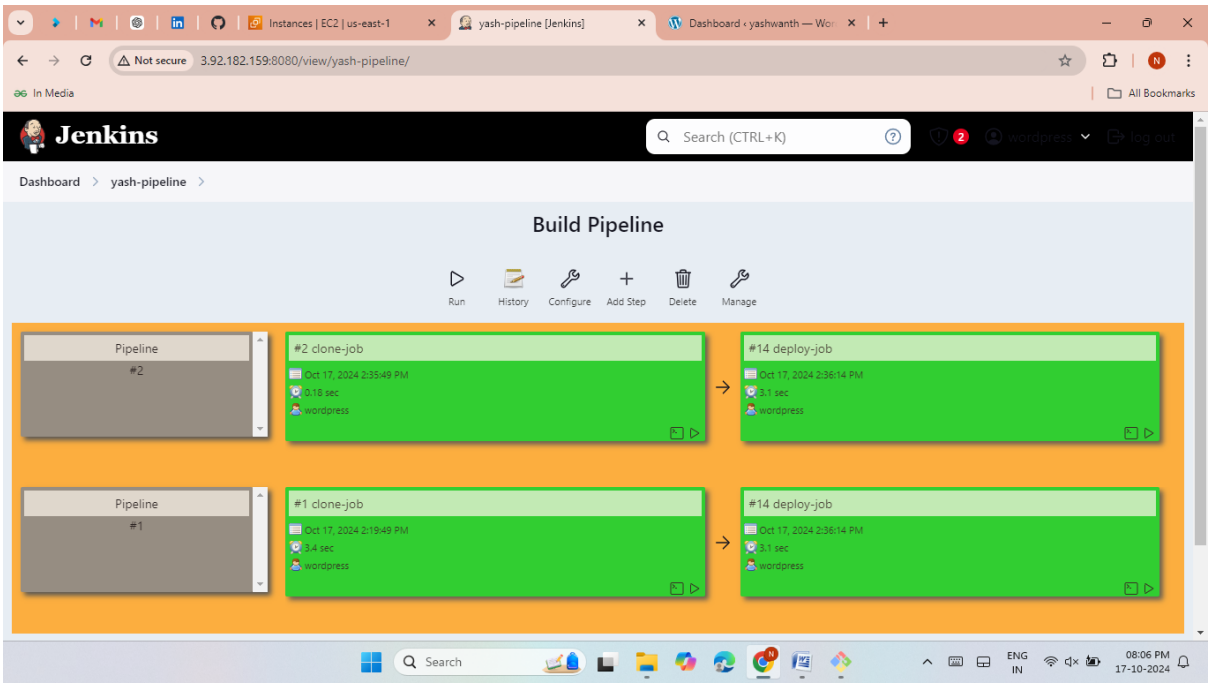
- Now install the build pipeline plugin for create a Jenkins pipeline.



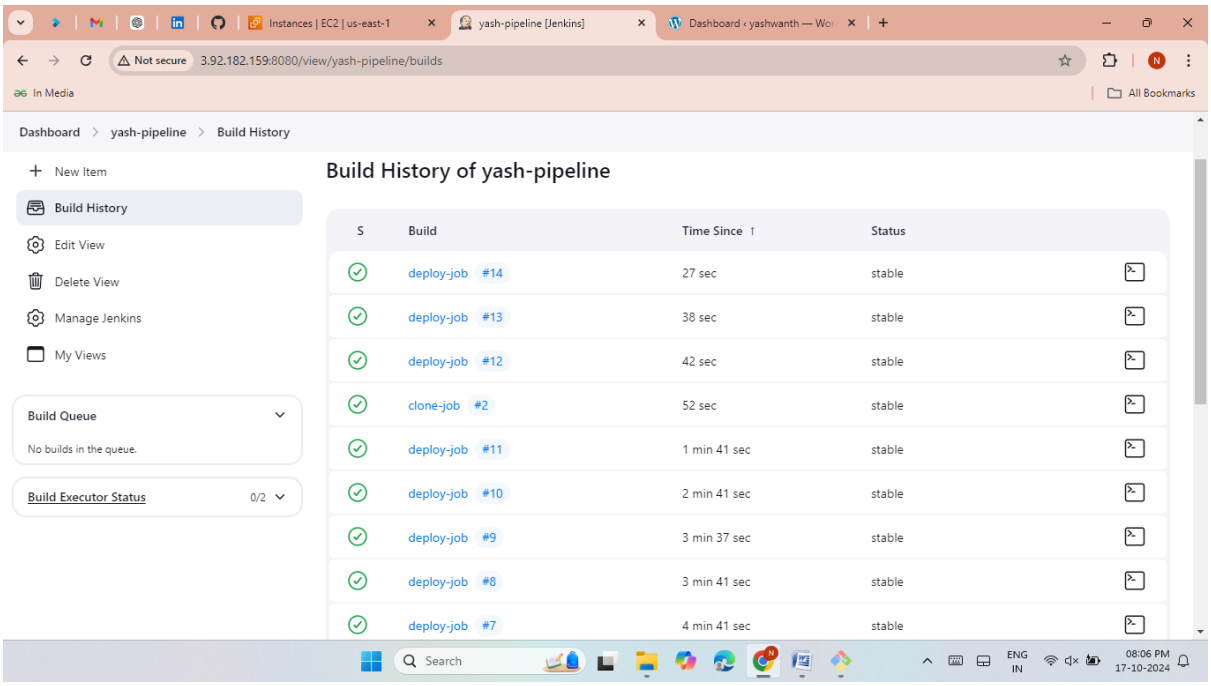
➢ Now create a Jenkins pipeline.



➢ Here the build pipeline is successfully created.



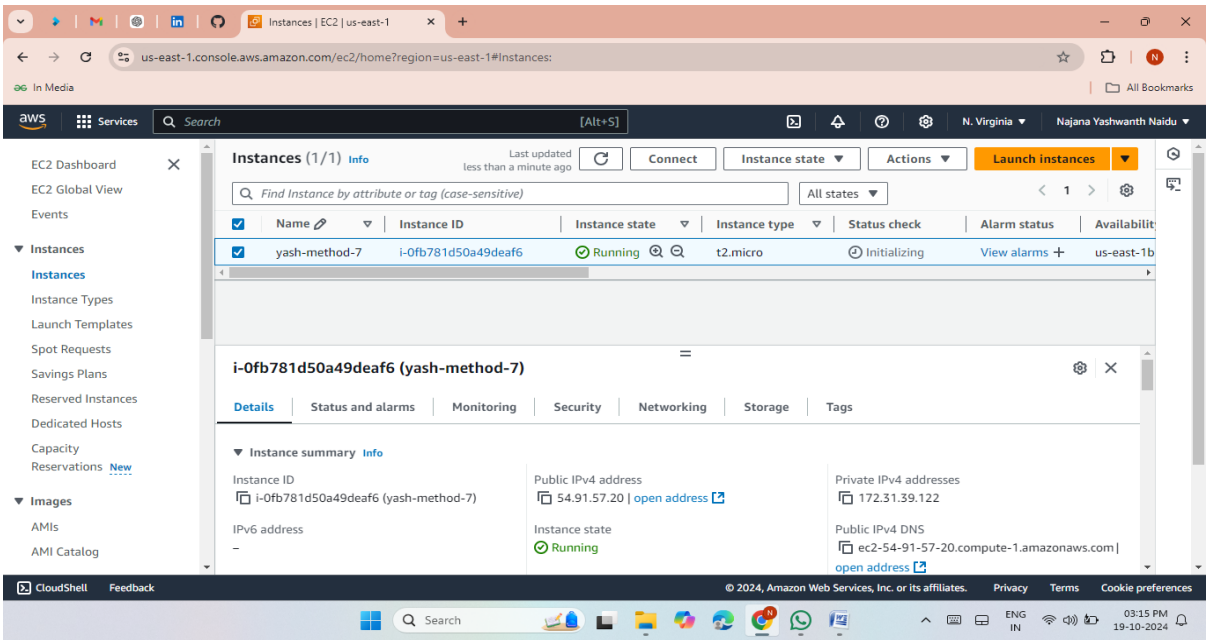
- Now here we can see the Build History.



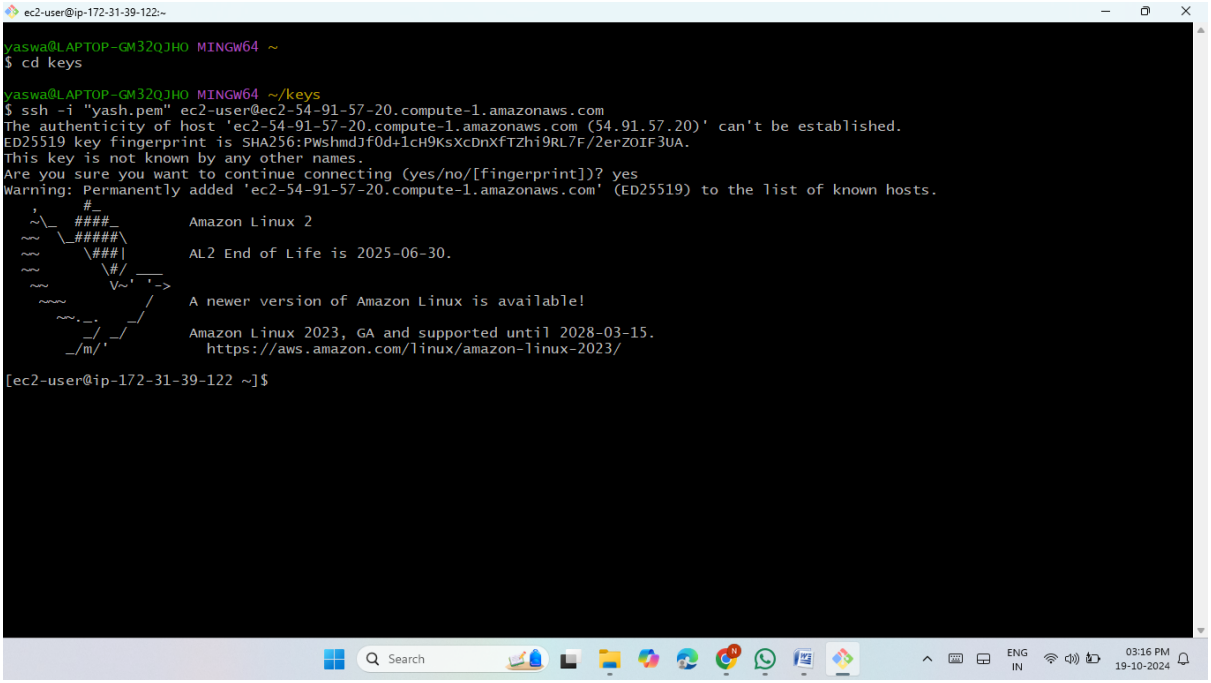
METHOD-7

Deploy WordPress web application by using terraform (create Ec2 instance along with userdata .sh file)

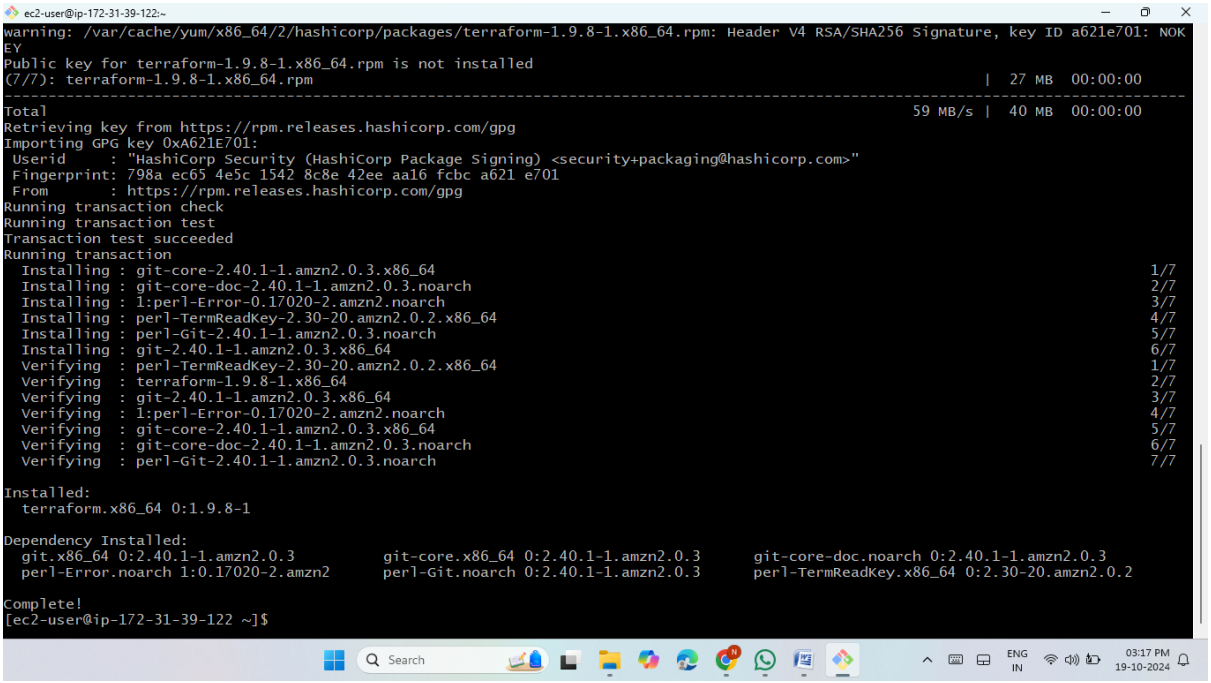
- First Launch an EC2 instance in our AWS account.



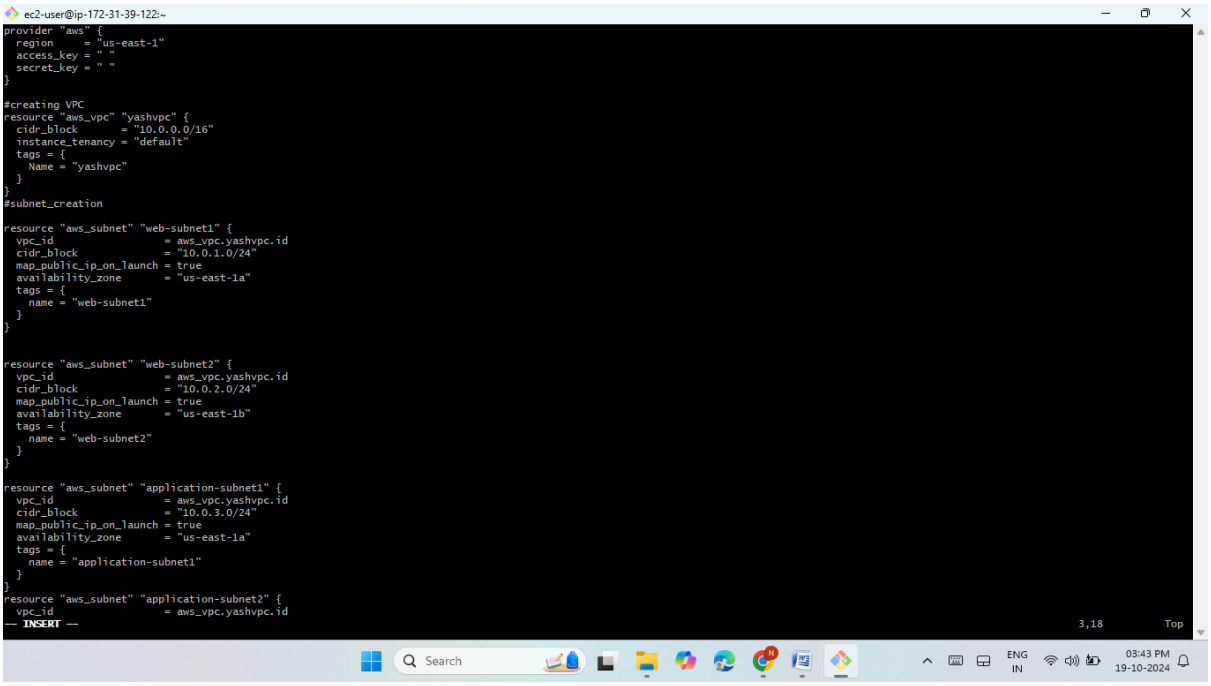
- Now Connect to locally.



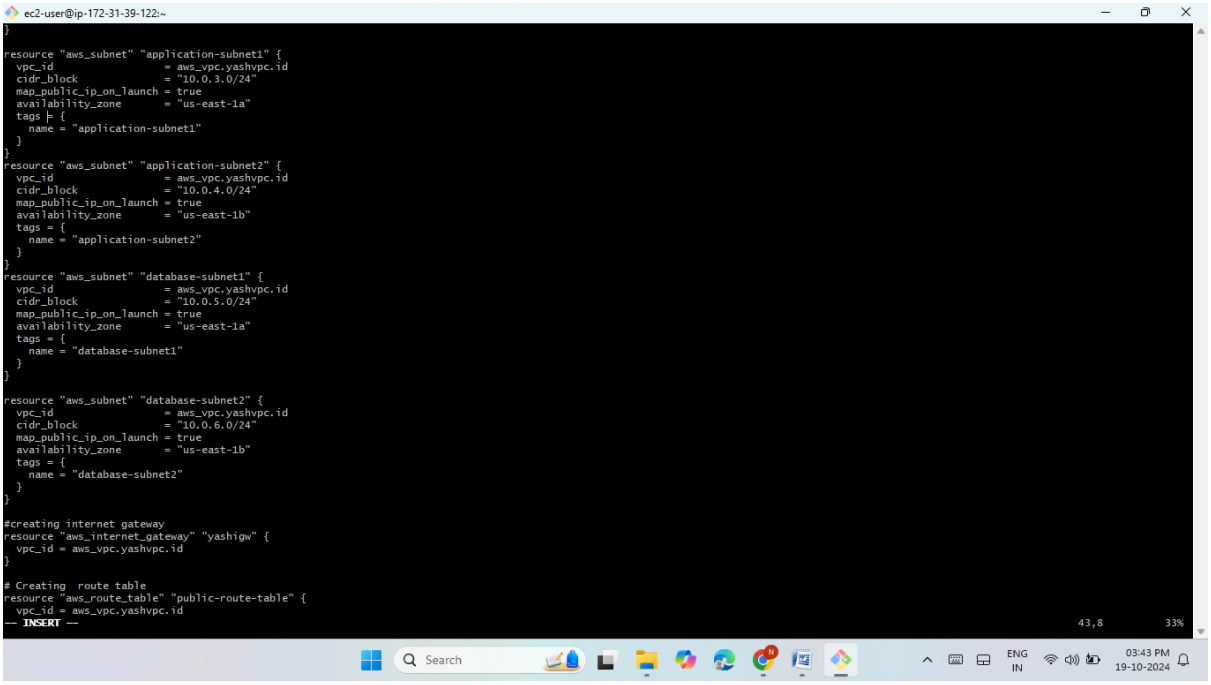
- Now install terraform form Google browser.



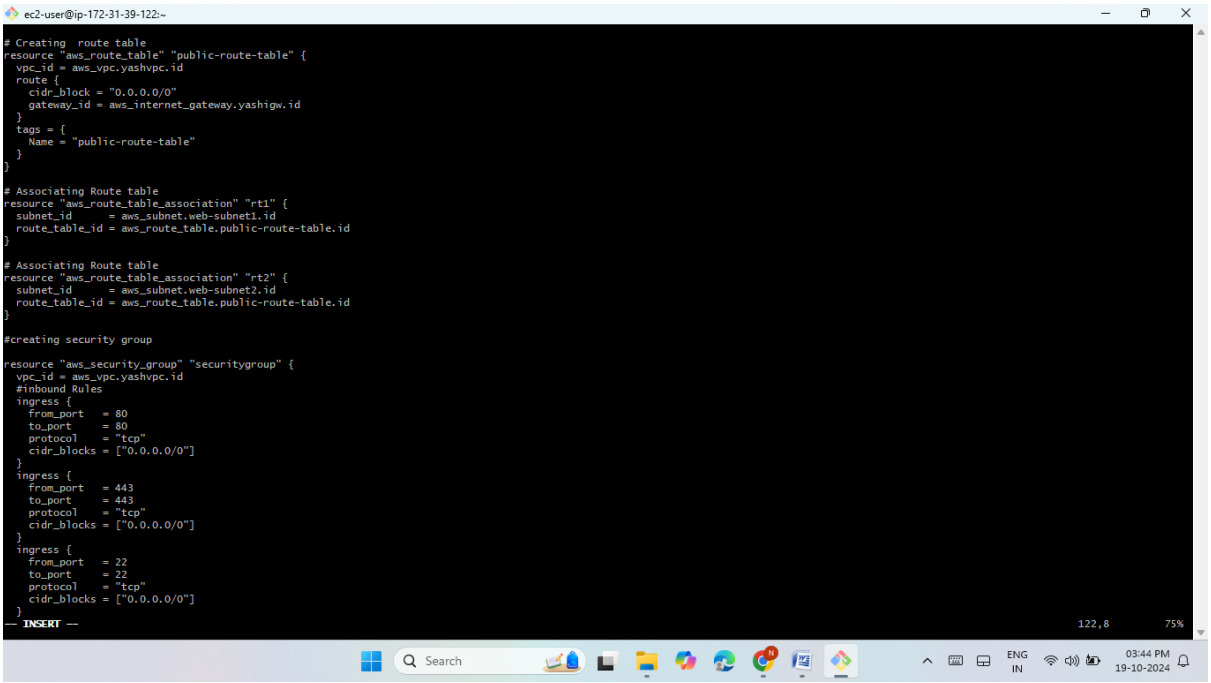
- Write a terraform script for launch an EC2 instance.
- Provide Access key and Secret key.
- Write script for EC2 and Security Groups, VPC, Subnets, Internet Gateway, Route Table.

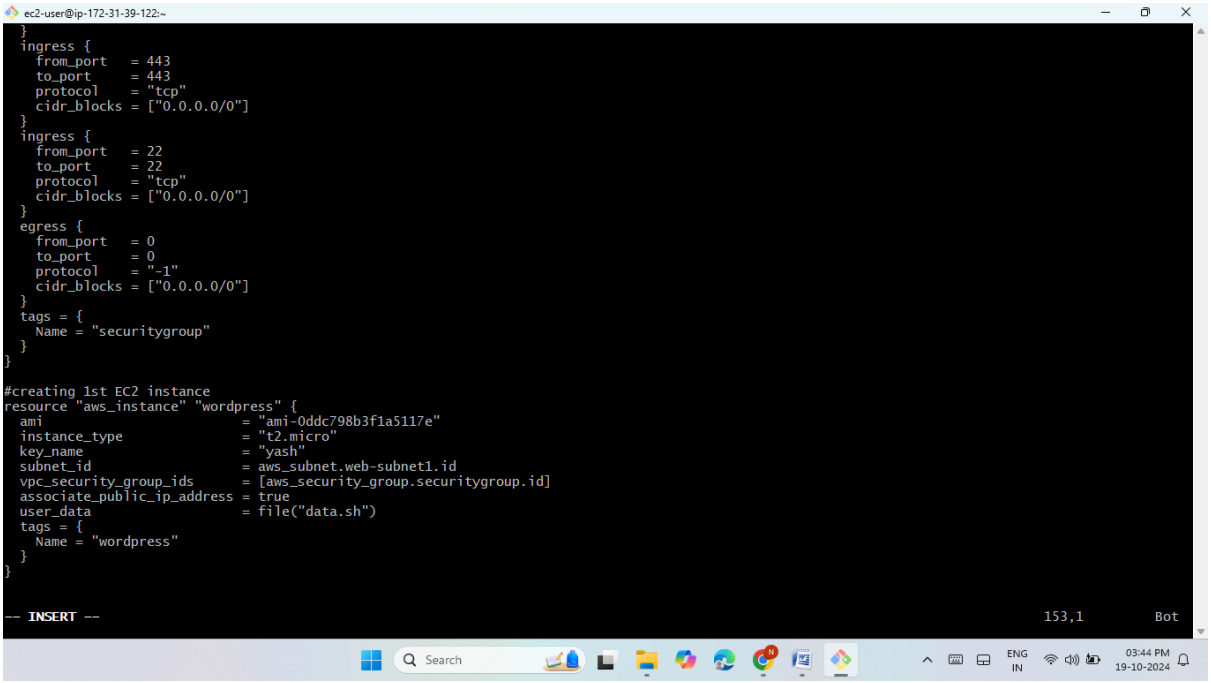


```
ec2-user@ip-172-31-39-122:~
provider "aws" {
  region     = "us-east-1"
  access_key = " "
  secret_key = " "
}

#creating VPC
resource "aws_vpc" "yashvpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"
  tags = {
    Name = "yashvpc"
  }
}
#subnet_creation

resource "aws_subnet" "web-subnet1" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1a"
  tags = {
    name = "web-subnet1"
  }
}


resource "aws_subnet" "web-subnet2" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.2.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1b"
  tags = {
    name = "web-subnet2"
  }
}

resource "aws_subnet" "application-subnet1" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.3.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1a"
  tags = {
    name = "application-subnet1"
  }
}
resource "aws_subnet" "application-subnet2" {
  vpc_id                  = aws_vpc.yashvpc.id
-- INSERT --                                                    3,18          Top
```

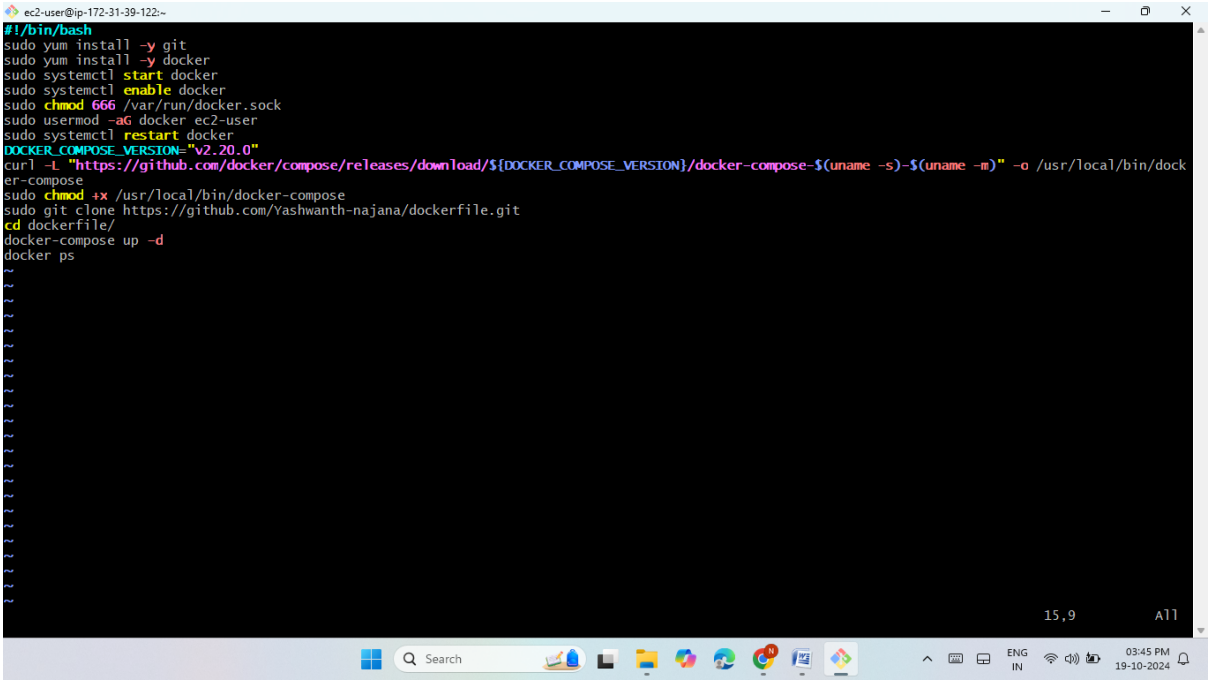


```
ec2-user@ip-172-31-39-122:~
}

resource "aws_subnet" "application-subnet1" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.3.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1a"
  tags = {
    name = "application-subnet1"
  }
}
resource "aws_subnet" "application-subnet2" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.4.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1b"
  tags = {
    name = "application-subnet2"
  }
}
resource "aws_subnet" "database-subnet1" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.5.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1a"
  tags = {
    name = "database-subnet1"
  }
}

resource "aws_subnet" "database-subnet2" {
  vpc_id                  = aws_vpc.yashvpc.id
  cidr_block              = "10.0.6.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1b"
  tags = {
    name = "database-subnet2"
  }
}

#creating internet gateway
resource "aws_internet_gateway" "yashigw" {
  vpc_id = aws_vpc.yashvpc.id
}

# Creating  route table
resource "aws_route_table" "public-route-table" {
  vpc_id = aws_vpc.yashvpc.id
-- INSERT --                                                    43,8          33%
```


ec2-user@ip-172-31-39-122:~
# Creating route table
resource "aws_route_table" "public-route-table" {
vpc_id = aws_vpc.yashvpc.id
route {
cidr_block = "0.0.0.0/0"
gateway_id = aws_internet_gateway.yashigw.id
}
tags = {
Name = "public-route-table"
}
}
# Associating Route table
resource "aws_route_table_association" "rt1" {
subnet_id = aws_subnet.web-subnet1.id
route_table_id = aws_route_table.public-route-table.id
}
# Associating Route table
resource "aws_route_table_association" "rt2" {
subnet_id = aws_subnet.web-subnet2.id
route_table_id = aws_route_table.public-route-table.id
}
#creating security group
resource "aws_security_group" "securitygroup" {
vpc_id = aws_vpc.yashvpc.id
#inbound Rules
ingress {
from_port = 80
to_port = 80
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
from_port = 443
to_port = 443
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
-- INSERT --
122,8
75%
Search
ENG
IN
03:44 PM
19-10-2024


ec2-user@ip-172-31-39-122:~
}
ingress {
from_port = 443
to_port = 443
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
egress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}
tags = {
Name = "securitygroup"
}
}
#creating 1st EC2 instance
resource "aws_instance" "wordpress" {
ami = "ami-0ddc798b3f1a5117e"
instance_type = "t2.micro"
key_name = "yash"
subnet_id = aws_subnet.web-subnet1.id
vpc_security_group_ids = [aws_security_group.securitygroup.id]
associate_public_ip_address = true
user_data = file("data.sh")
tags = {
Name = "wordpress"
}
}
-- INSERT --
153,1
Bot
Search
ENG
IN
03:44 PM
19-10-2024

- Now write the script for data file.
- Clone the docker-compose file form github.



- Now enter the following commands

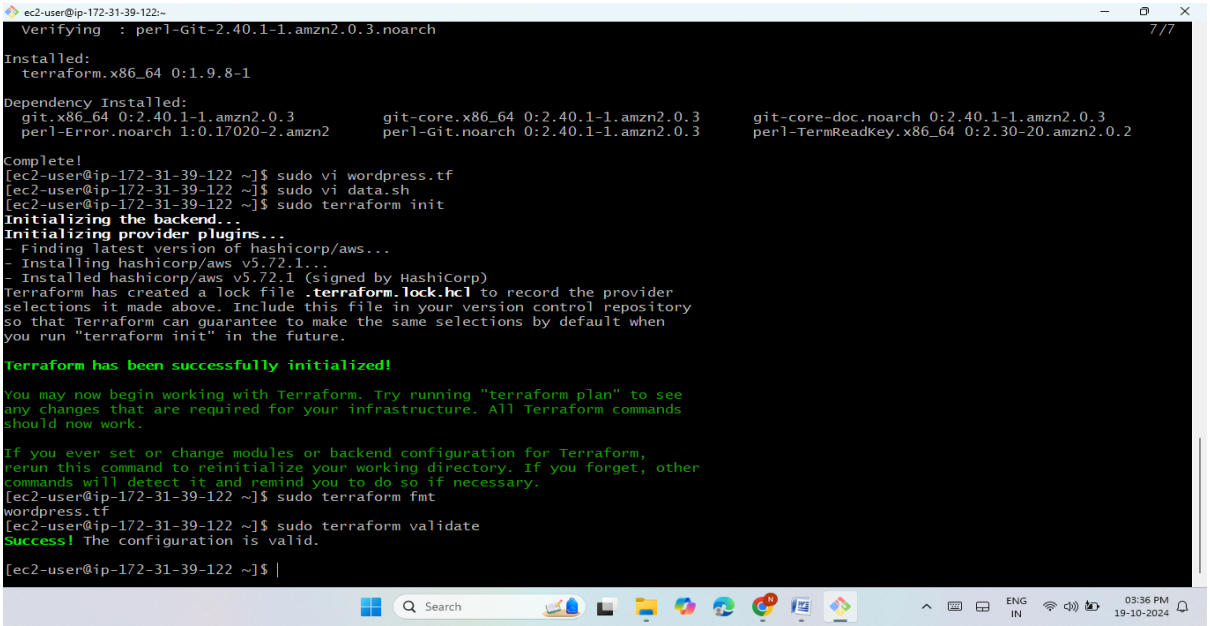
terraform init

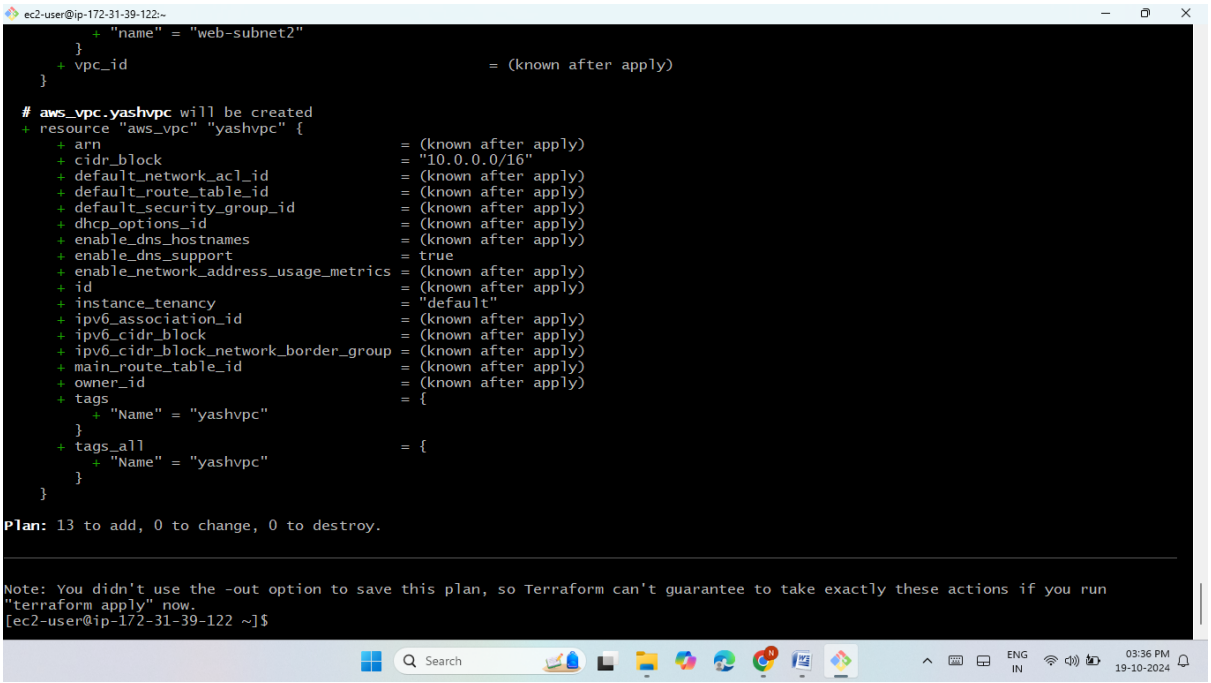
terraform fmt

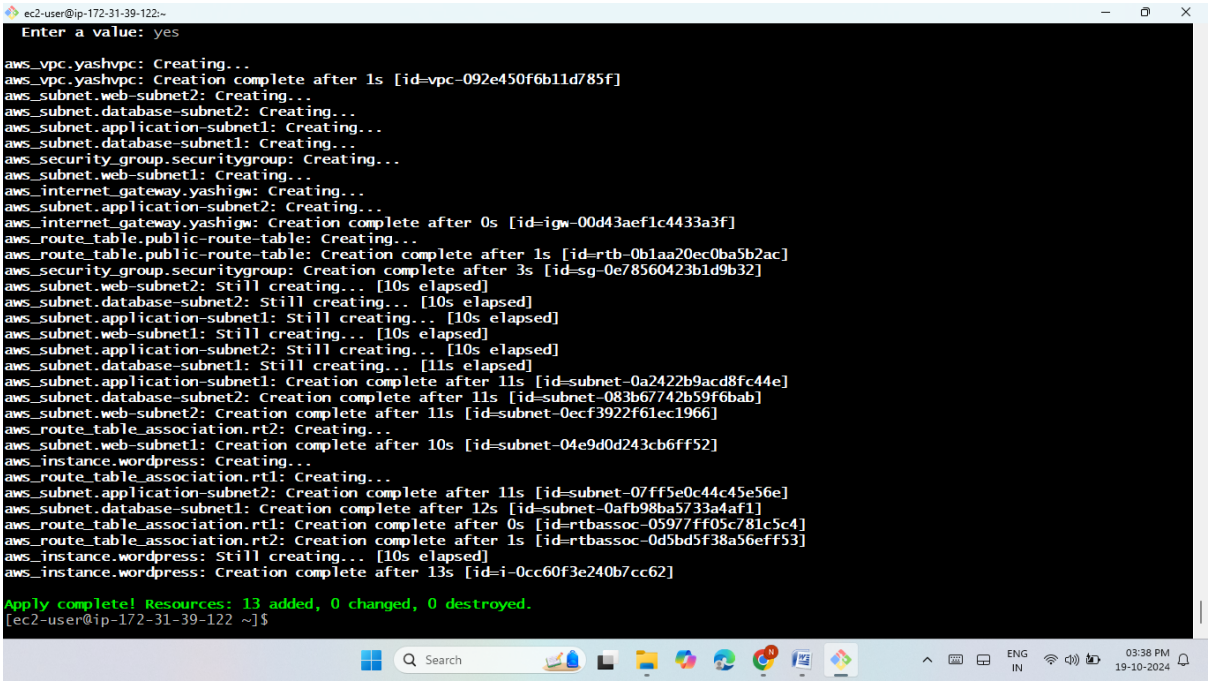
terraform validate

terraform plan

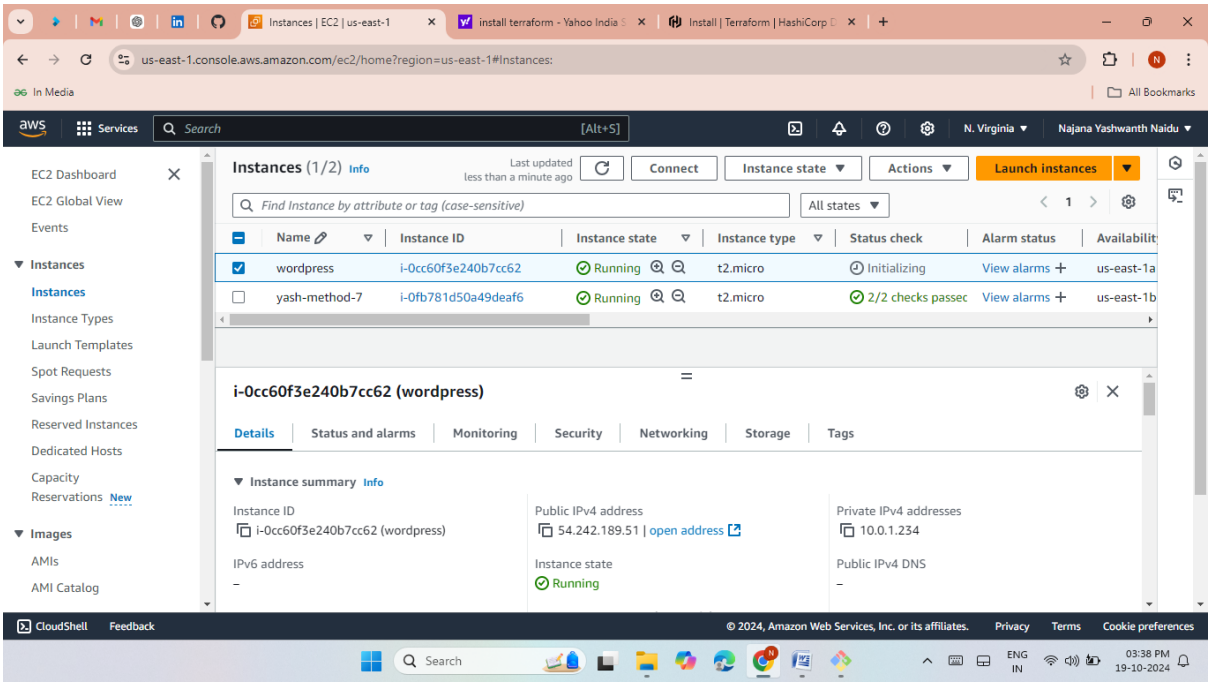
terraform apply



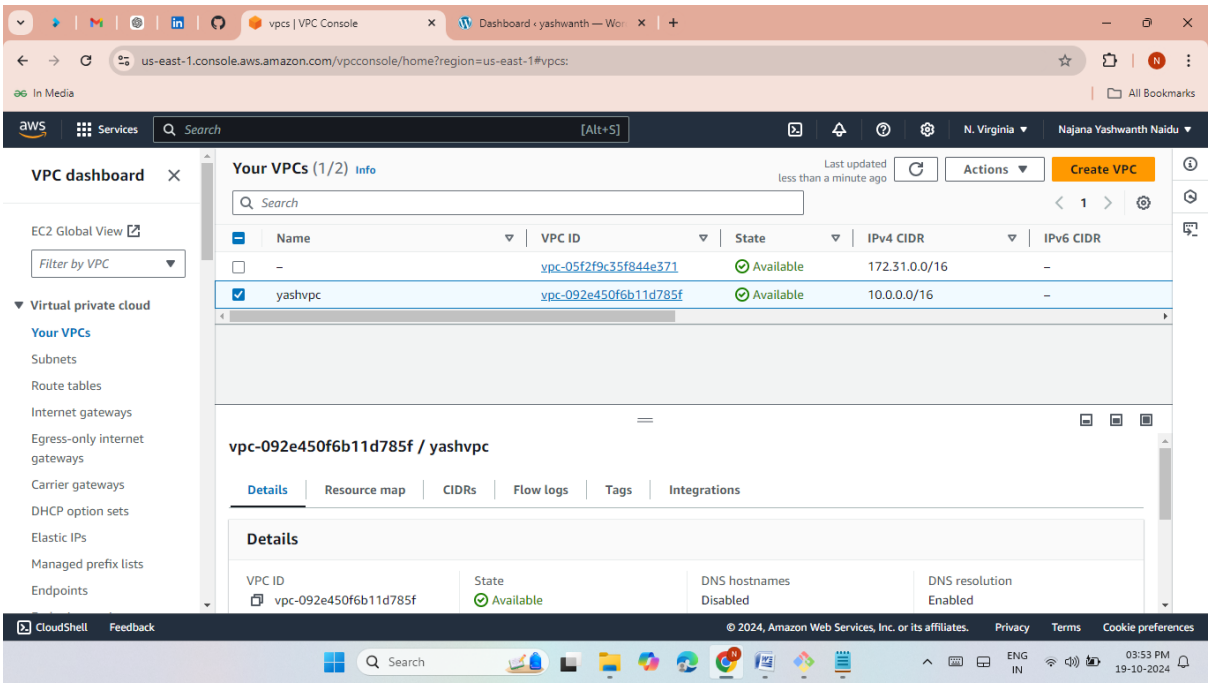

ec2-user@ip-172-31-39-122:~
+ "name" = "web-subnet2"
}
+ vpc_id = (known after apply)
}
# aws_vpc.yashvpc will be created
+ resource "aws_vpc" "yashvpc" {
+ arn = (known after apply)
+ cidr_block = "10.0.0.0/16"
+ default_network_acl_id = (known after apply)
+ default_route_table_id = (known after apply)
+ default_security_group_id = (known after apply)
+ dhcp_options_id = (known after apply)
+ enable_dns_hostnames = (known after apply)
+ enable_dns_support = true
+ enable_network_address_usage_metrics = (known after apply)
+ id = (known after apply)
+ instance_tenancy = "default"
+ ipv6_association_id = (known after apply)
+ ipv6_cidr_block = (known after apply)
+ ipv6_cidr_block_network_border_group = (known after apply)
+ main_route_table_id = (known after apply)
+ owner_id = (known after apply)
+ tags = {
+ "Name" = "yashvpc"
}
+ tags_all = {
+ "Name" = "yashvpc"
}
}
Plan: 13 to add, 0 to change, 0 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
[ec2-user@ip-172-31-39-122 ~]$
Search
ENG IN
03:36 PM 19-10-2024


ec2-user@ip-172-31-39-122:~
Enter a value: yes
aws_vpc.yashvpc: Creating...
aws_vpc.yashvpc: Creation complete after 1s [id=vpc-092e450f6b11d785f]
aws_subnet.web-subnet2: Creating...
aws_subnet.database-subnet2: Creating...
aws_subnet.application-subnet1: Creating...
aws_subnet.database-subnet1: Creating...
aws_security_group.securitygroup: Creating...
aws_subnet.web-subnet1: Creating...
aws_internet_gateway.yashigw: Creating...
aws_subnet.application-subnet2: Creating...
aws_internet_gateway.yashigw: Creation complete after 0s [id=igw-00d43aef1c4433a3f]
aws_route_table.public-route-table: Creating...
aws_route_table.public-route-table: Creation complete after 1s [id=rtb-0b1aa20ec0ba5b2ac]
aws_security_group.securitygroup: Creation complete after 3s [id=sg-0e78560423b1d9b32]
aws_subnet.web-subnet2: Still creating... [10s elapsed]
aws_subnet.database-subnet2: Still creating... [10s elapsed]
aws_subnet.application-subnet1: Still creating... [10s elapsed]
aws_subnet.web-subnet1: Still creating... [10s elapsed]
aws_subnet.application-subnet2: Still creating... [10s elapsed]
aws_subnet.database-subnet1: Still creating... [11s elapsed]
aws_subnet.application-subnet1: Creation complete after 11s [id=subnet-0a2422b9acd8fc44e]
aws_subnet.database-subnet2: Creation complete after 11s [id=subnet-083b67742b59f6bab]
aws_subnet.web-subnet2: Creation complete after 11s [id=subnet-0ecf3922f61ec1966]
aws_route_table_association.rt2: Creating...
aws_subnet.web-subnet1: Creation complete after 10s [id=subnet-04e9d0d243cb6ff52]
aws_instance.wordpress: Creating...
aws_route_table_association.rt1: Creating...
aws_subnet.application-subnet2: Creation complete after 11s [id=subnet-07ff5e0c44c45e56e]
aws_subnet.database-subnet1: Creation complete after 12s [id=subnet-0afb98ba5733a4af1]
aws_route_table_association.rt1: Creation complete after 0s [id=rtbassoc-05977ff05c781c5c4]
aws_route_table_association.rt2: Creation complete after 1s [id=rtbassoc-0d5bd5f38a56eff53]
aws_instance.wordpress: Still creating... [10s elapsed]
aws_instance.wordpress: Creation complete after 13s [id=i-0cc60f3e240b7cc62]
Apply complete! Resources: 13 added, 0 changed, 0 destroyed.
[ec2-user@ip-172-31-39-122 ~]$
Search
ENG IN
03:38 PM 19-10-2024

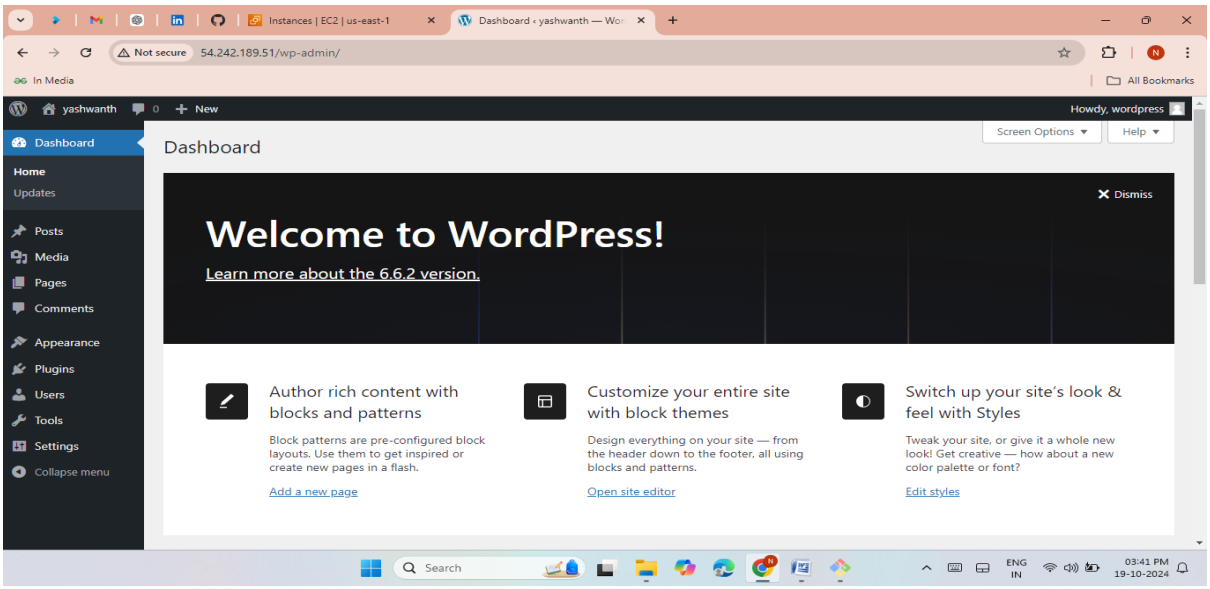
- Here the EC2 instance was launched.
- Past the instance static Public IP in browser.



- Here the VPC, Subnet, Internet Gateway, Route Table, Security Gropus also created.



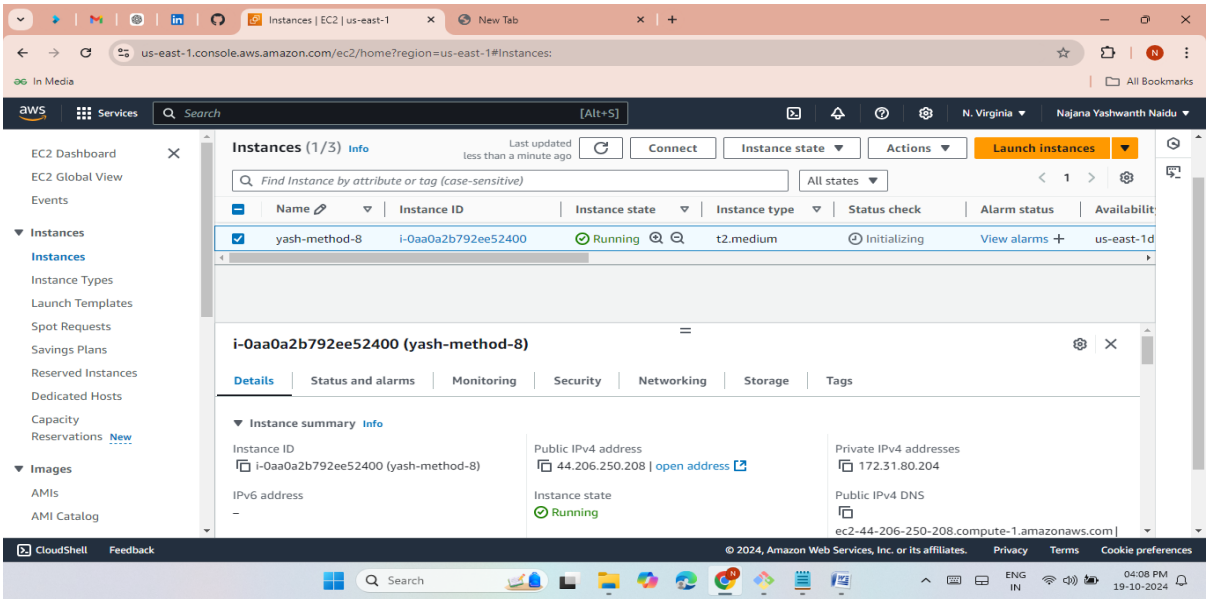
- Here the wordpress was launched.



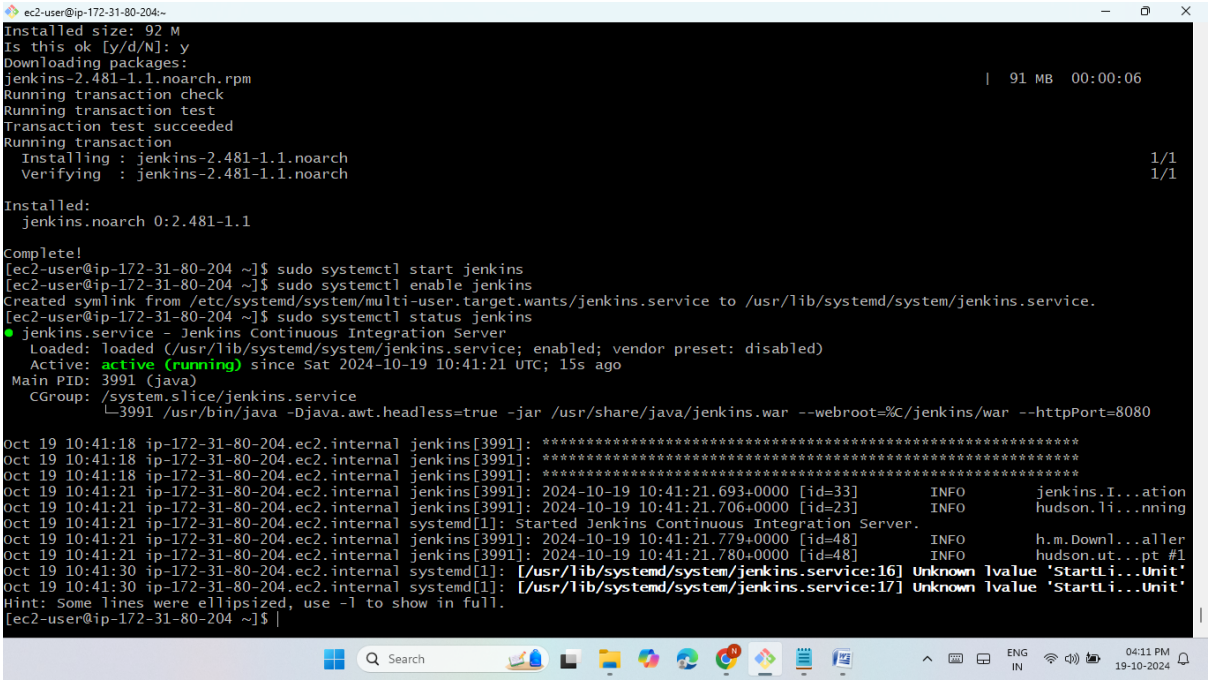
METHOD-8

Deploy WordPress web application by using git (clone terraform script which helps to deploy WordPress web application), jenkins (in execute shell install terraform, init, fmt, validate and apply with automatic command as terraform apply --auto-approve) and terraform.

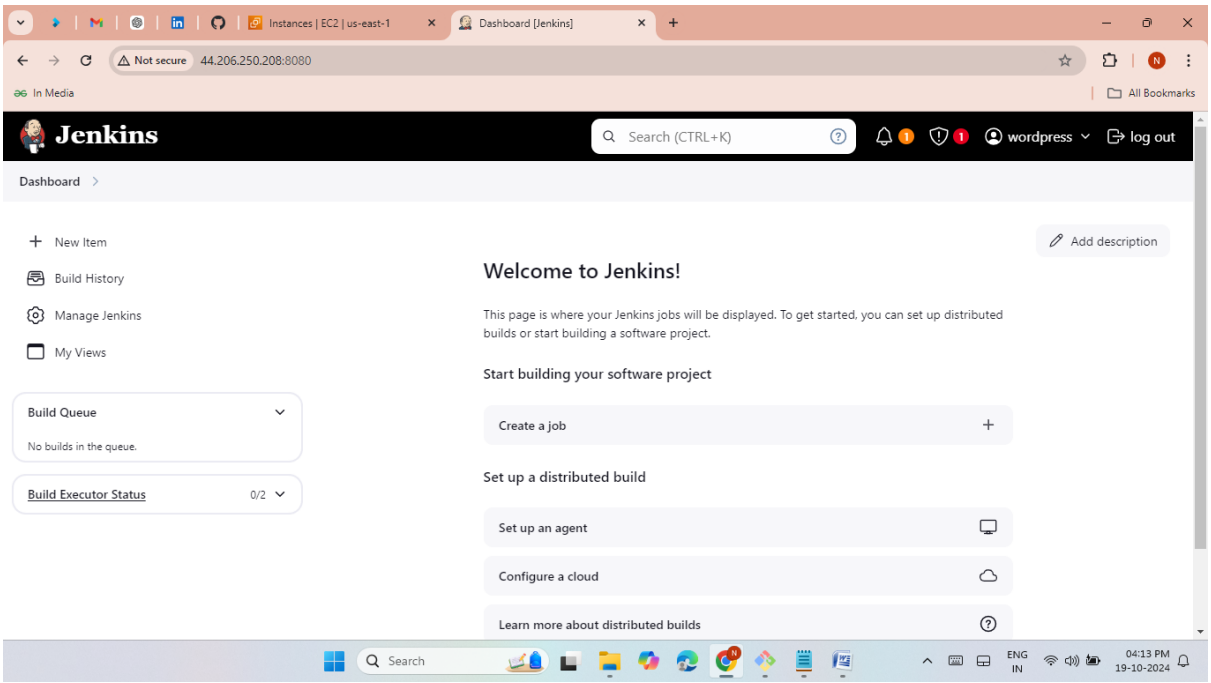
- First launch the EC2 instance with the Jenkins port number(8080).



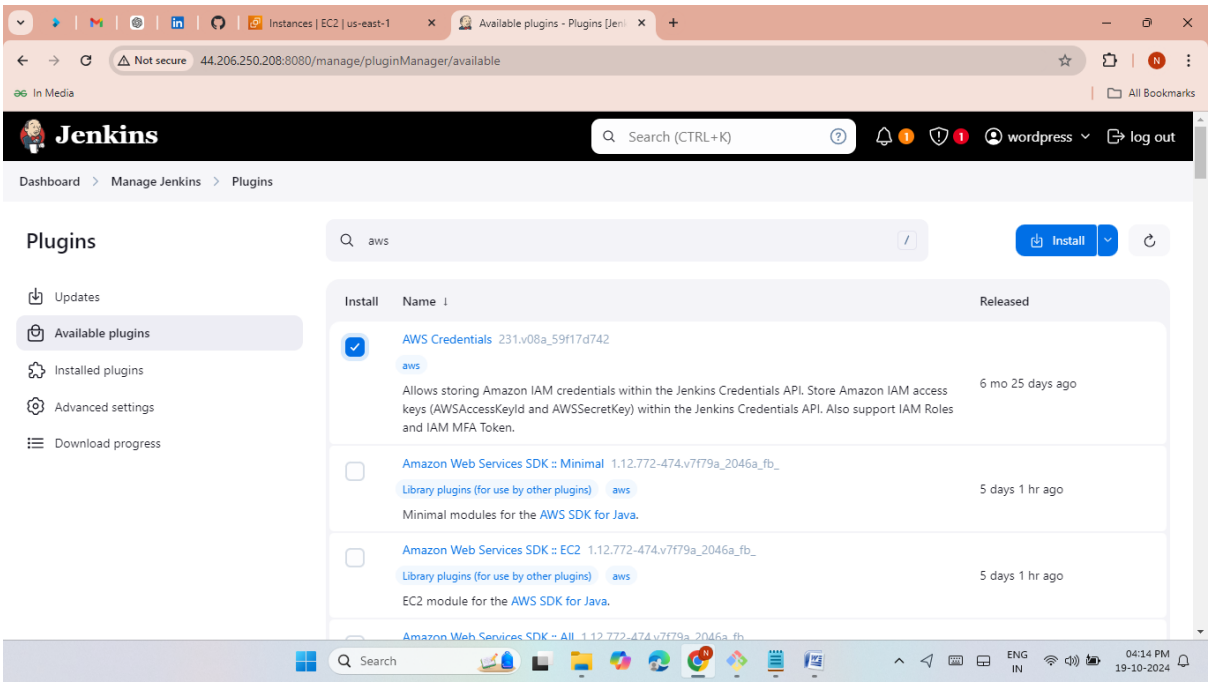
- Now Install git and Jenkins.
- Now start and enable the Jenkins.



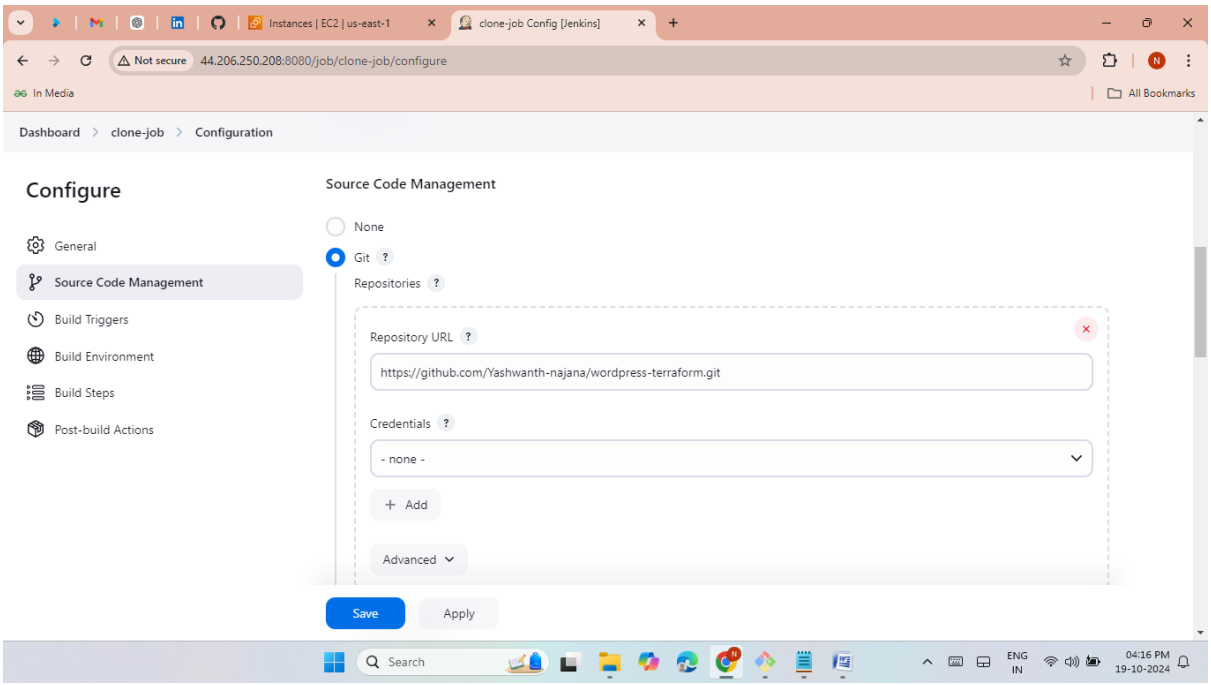
- Here the Jenkins was launched successfully.



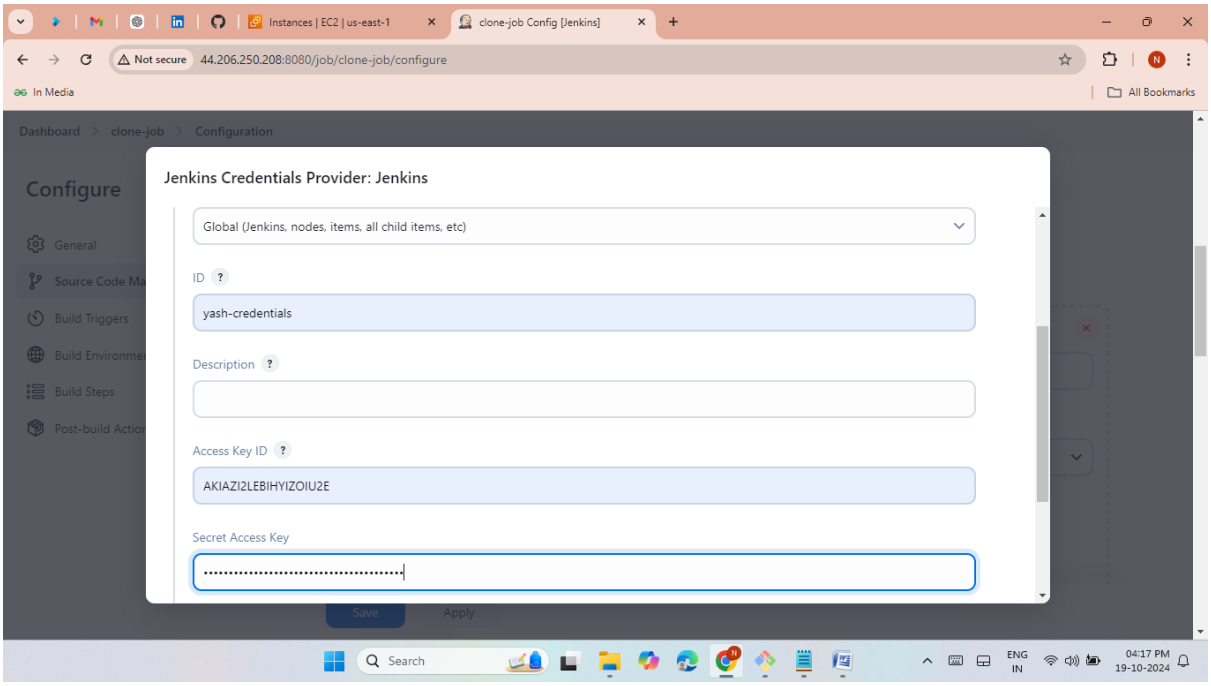
➢ Now install the AWS credentials plugin.



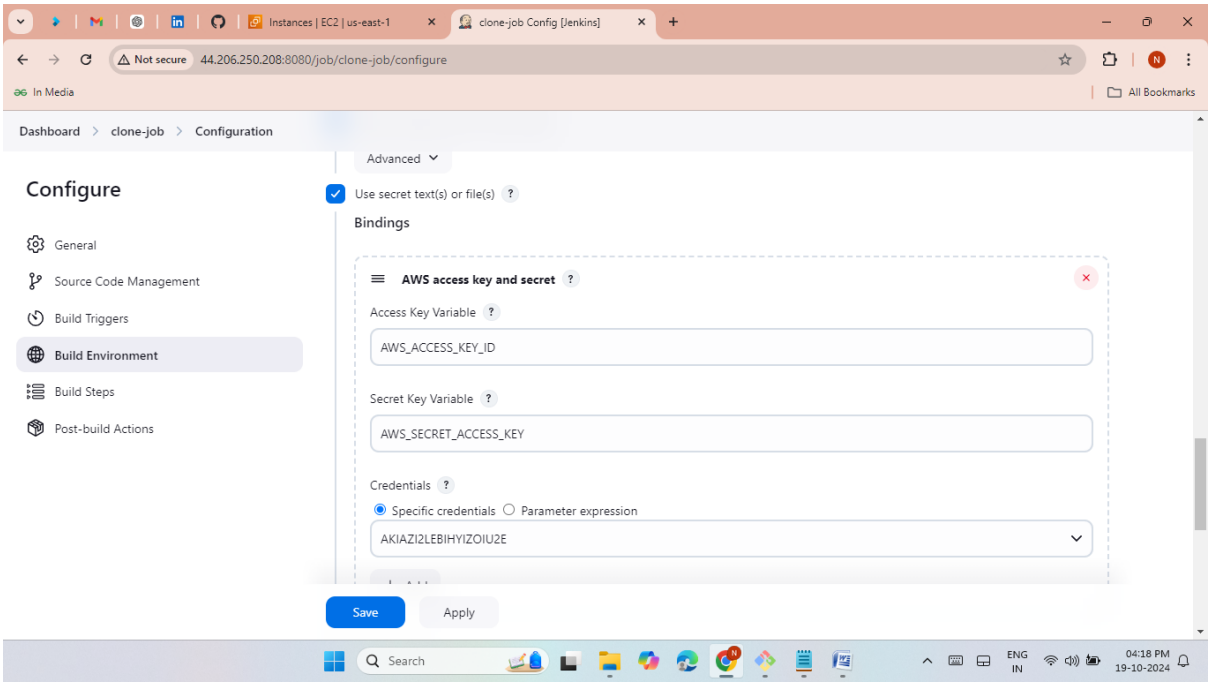
➢ Now create a clone-job and clone the terraform script form github.



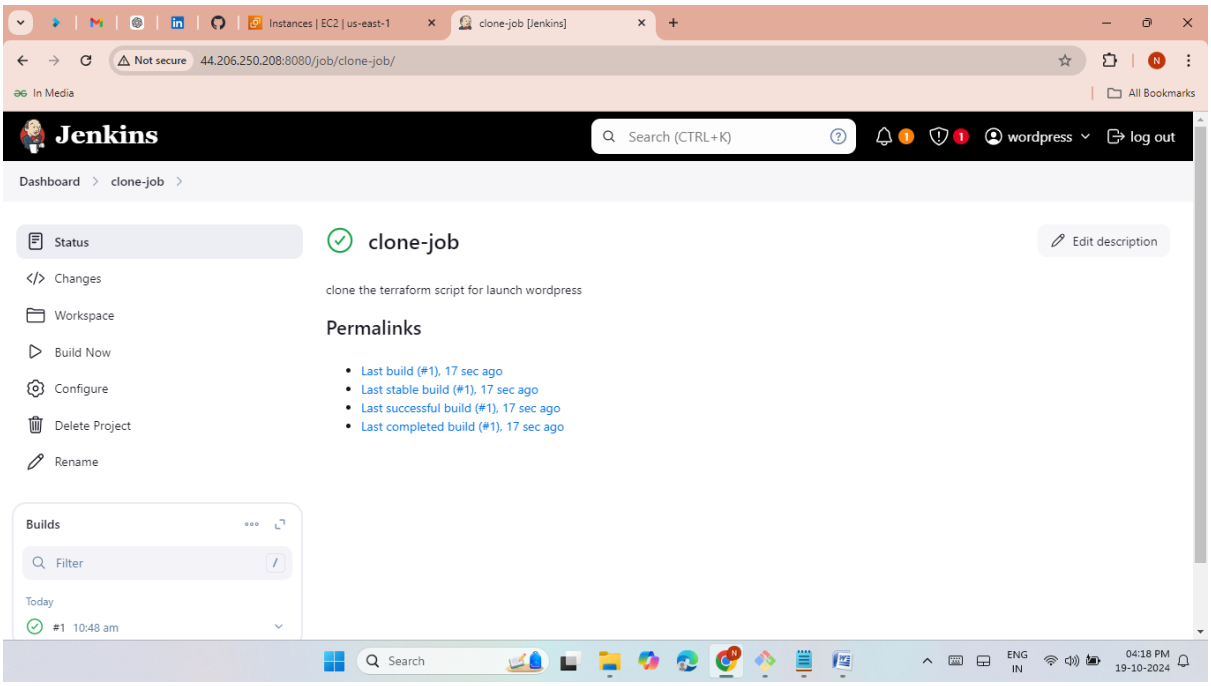
- Then create the credentials.
- Add AWS credentials(access key and secret key).



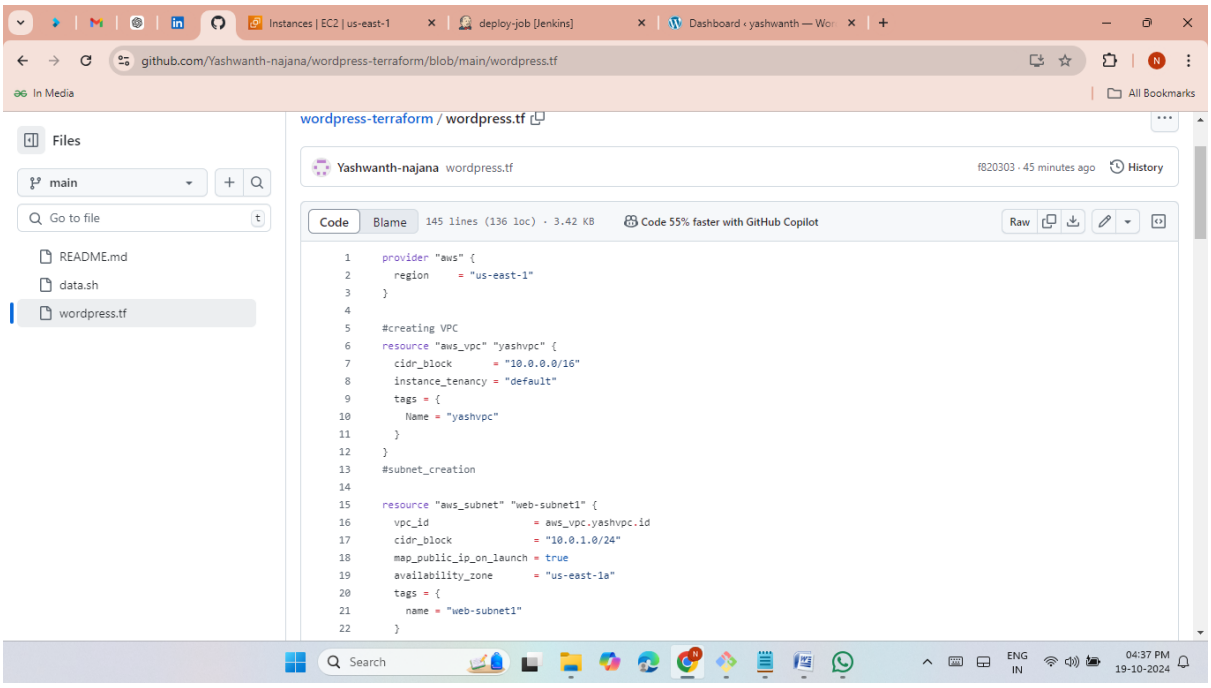
- Now add the AWS access key and secret key.
- Now apply the changes.



➢ Here the clone-job is success.

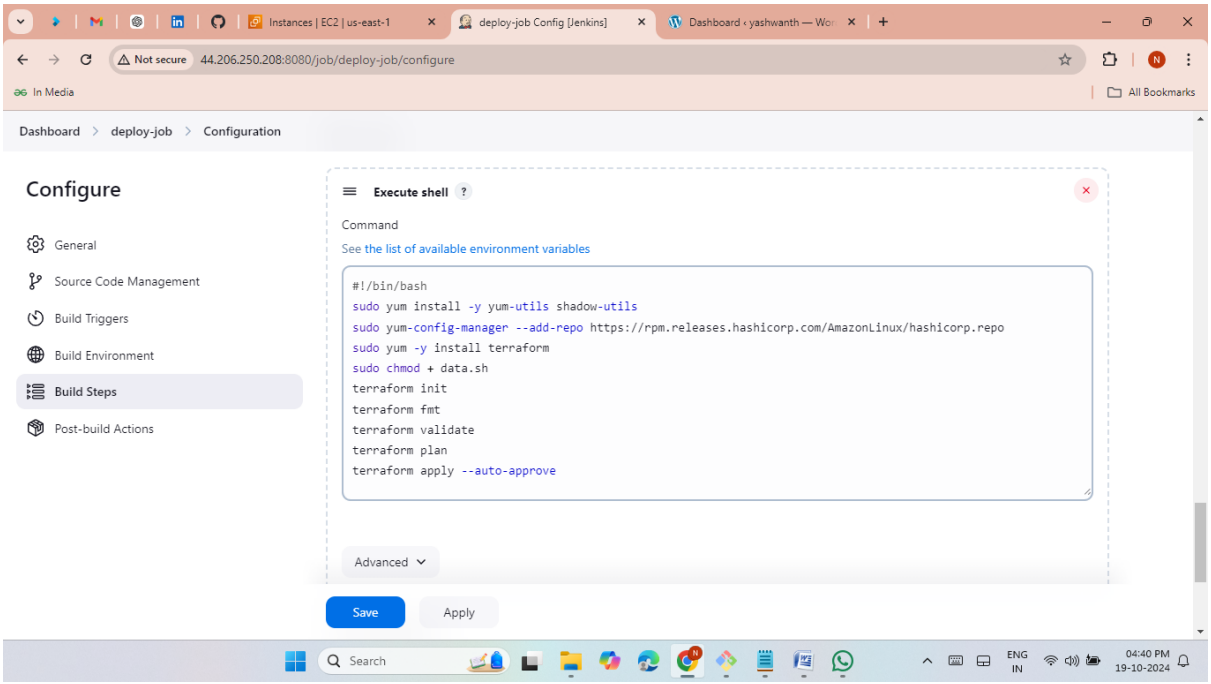


➢ This is the terraform script.



- Now create a deploy-job and copy the clone-job.
- Now install the terraform in execute shell.
- Now add the terraform commands.

#!/bin/bash
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
sudo chmod + data.sh
terraform init
terraform fmt
terraform validate
terraform plan
terraform apply --auto-approve



- Edit the **sudoers** File:
- You need to add the Jenkins user to the **sudoers** file to allow it to execute commands without a password.

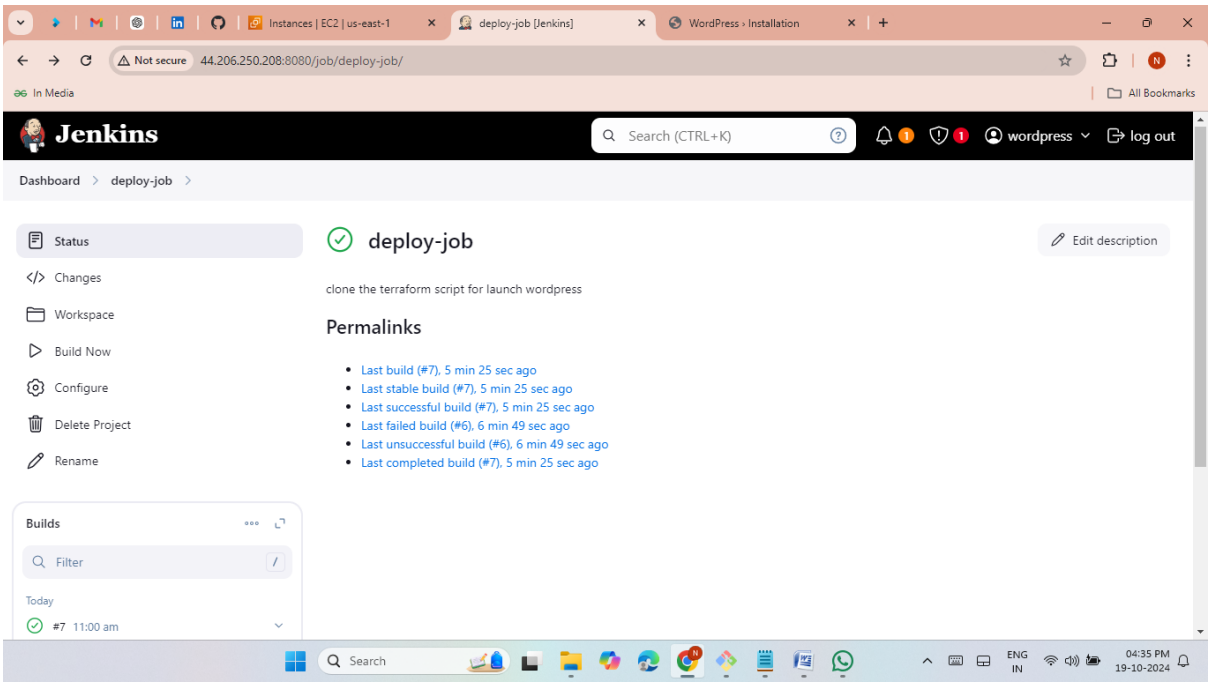
Sudo visudo

- Add the Following Line ->At the end of the file

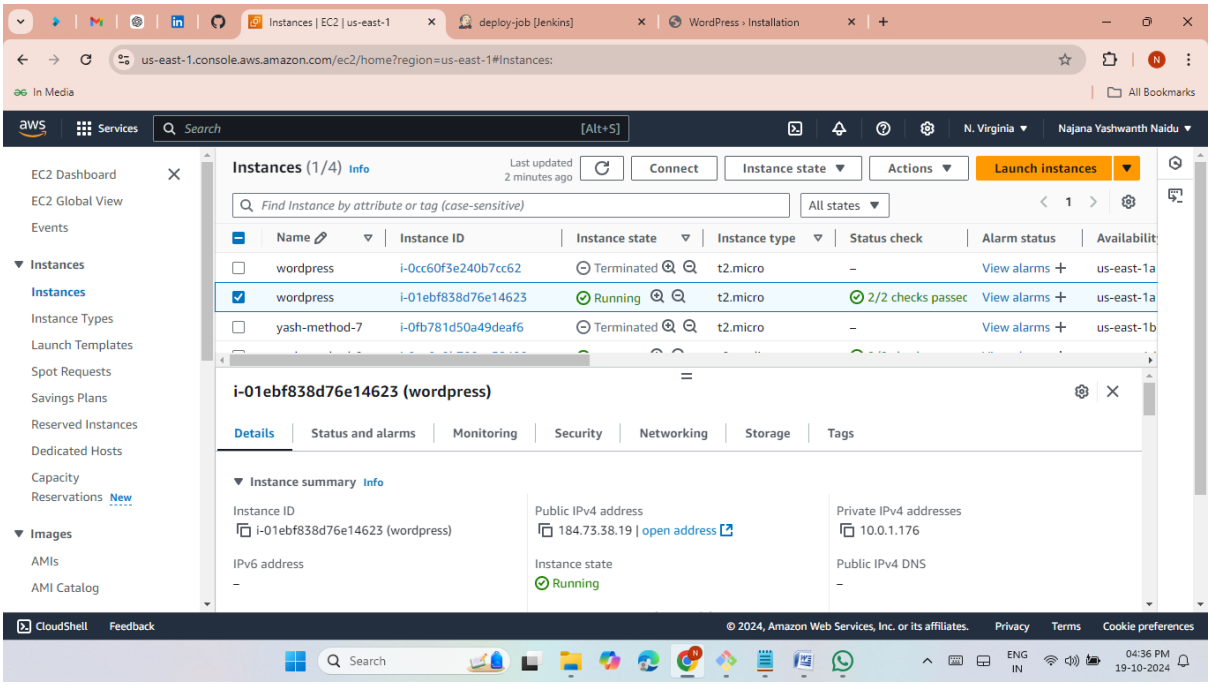
jenkins ALL=(ALL) NOPASSWD: ALL

- Now restart the Jenkins with the command **(sudo systemctl restart Jenkins).**

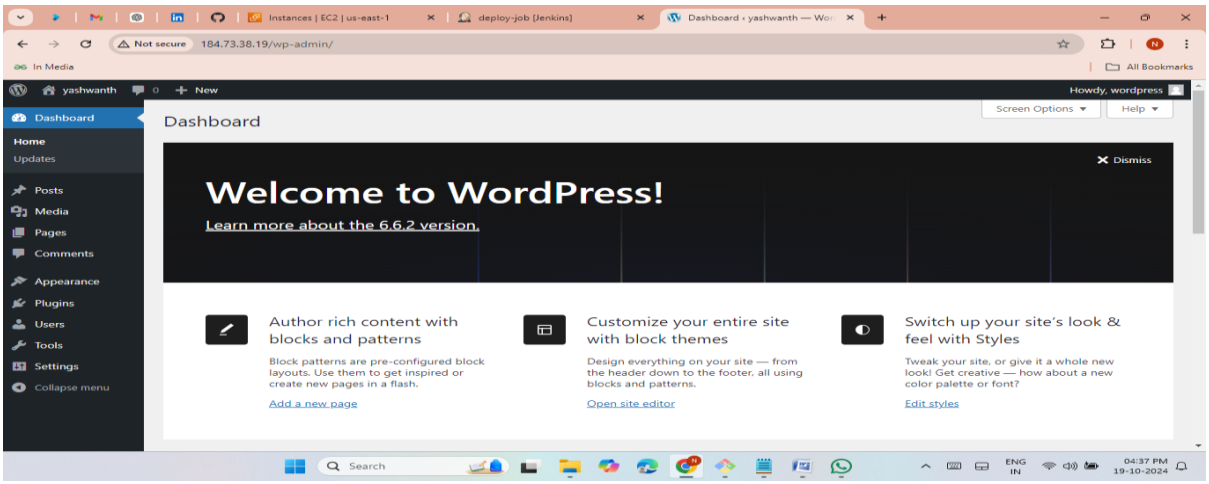
- Now click on build the job was success.



- Here the EC2 instances was automatically launched with help of terraform script.
- Also create the VPC, Subnet, Internet Gateway, Route Table, Security Groups with help of terraform script.



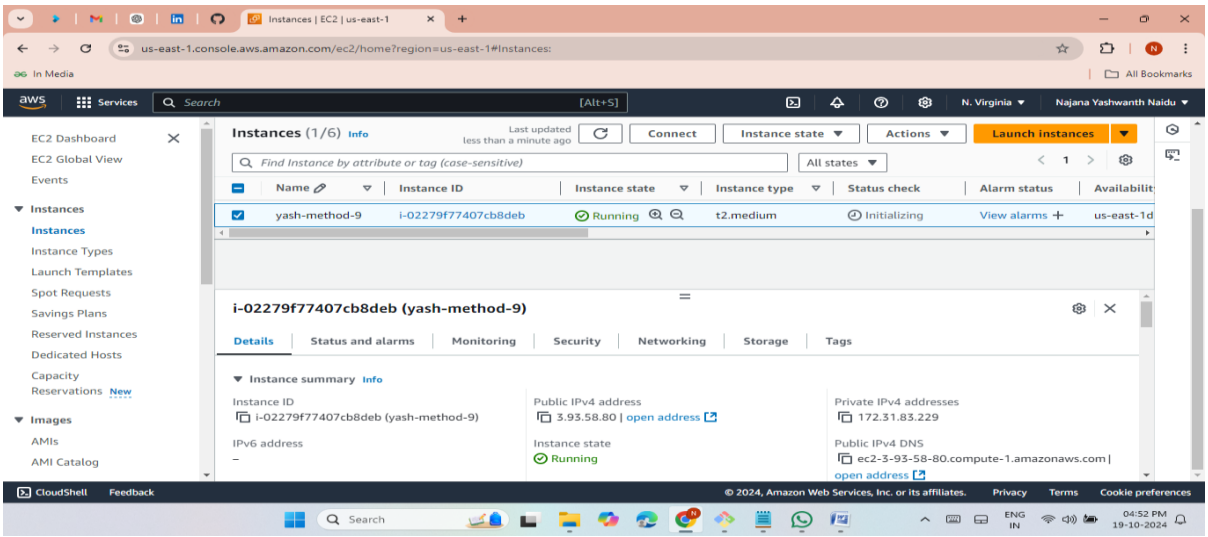
- Here the wordpress is launched successfully.



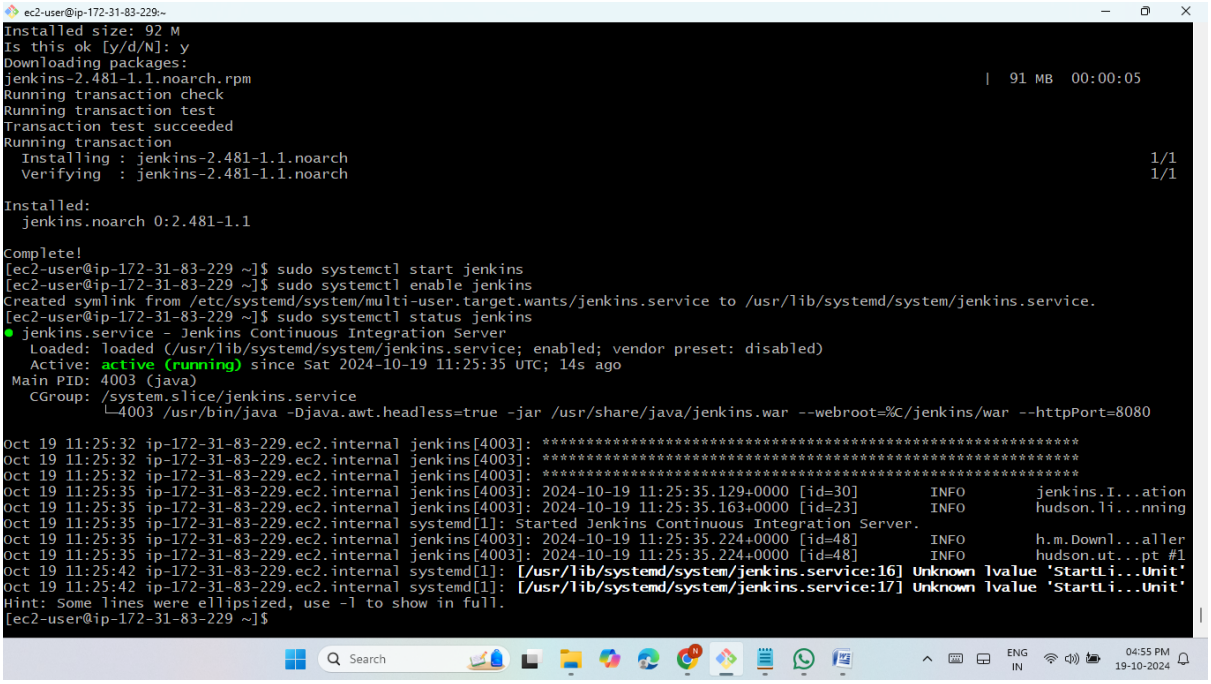
METHOD-9

Deploy WordPress web application by using git (clone terraform script which helps to deploy WordPress web application), jenkins (in execute shell install terraform, init, fmt, validate and apply with automatic command as terraform apply --auto-approve) and terraform and create jenkins pipeline and add build periodically and poll scm to initial job of pipeline and check the changes happened or not which are made in github repo.

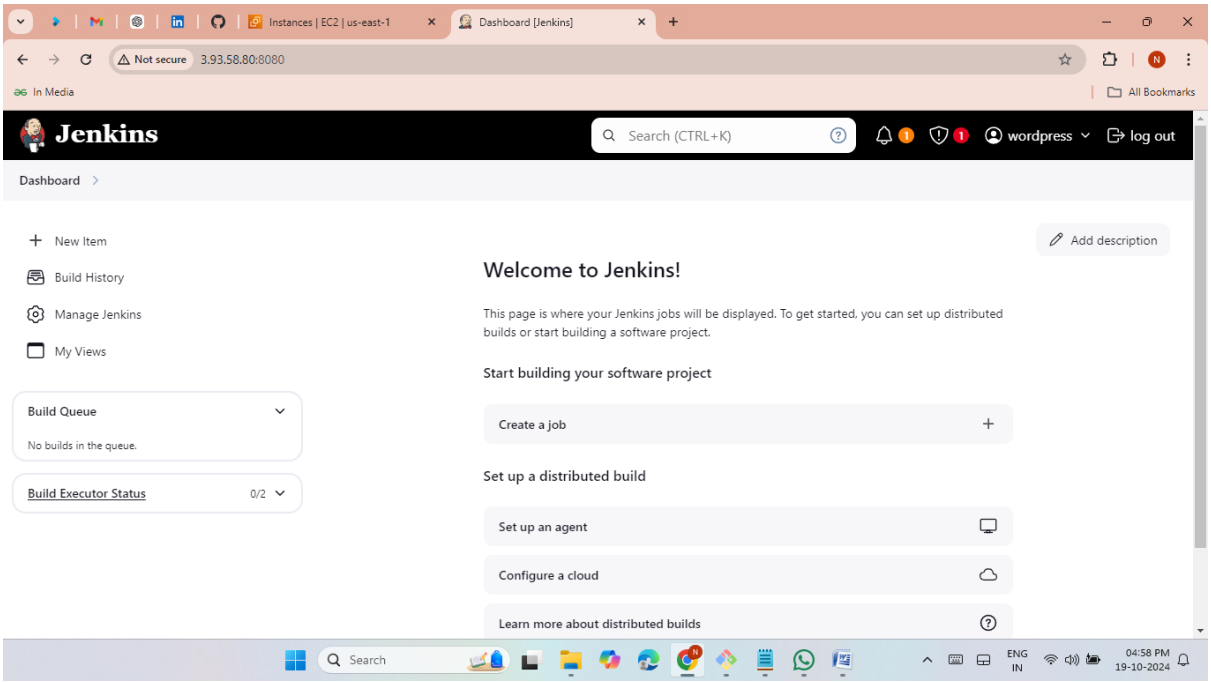
- First launch the EC2 instance with the Jenkins port number(8080).



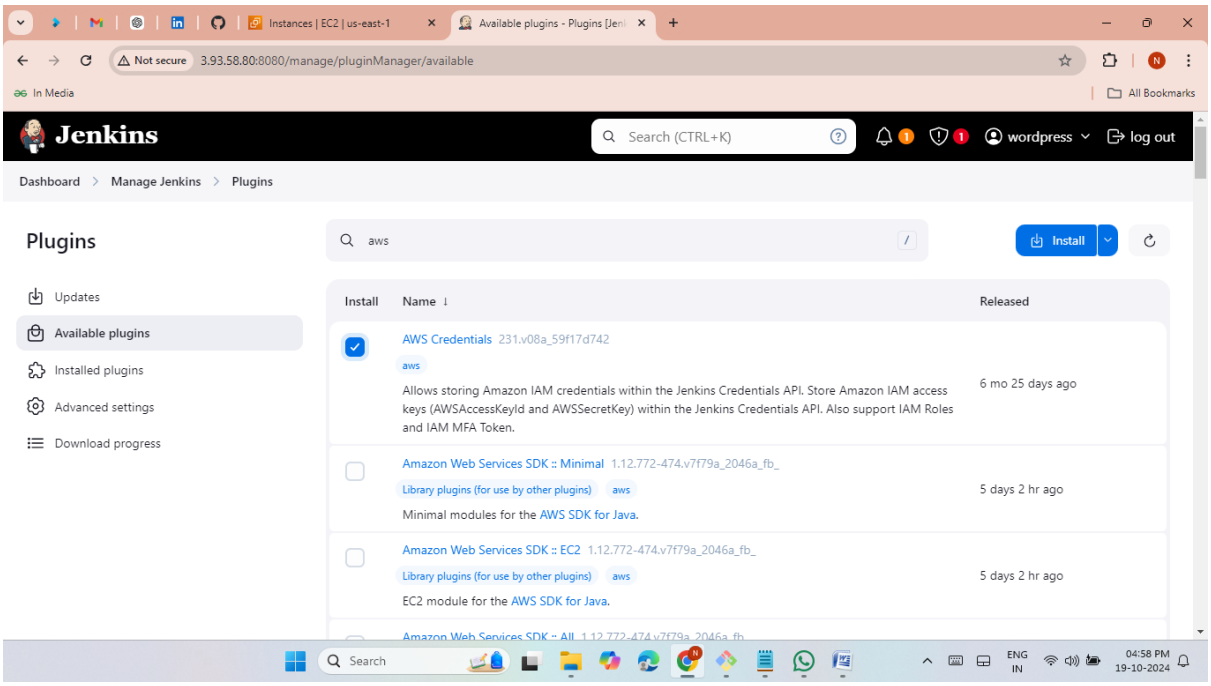
- Now Install git and Jenkins.
- Now start and enable the Jenkins.



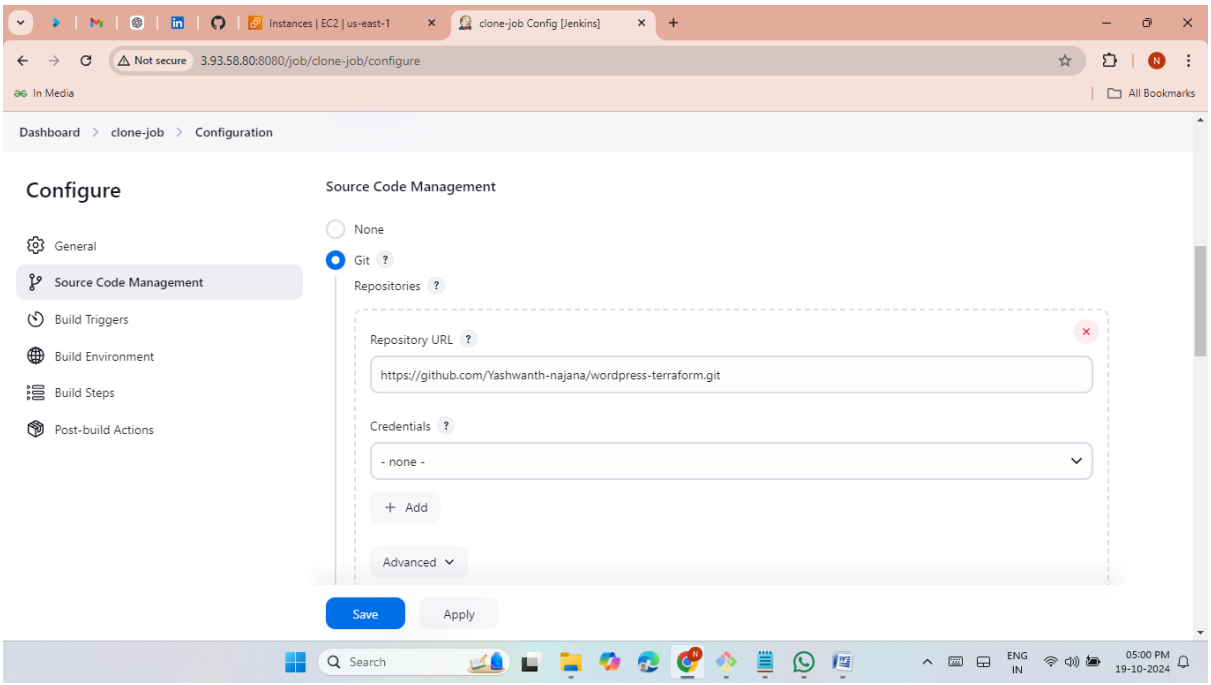
- Here the Jenkins was launched successfully.



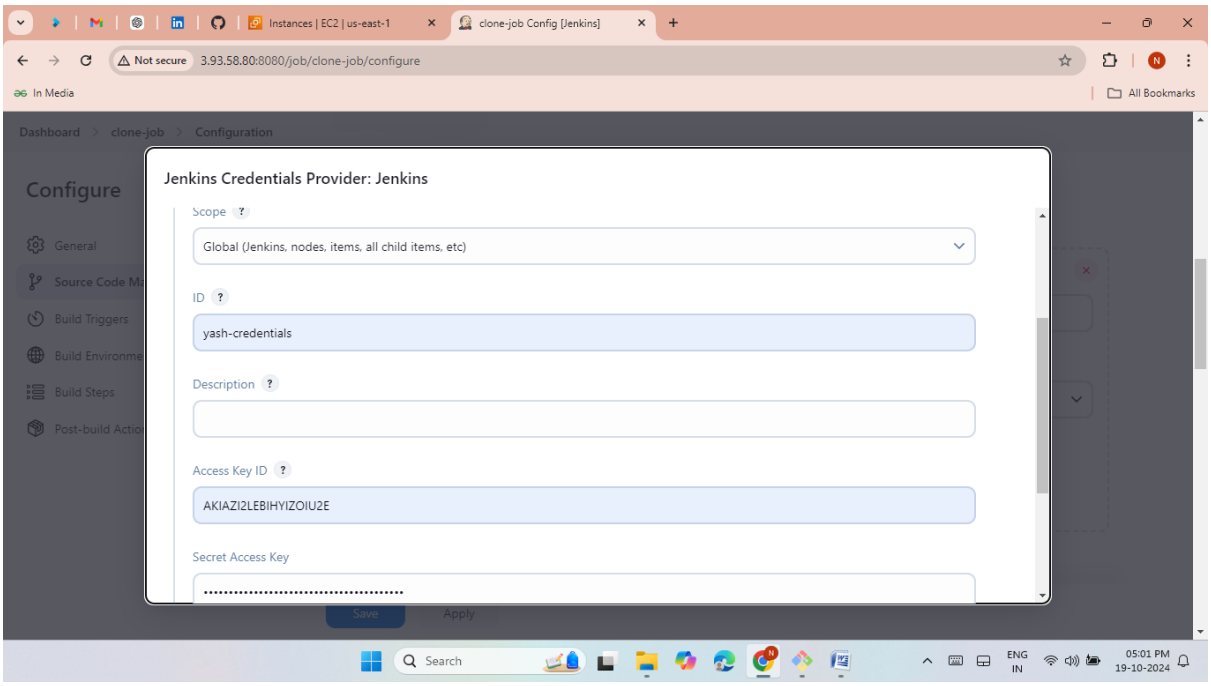
- Now install the AWS credentials plugin.



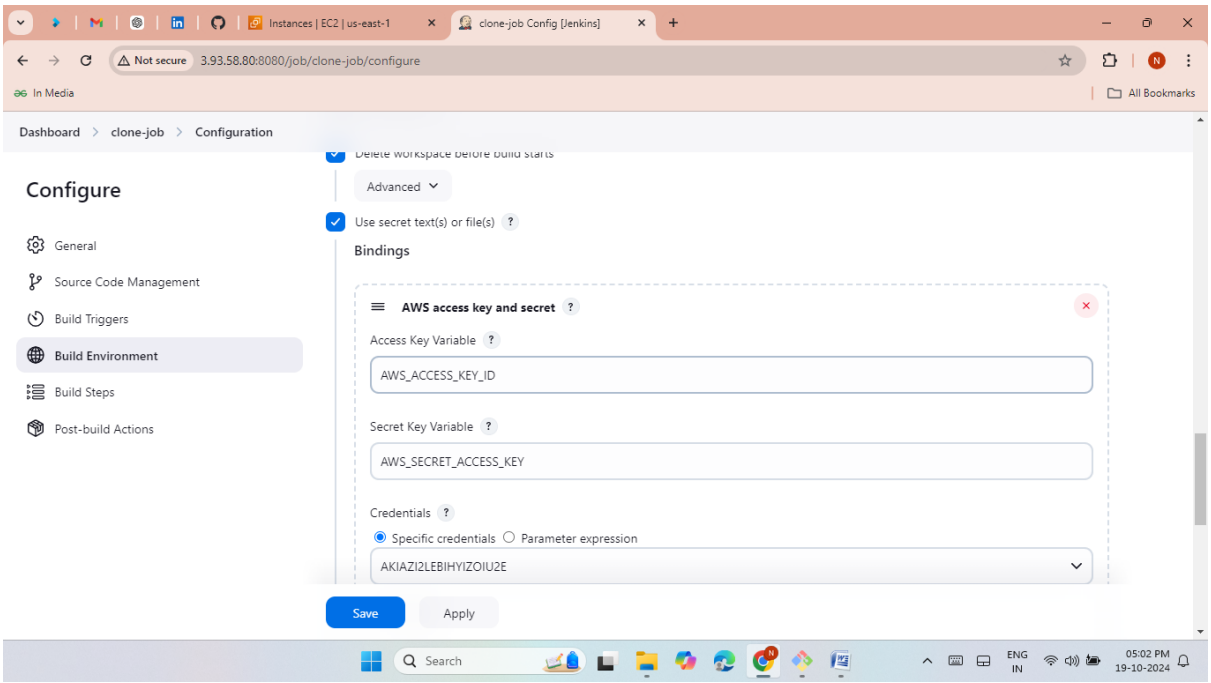
- Now create a clone-job and clone the terraform script form github.



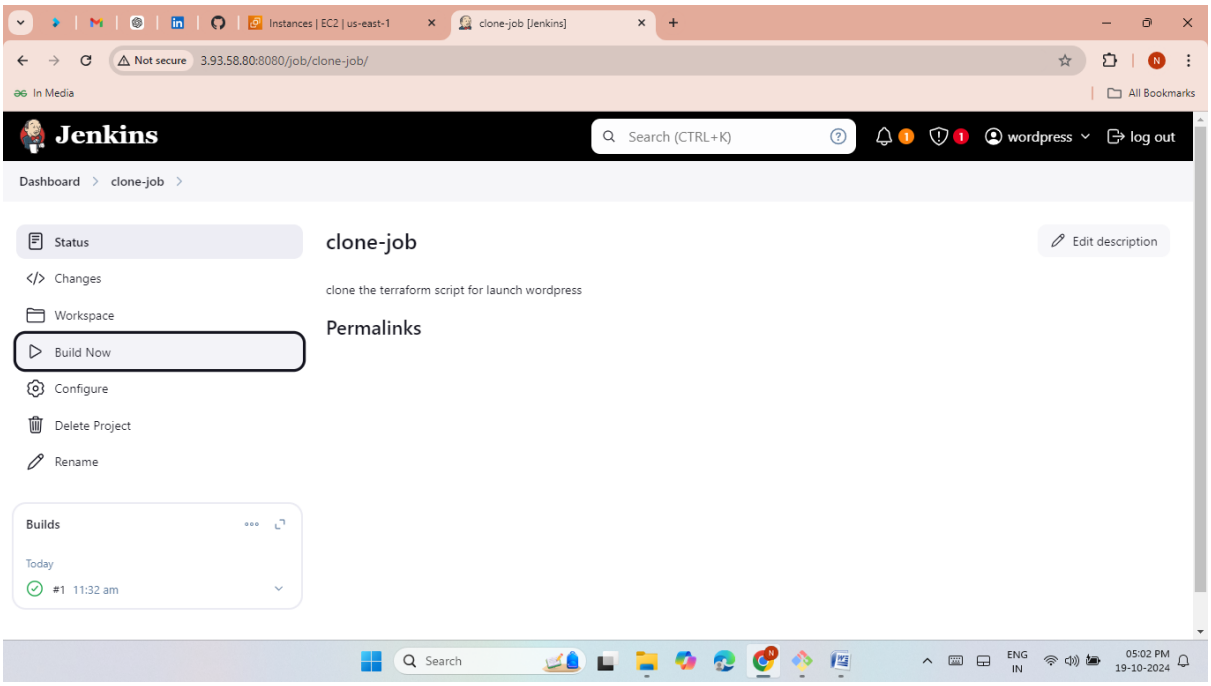
- Then create the AWS credentials.



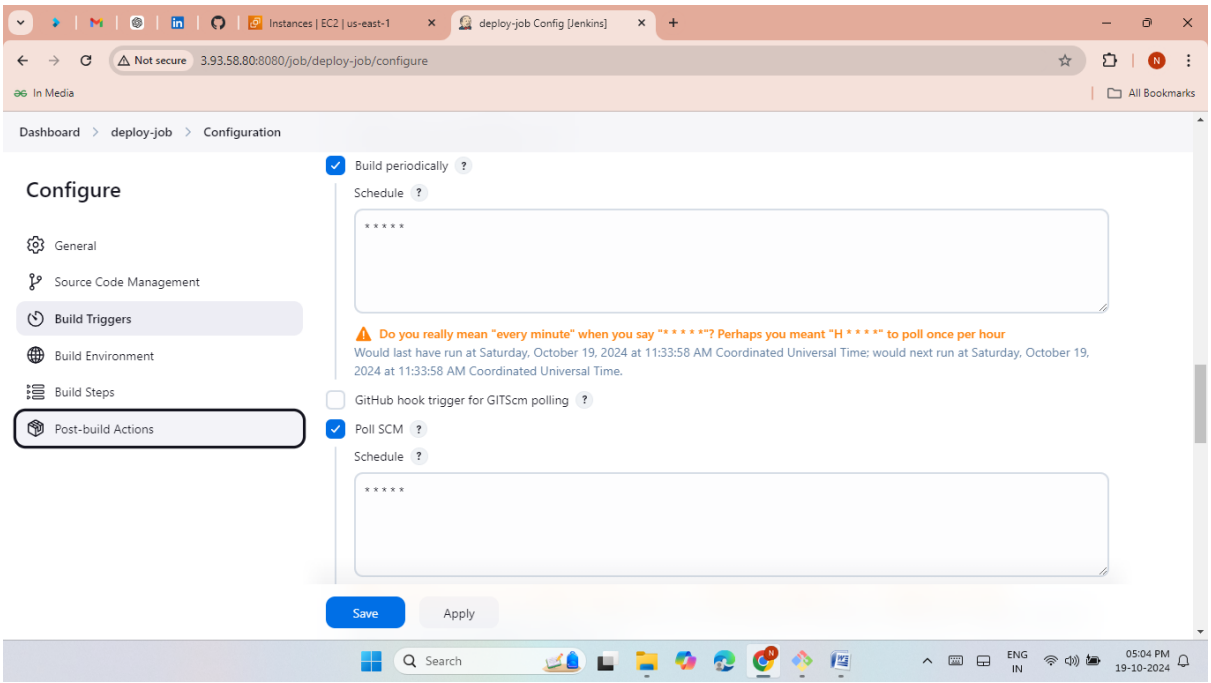
- Now connect the access keys and secret keys.
- Now apply the changes.



- Here the clone job is success.



- Now create new job and connect to the clone-job.
- Now Add the Build Periodically and Poll SCM.



- Now install the terraform in execute shell.
- Now add the terraform commands.

#!/bin/bash

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

sudo chmod + data.sh

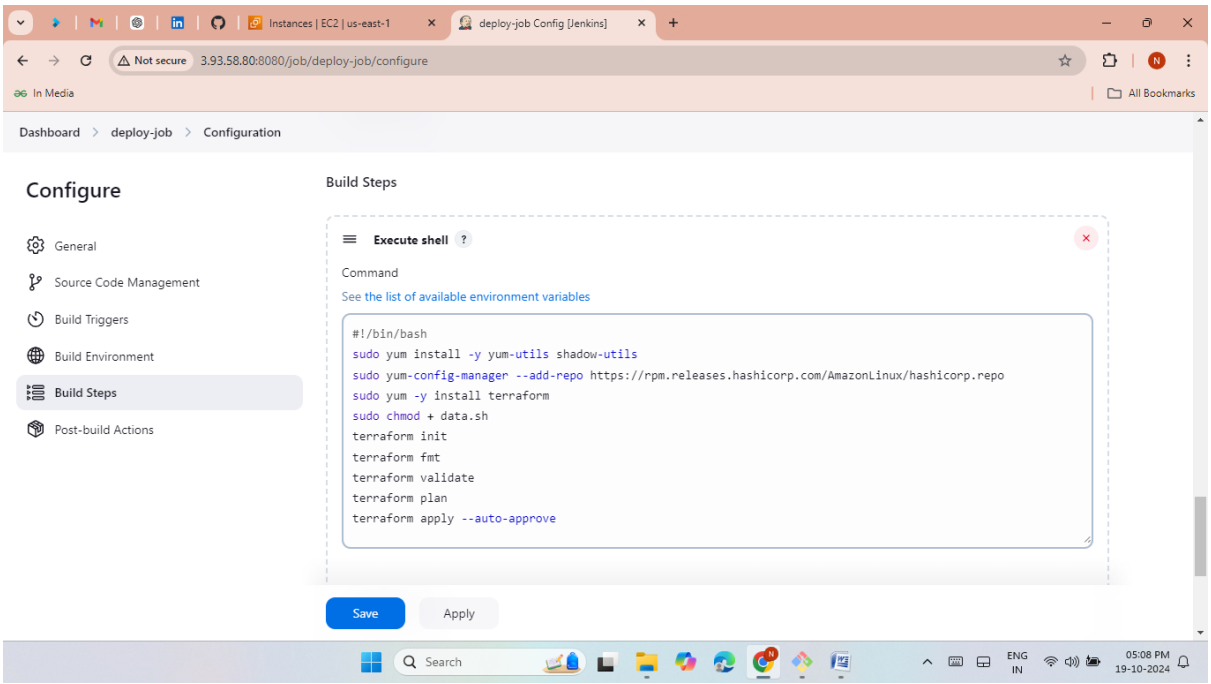
terraform init

terraform fmt

terraform validate

terraform plan

terraform apply --auto-approve

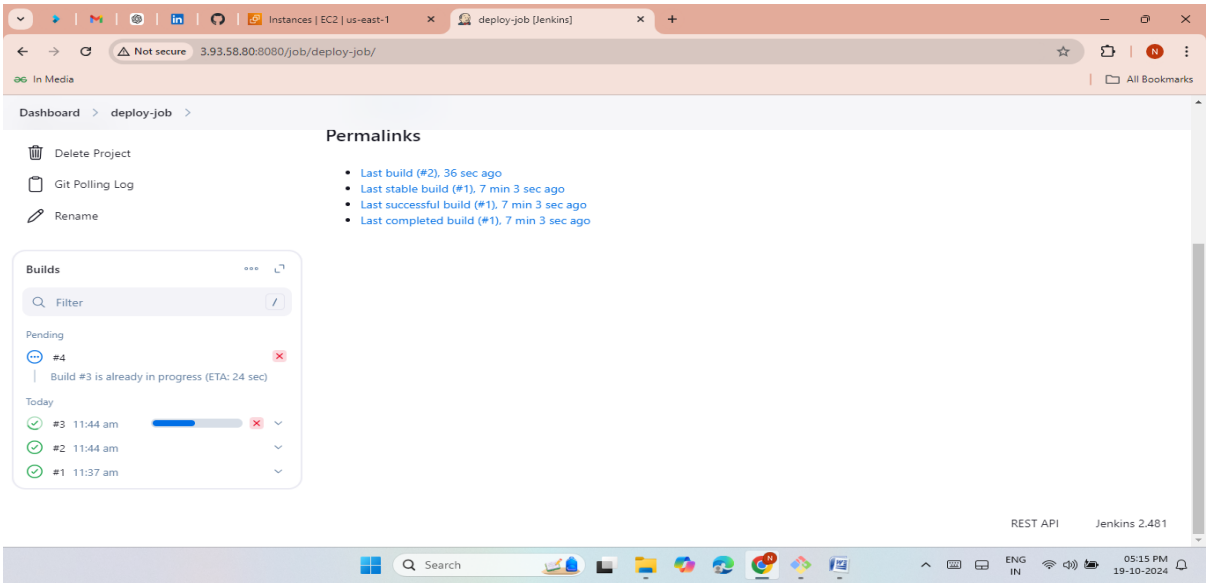


- Give permissions to jenkins.

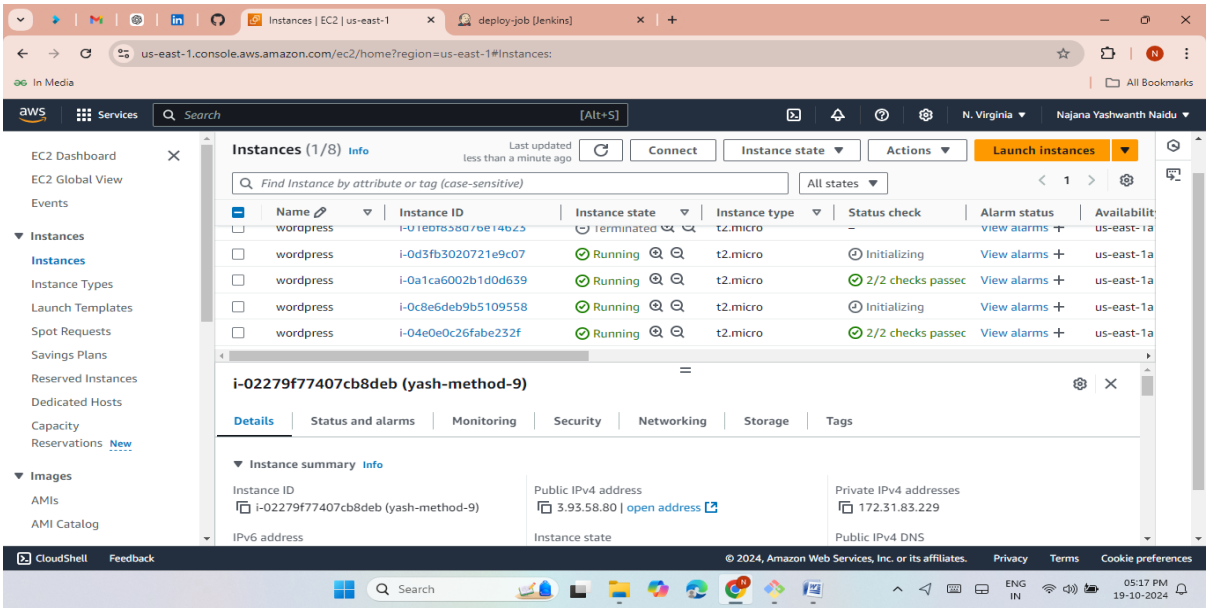
  Sudo visudo

  jenkins ALL=(ALL) NOPASSWD: ALL

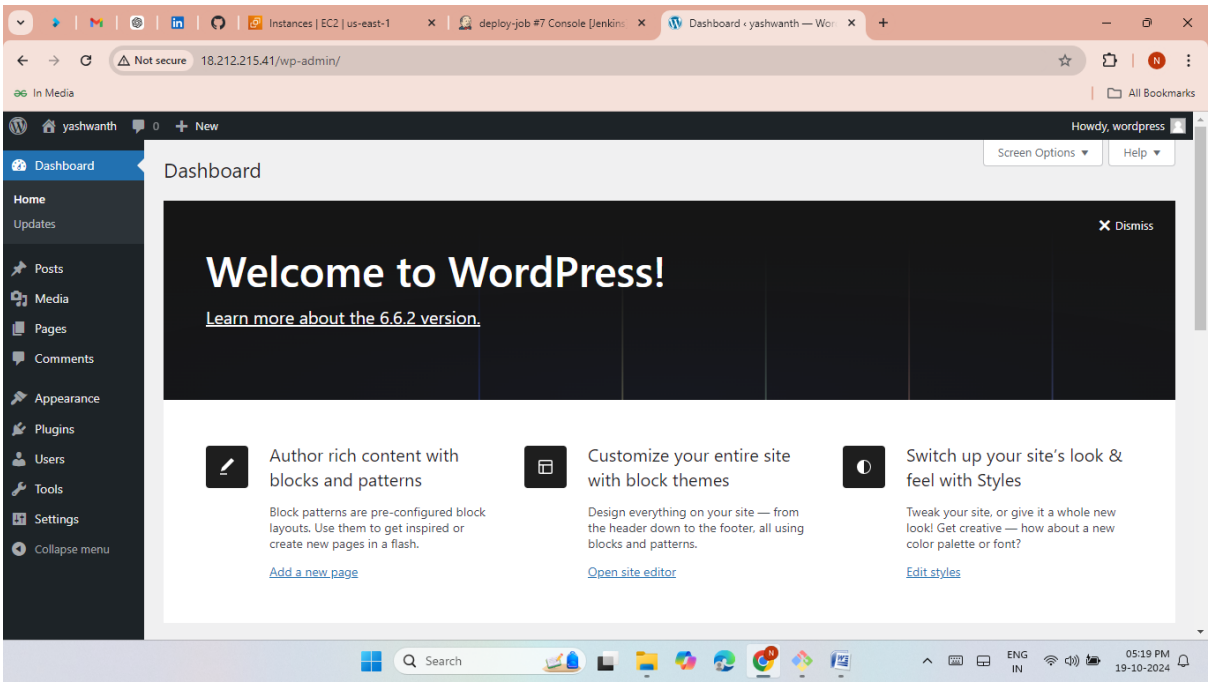
- Now restart the Jenkins with the command **(sudo systemctl restart Jenkins).**
- Now click on build the job was success.
- It runs every MINUTE HOUR DOM(DAY OF MONTH) MONTH DOW(DAY OF WEEK).
- Because I have given (* * * * *).

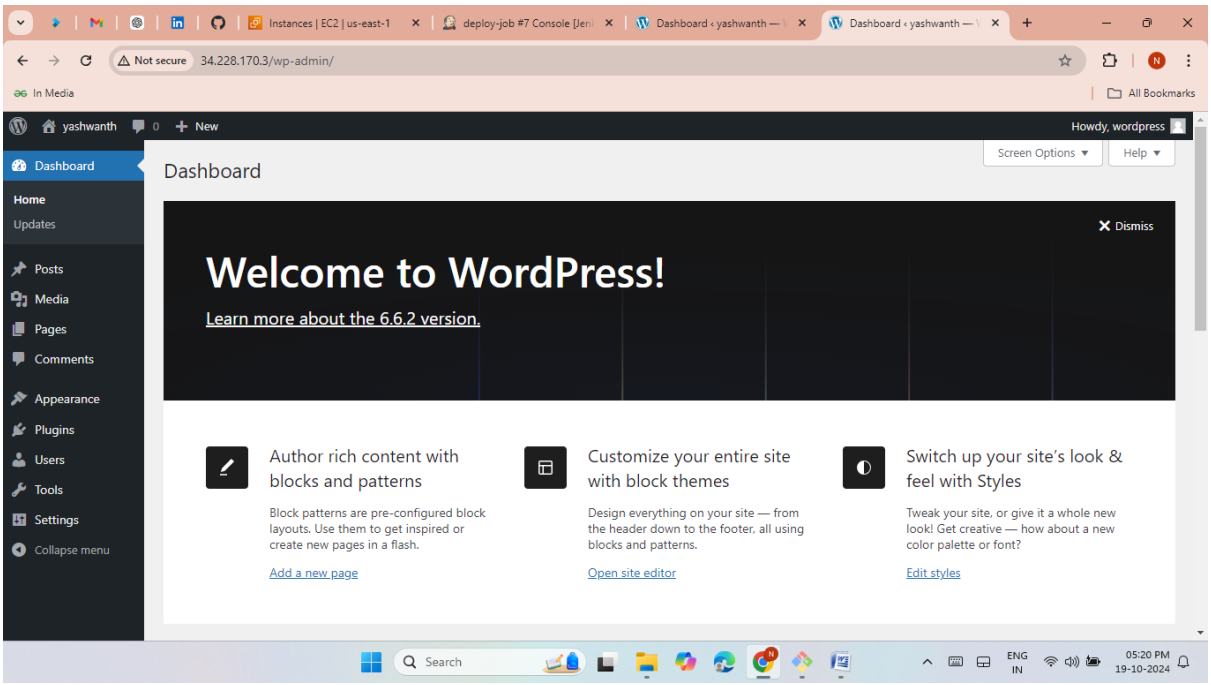


- Here the instances was automatically launched every one min.
- Also create the VPC, Subnet, Internet Gateway, Route Table, Security Groups with help of terraform script.
- Because of Build Periodically and Poll SCM.

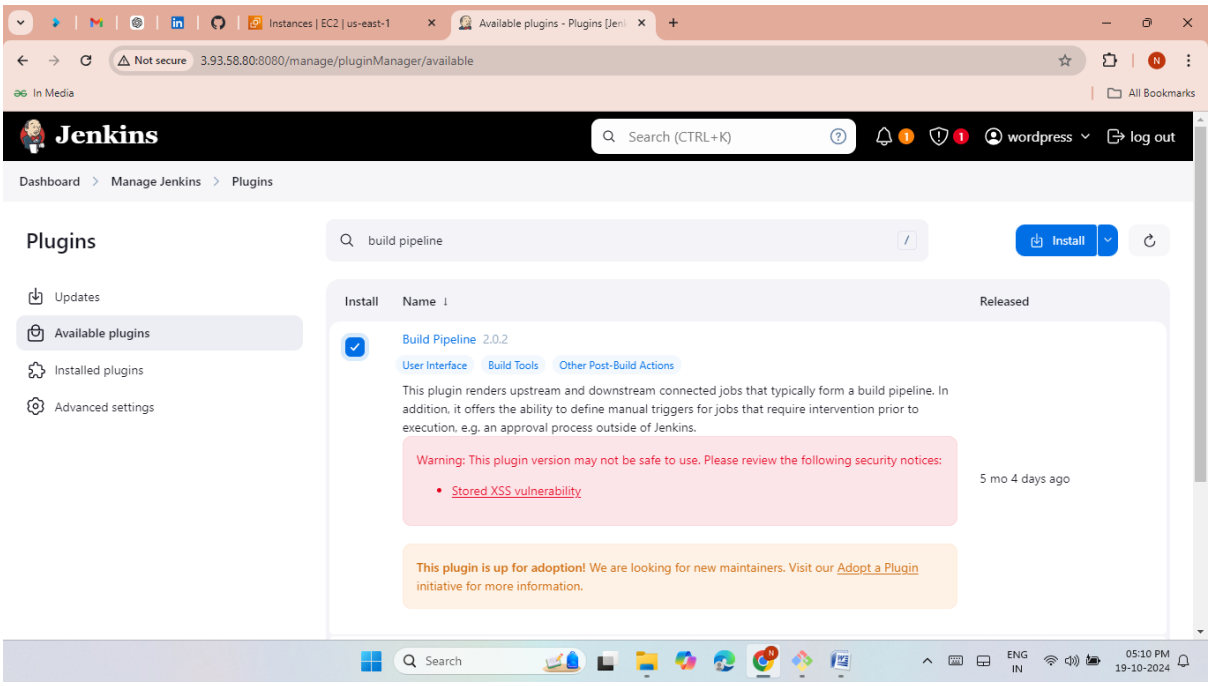


- Here the all wordpress applications was launced.

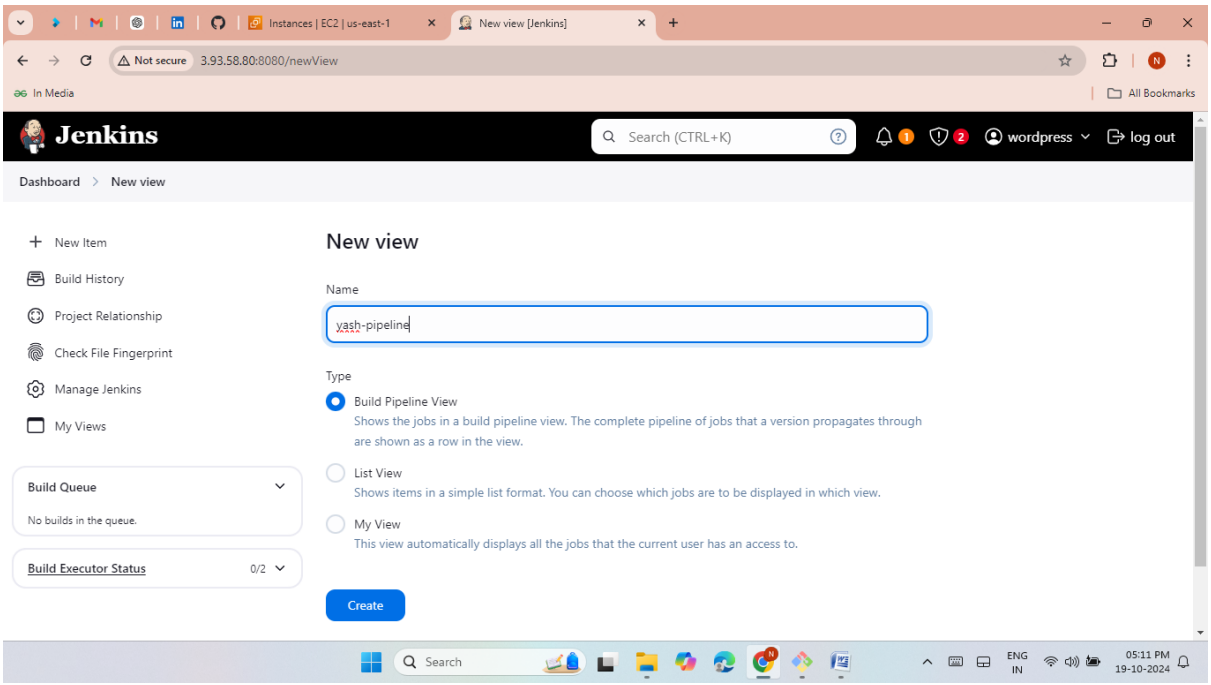




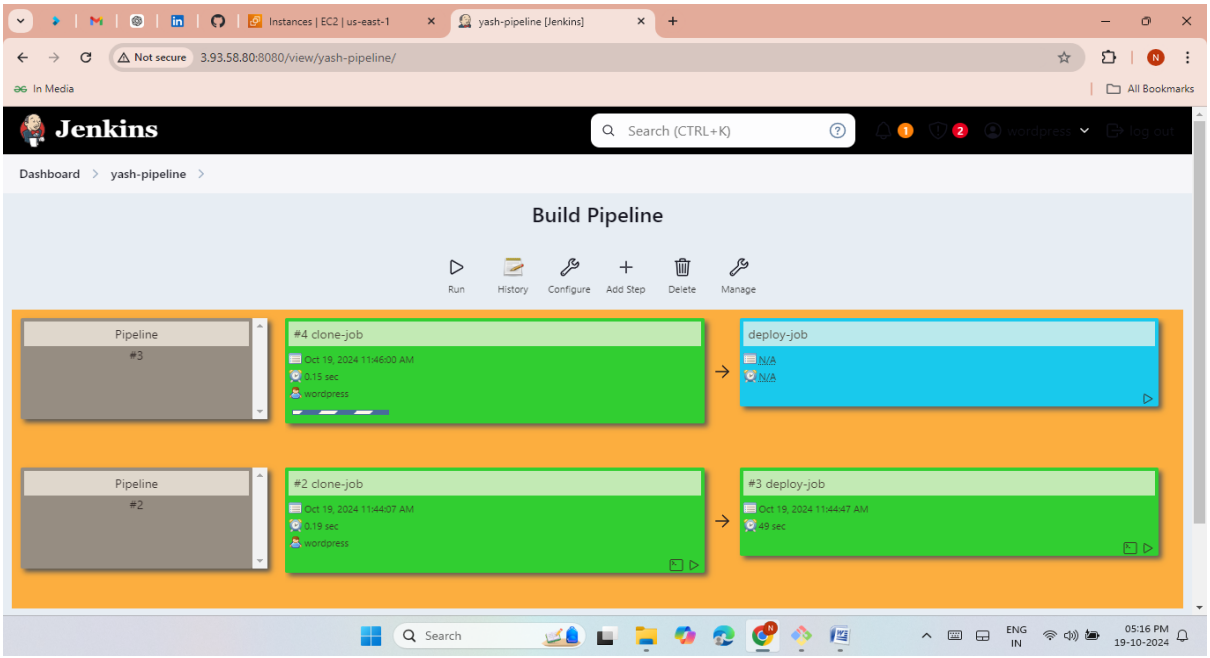
➢ Now install the build pipeline plugin for create a Jenkins pipeline.



➢ Now create a Jenkins pipeline.



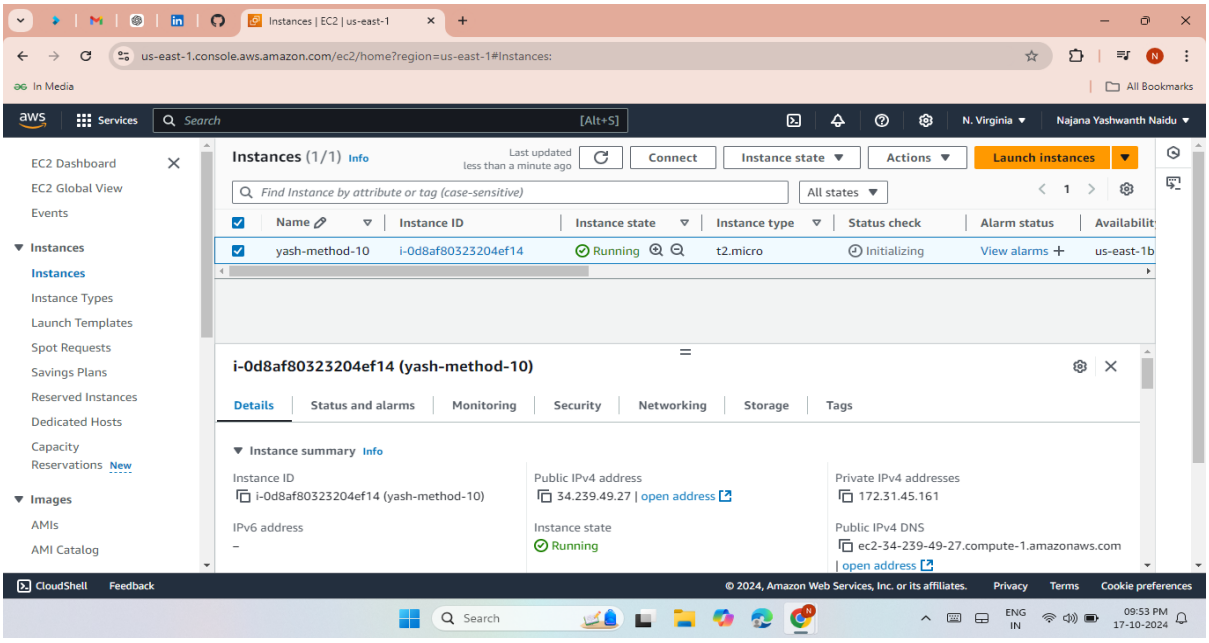
- Here the build pipeline is successfully created.



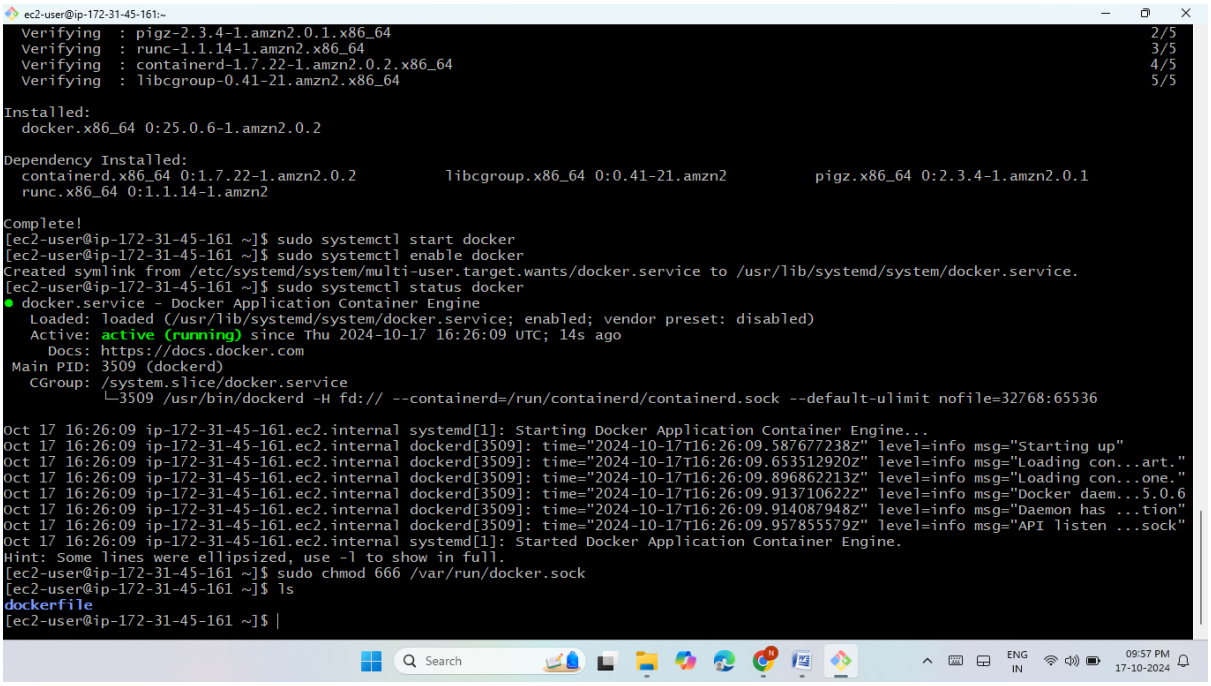
METHOD-10

Deploy WordPress web application by using k8's (Declarative manifest method) with the help of docker hub images.

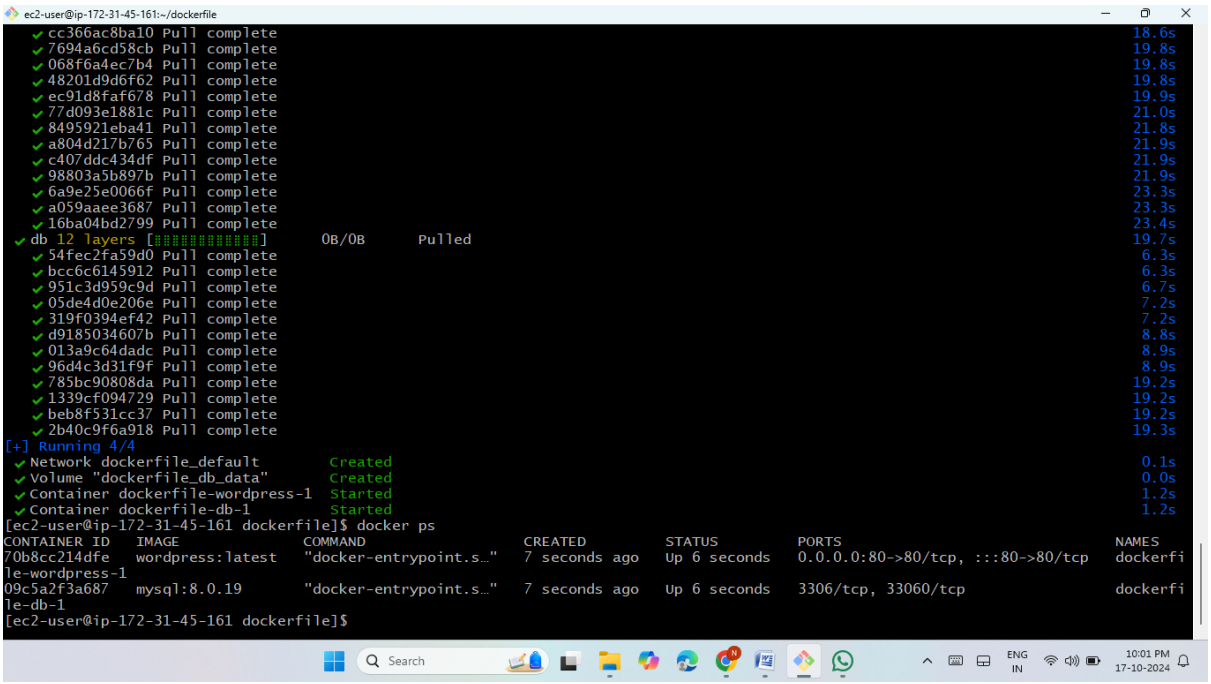
- First launch an EC2 instance for push the docker image into dockerhub.



- Now install docker.
- Now start and enable the docker.
- Give permissions to the docker with the command(**sudo chmod 666 var/run/docker.sock**).
- Now install git and clone the docker-compose file.



- Now install docker-compose and give permissions.
- Now run the docker-compose file with the command (**docker-compose up –d**).
- Now we can see the docker images.

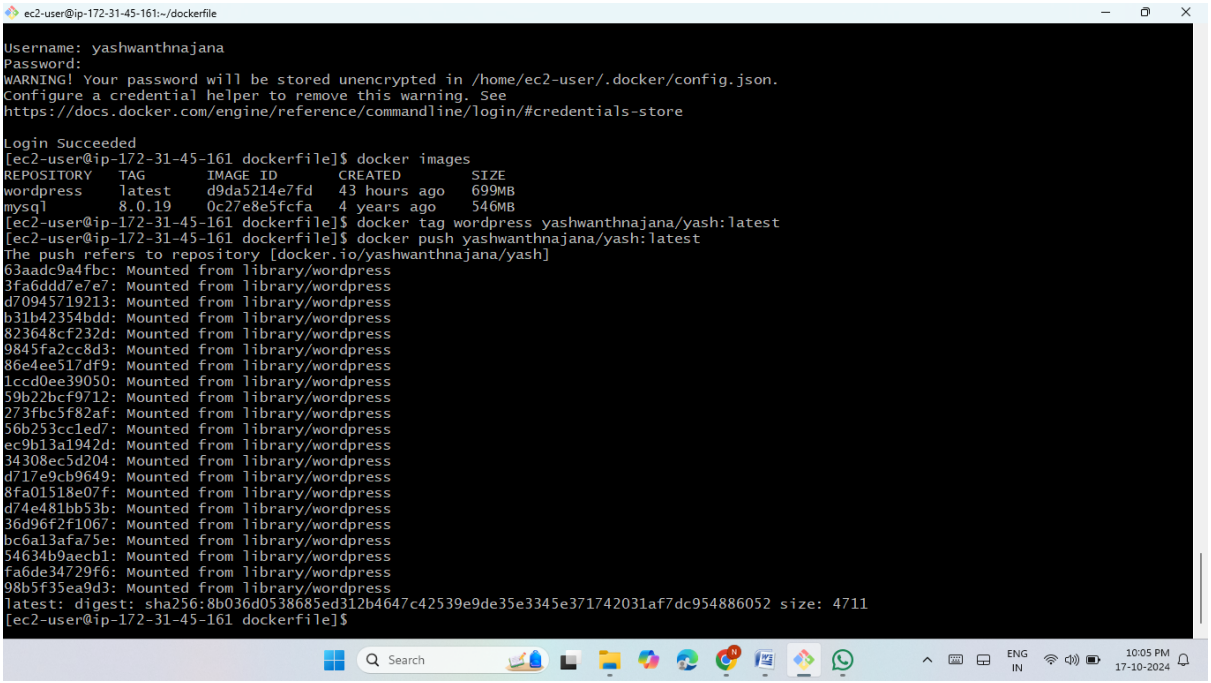


➢ Now run the following commands to push the docker images into DockerHub platform.

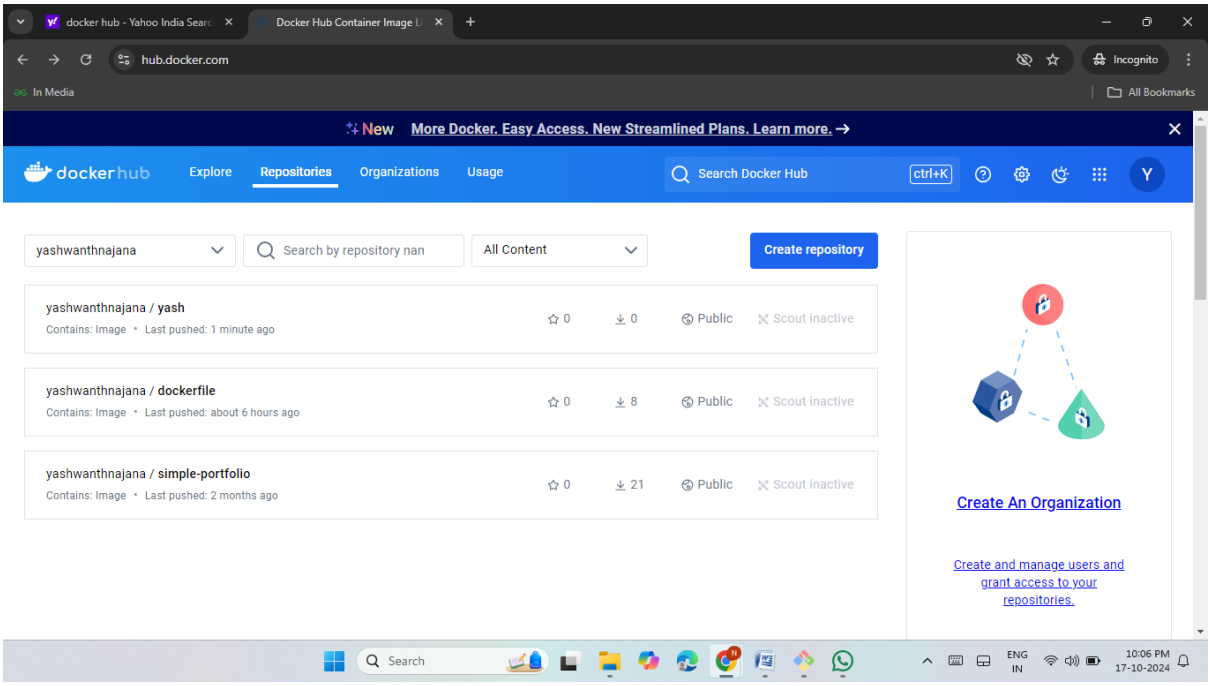
docker login (give the DockerHub user and password).

docker tag <image name> <username>/<repo name>:<tag>

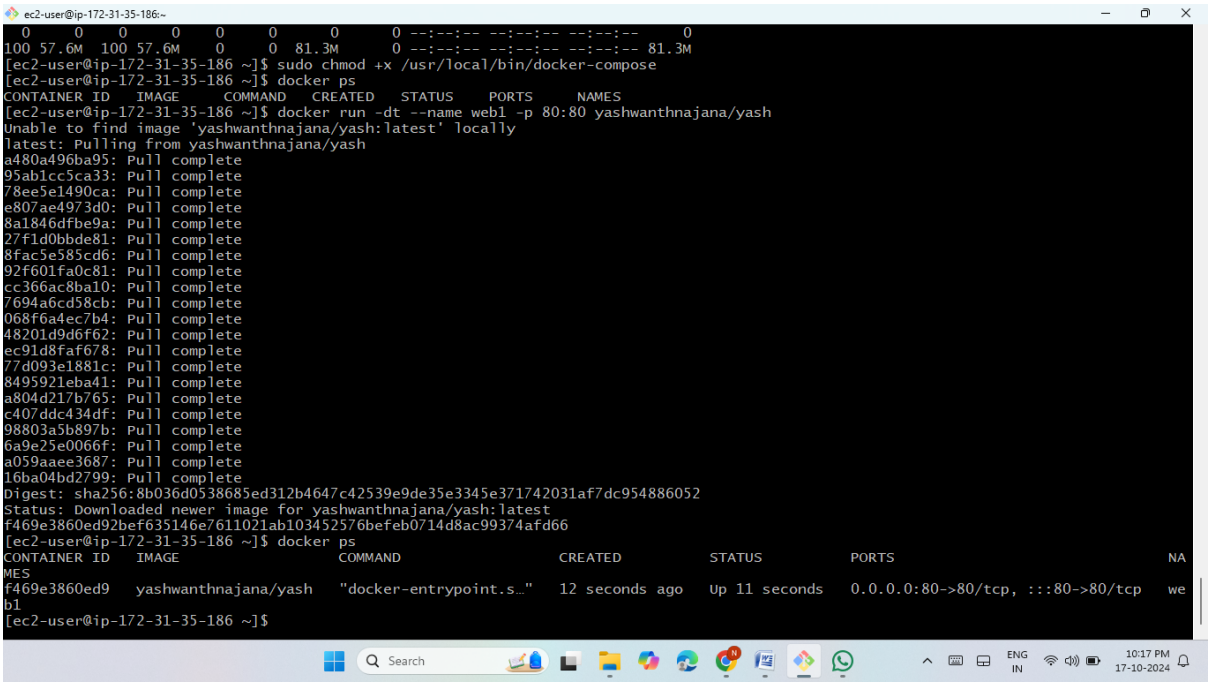
docker push <username>/<repo name>:<tag>



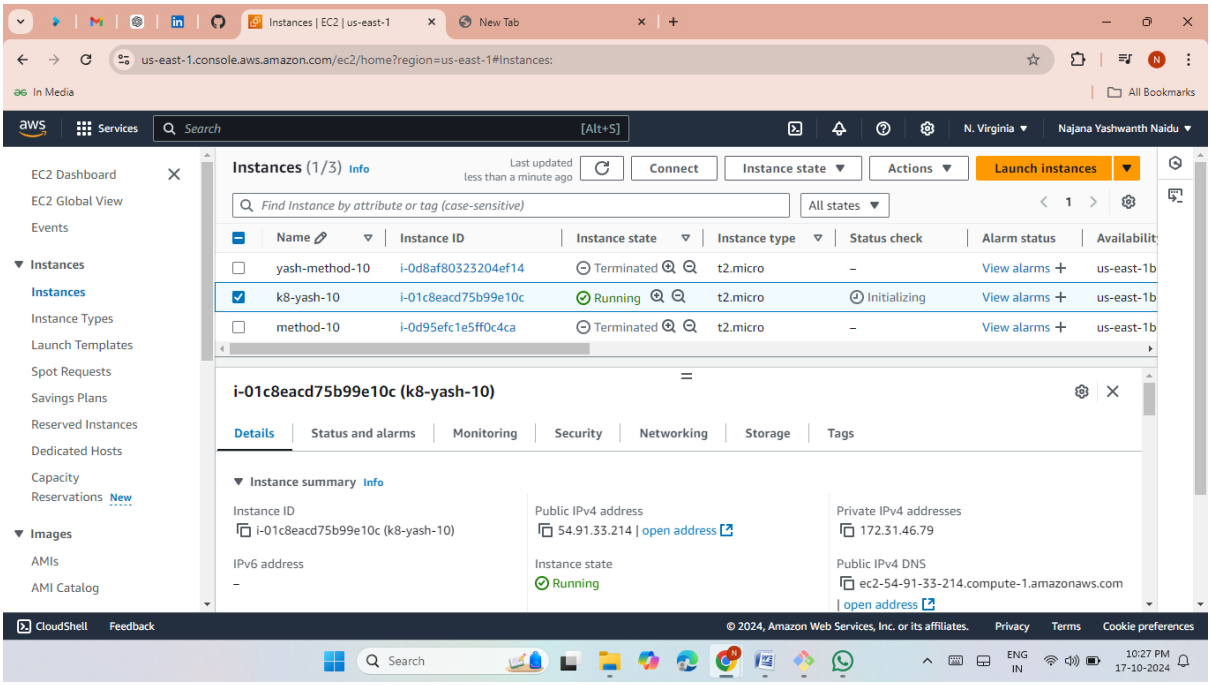
➢ Here the docker image is pushed successfully (yash).



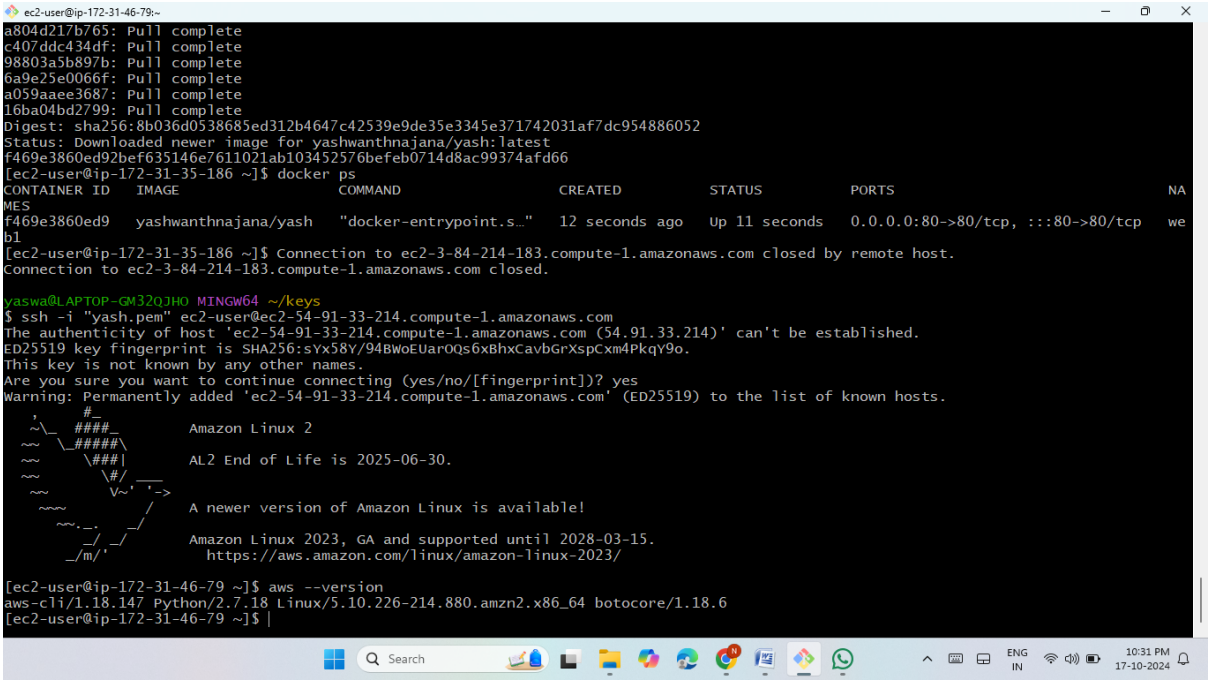
- Launch another instances and pull the docker image and run wordpress application (it is for checking purpose).
- Succesufully hosted the wordpress.



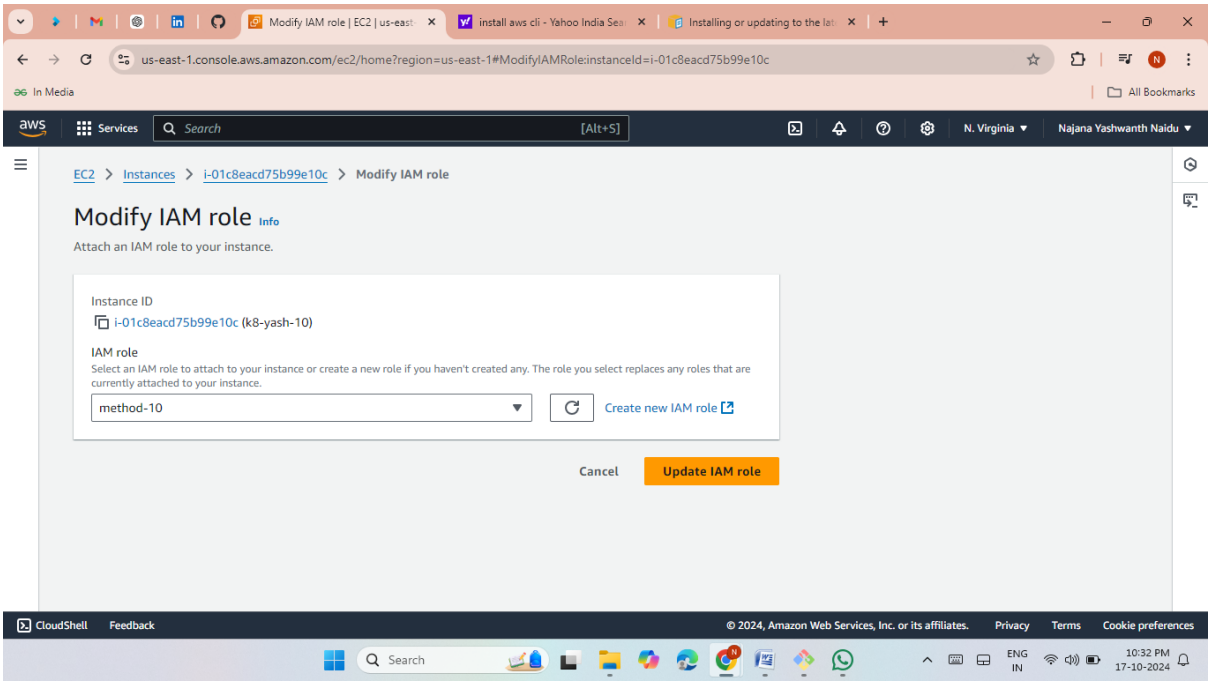
- Now launch an EC2 instance for hosting the wordpress in kubernetes.



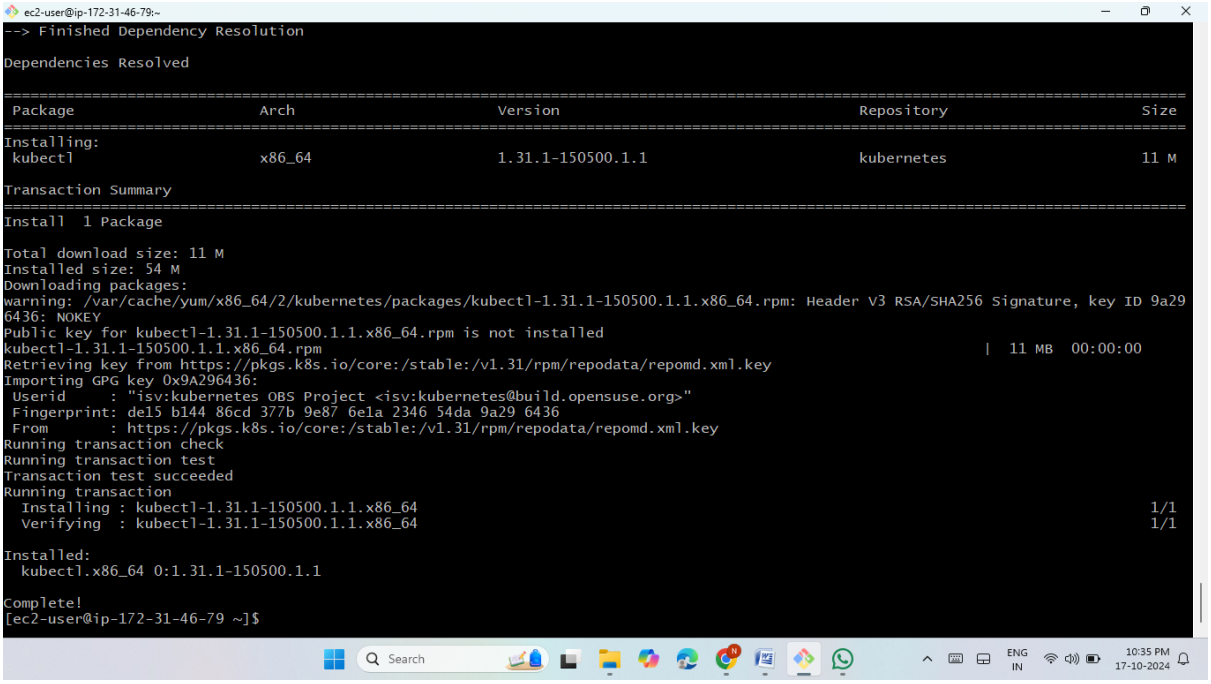
- Now install aws cli (in linux by default aws cli is there. If we want updated version, we can download).



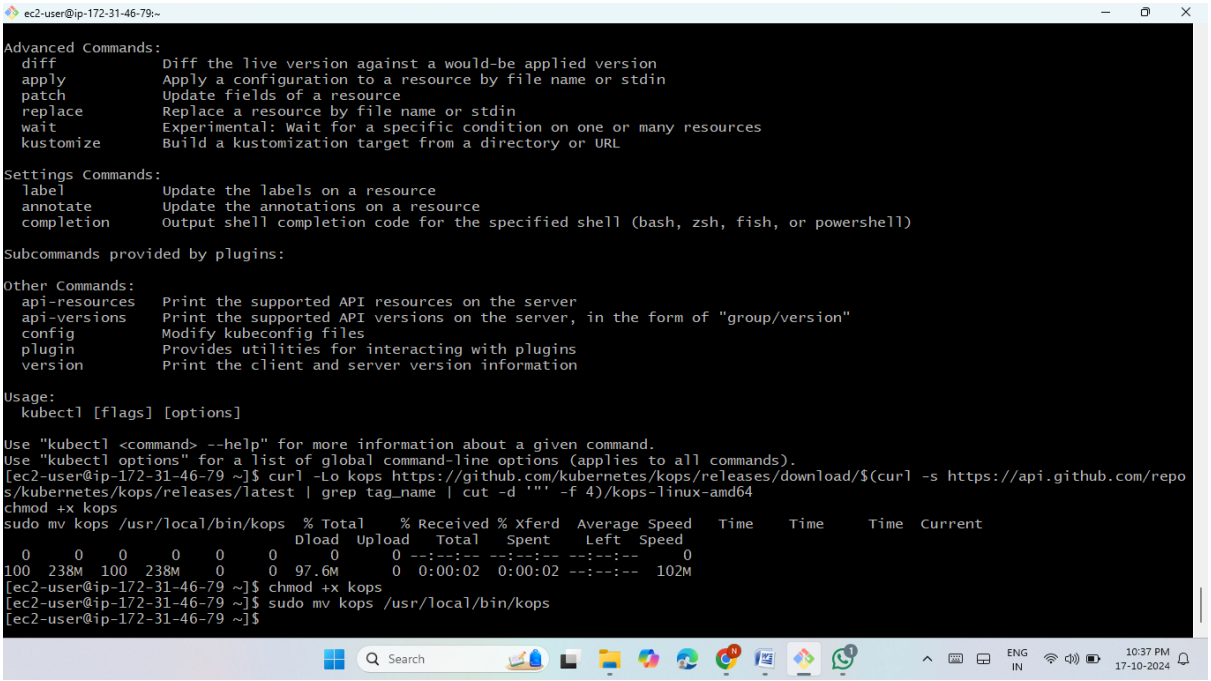
- Create role with all admin permissions and attach it to the EC2 instance.



- Now first Install kubectl package from Google.
- Then install kubectl.



- Now install kops from Google.



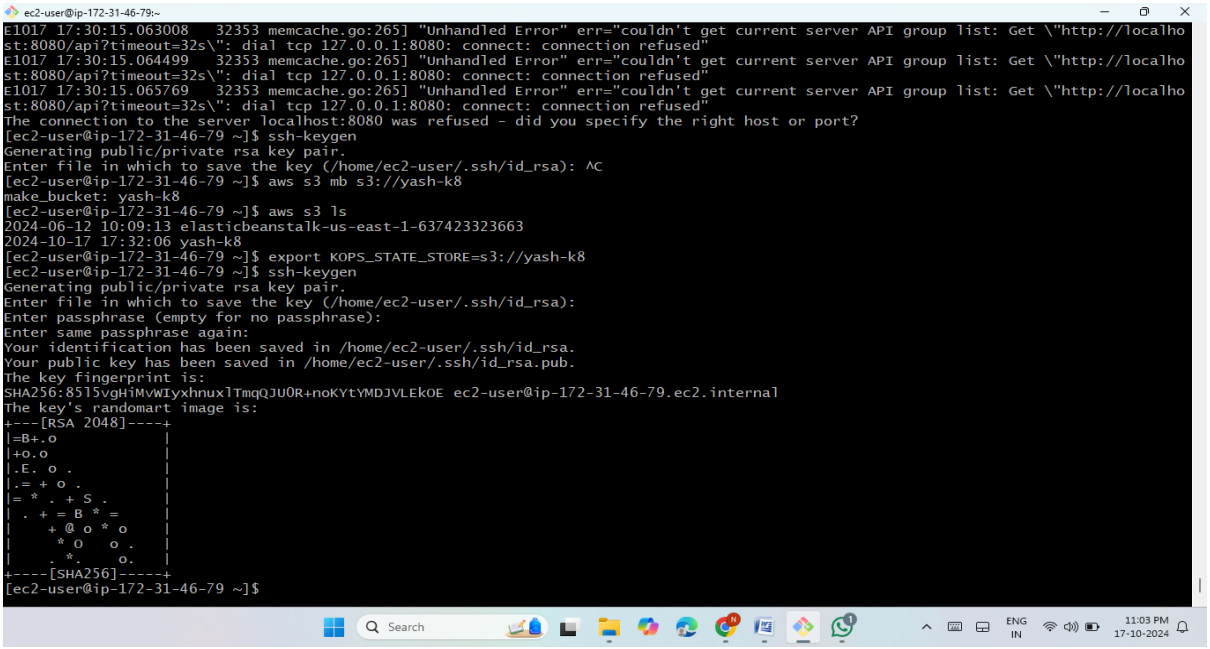
- Now run the following commands.

aws s3 mb s3://yash-k8 (to create s3 bucket to store the data).

aws s3 ls (to show the s3).

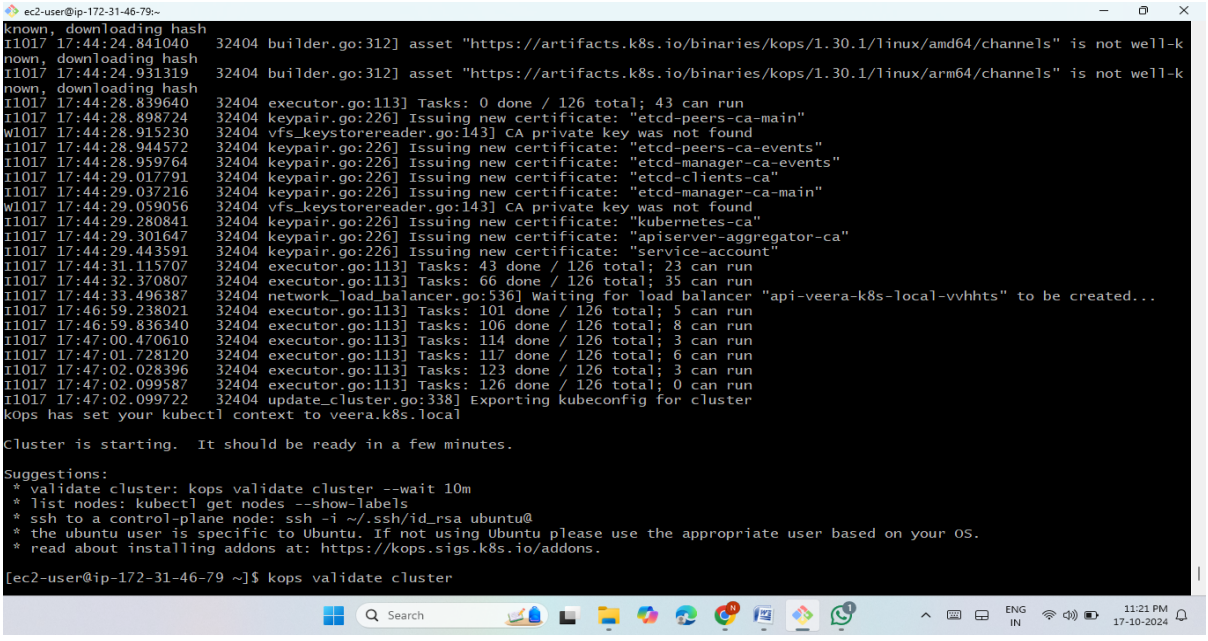
export KOPS_STATE_STORE=s3://yash-k8 (attach s3 bucket to kops for backup).

ssh-keygen (now enter).

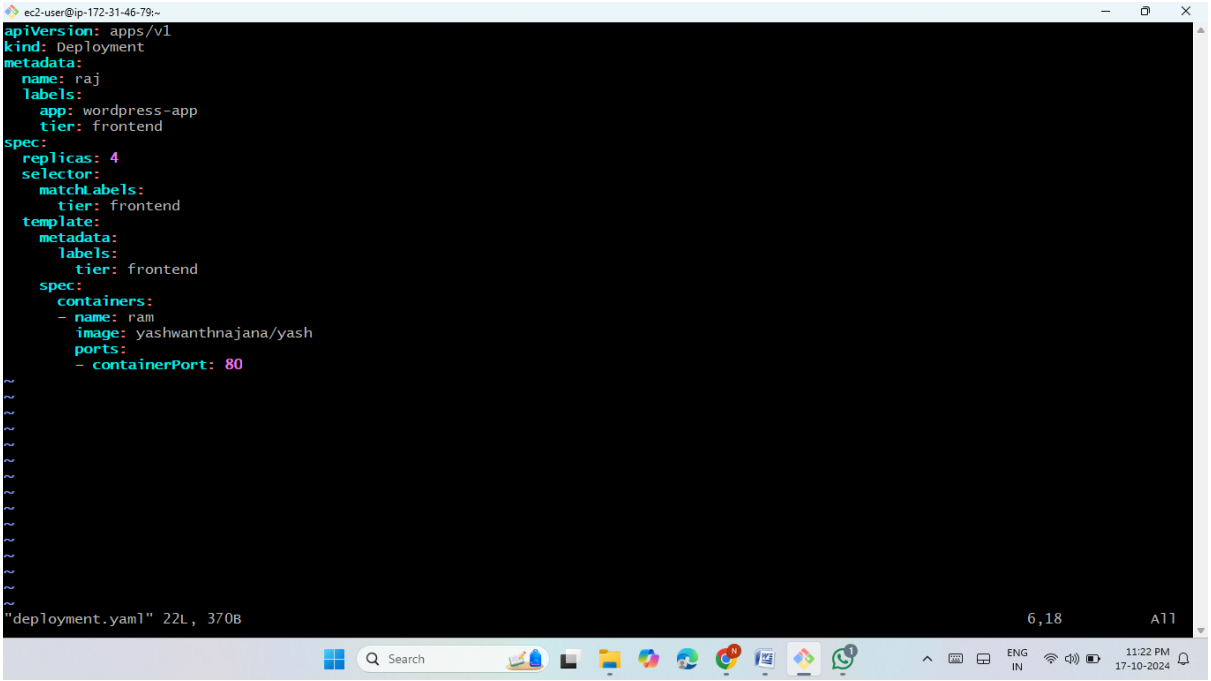


kops create cluster --name <cluster name>.k8s.local --state s3://<bucket name> --zones us-east-1b,us-east-1a --node-count 2 --yes (to create Cluster and it create VPC, Subnet, IG, Route, LoadBalancer, Target Group ,Auto Scalling Group, Instances).

kops validate cluster (to see the details od cluster).

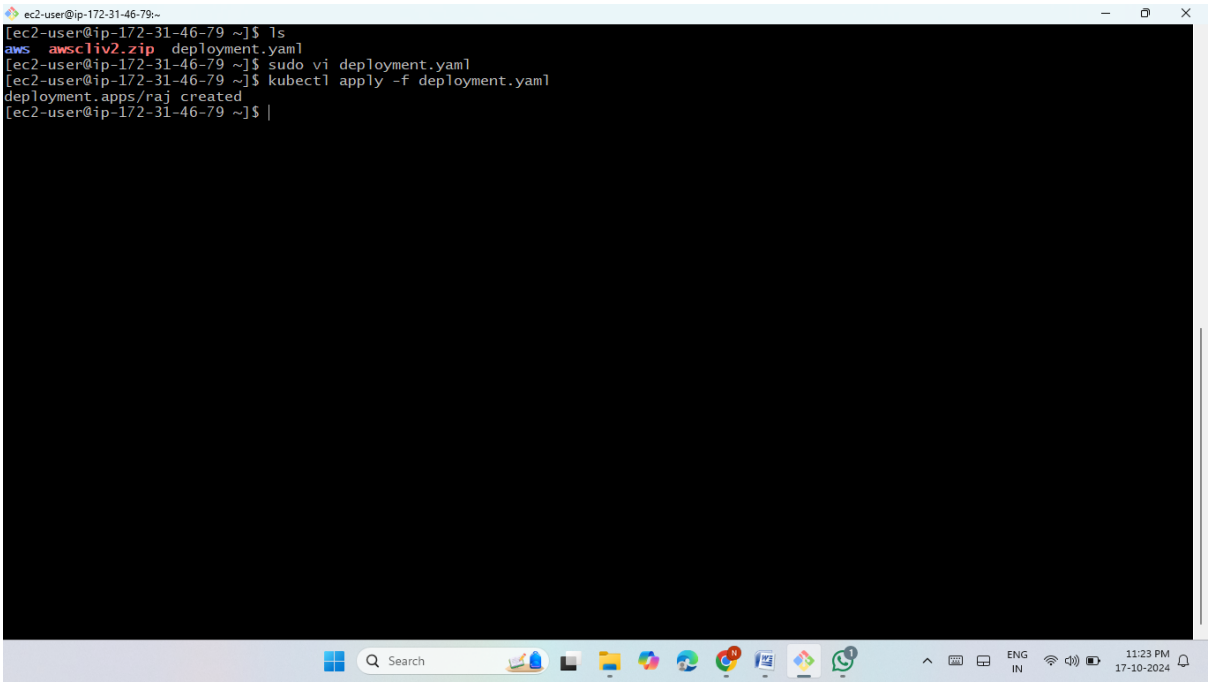


- Now create a deployment.yaml file for run the dockerhub image by using **declarative manifest method**.

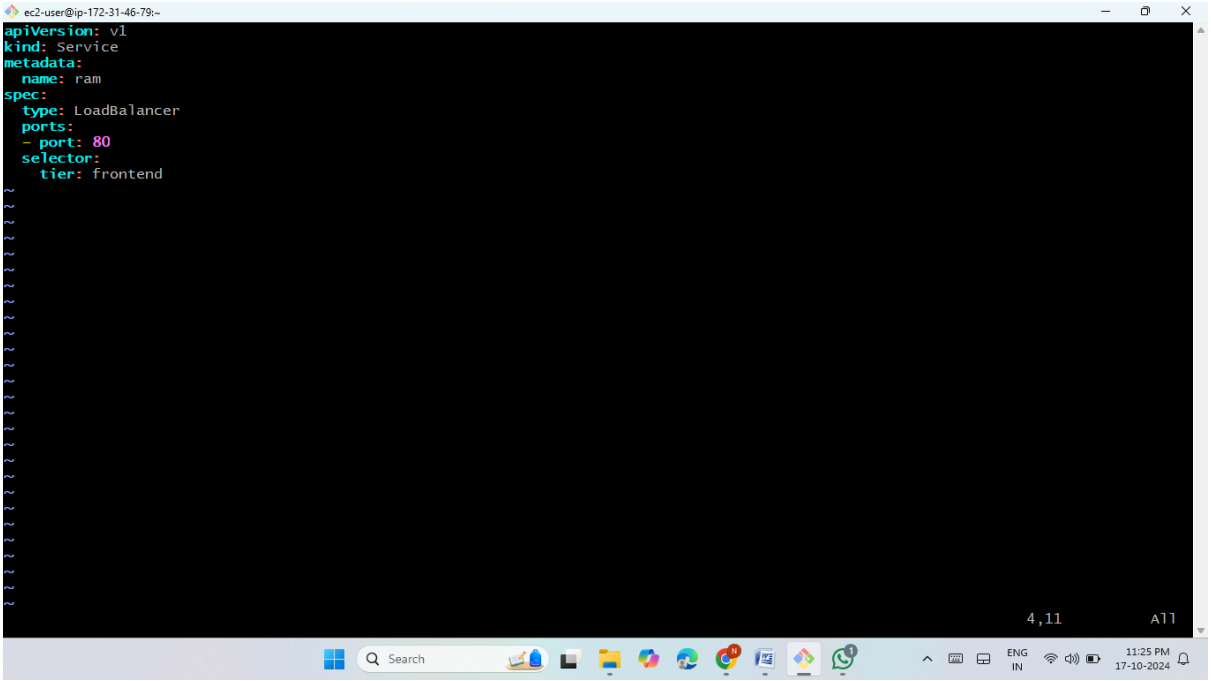


- Now run the following deployment.yaml file with the command.

  kubectl apply –f deployment.yaml



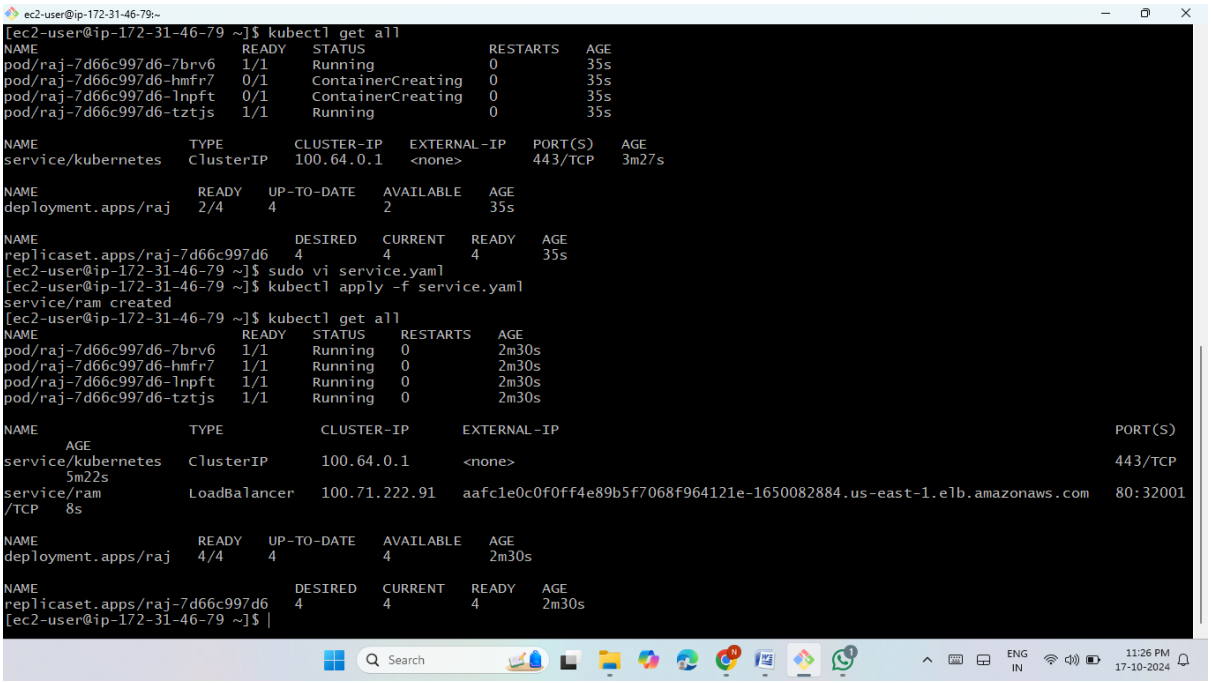
- Now create a service.yaml file for run the dockerhub image by using **declarative manifest method**.



- Now run the following service.yaml file with the command.

kubectl apply –f service.yaml

kubectl get all



- Now with help of LoadBalancer DNS link the wordpress was hosted successfully.