PROJECT-2

Wordpress Application Deployment Using Multiple Methods

Method-1

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

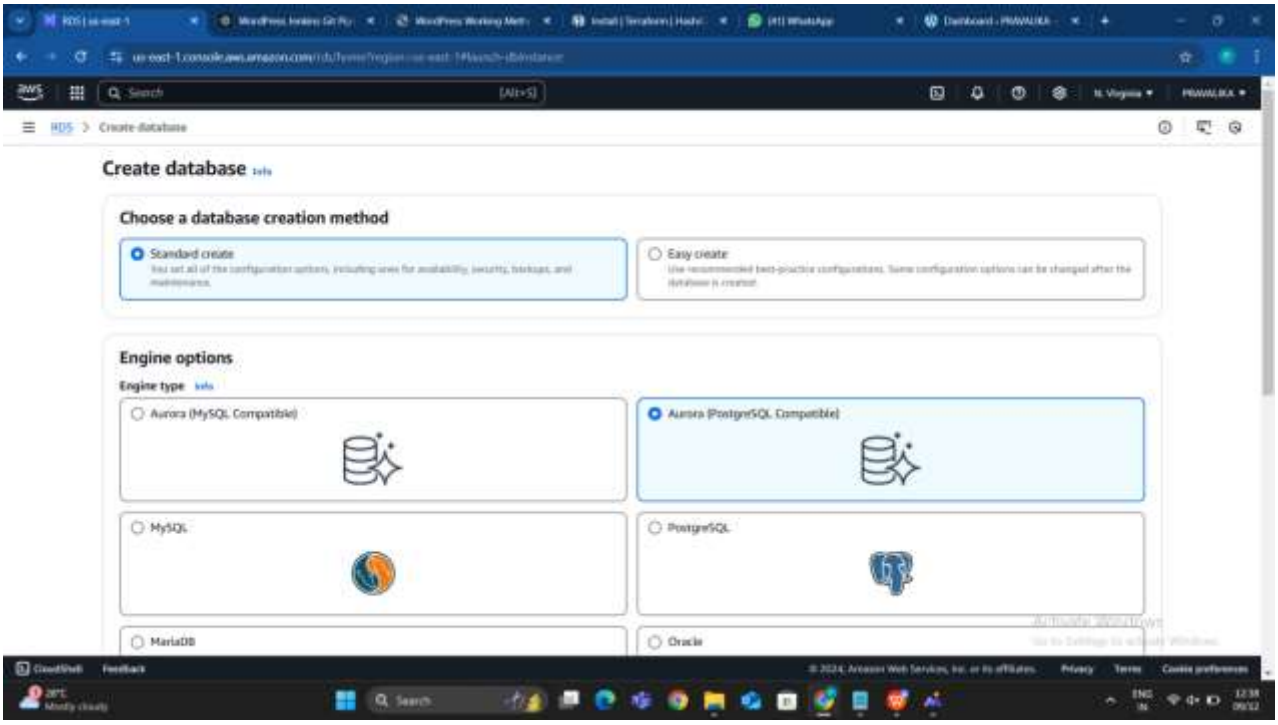
Step 1: Set Up an RDS MySQL Instance

Log in to AWS Console:

Go to the [AWS Management Console]

Create RDS MySQL Instance:

- Navigate to **RDS** → Click **Create database**.



- **Choose a database creation method**: Select **Standard create**.
- **Engine options**: Select **MySQL**.
- **Version**: Choose the latest version of MySQL.

Database Settings:

- **DB instance class**: Choose a free-tier eligible instance like db.t3.micro.
- **Storage**: Use default values or increase storage as needed.
- **DB instance identifier**: Enter a name (e.g., wordpress-rds).
- **Master username**: Enter a username (e.g., admin).
- **Master password**: Set a strong password (e.g., admin123).

· **Connectivity**:

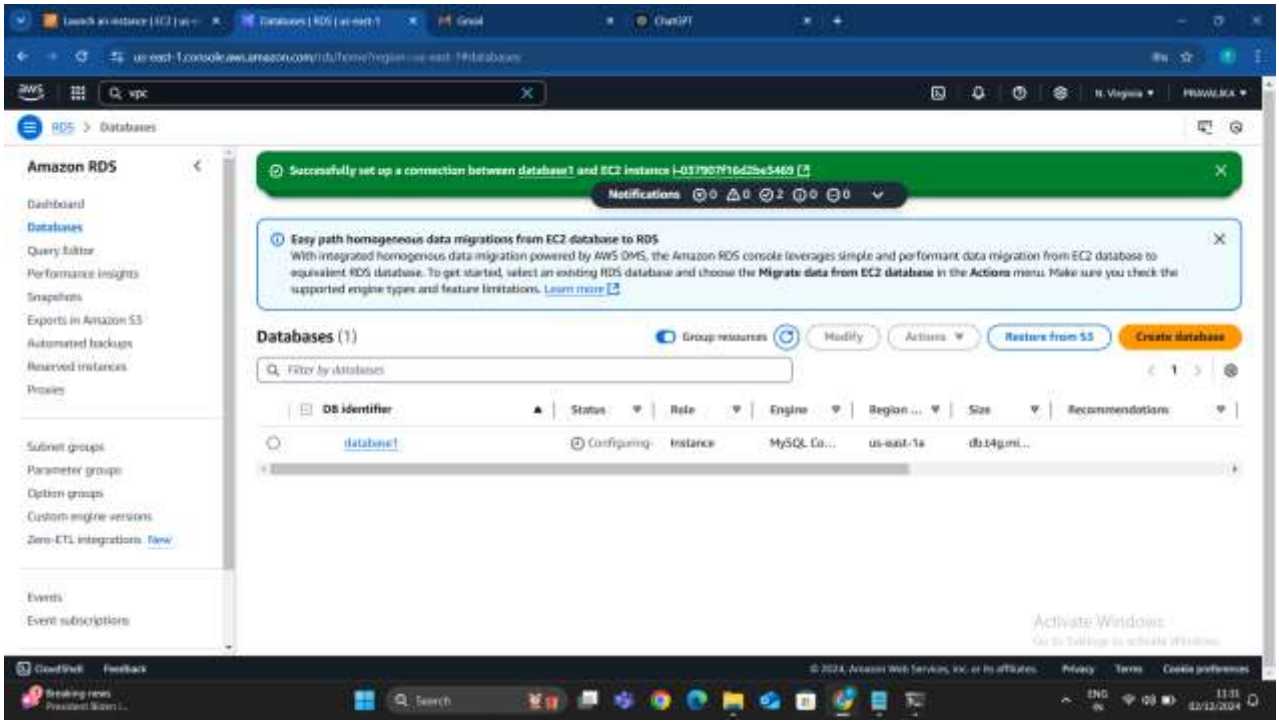
- **VPC**: Use the default VPC (or select an existing one).
- **Public access**: Enable if you want external access to the database.
- **Subnet group**: Use the default subnet group.
- **Security group**:
  - Choose an existing one or create a new one.
  - Ensure it allows **MySQL (TCP port 3306)** from the EC2 instance's private or public IP.

· **Additional Configurations**:

- · **Database authentication**: Use password authentication.
- **Monitoring and backups**: Enable as needed.

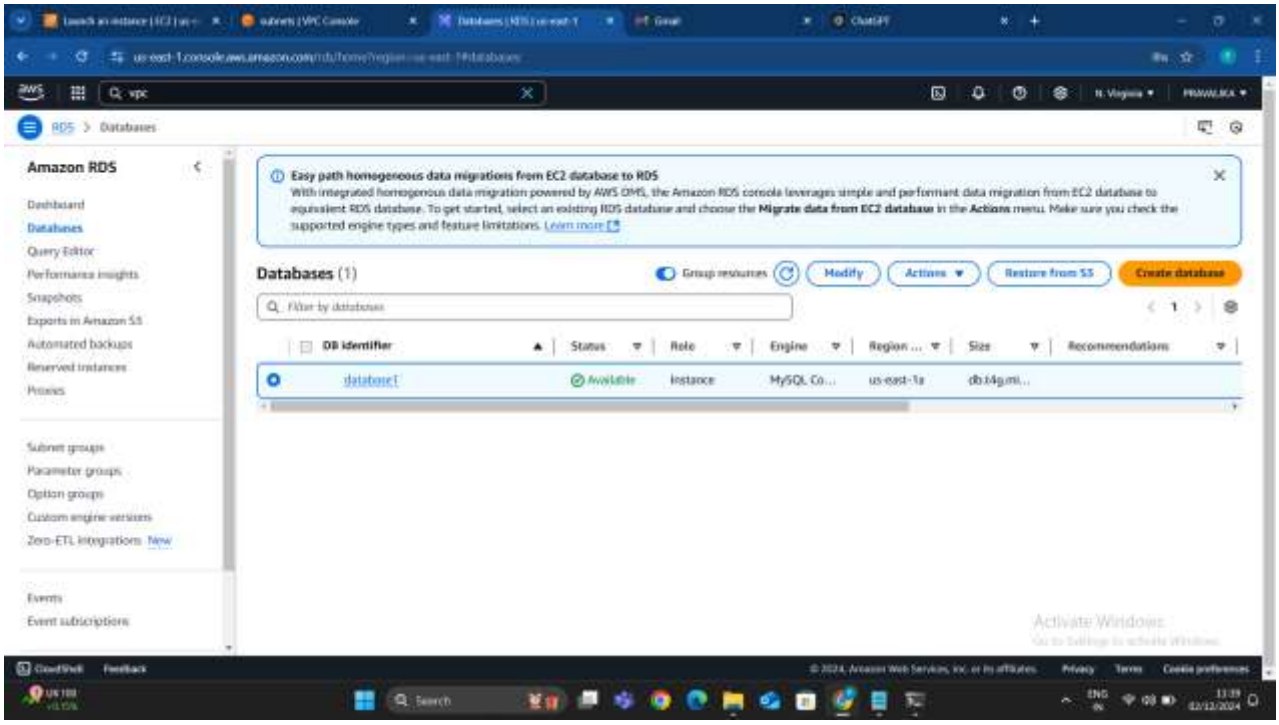
· **Create Database**:

- · Review the settings and click **Create database**.



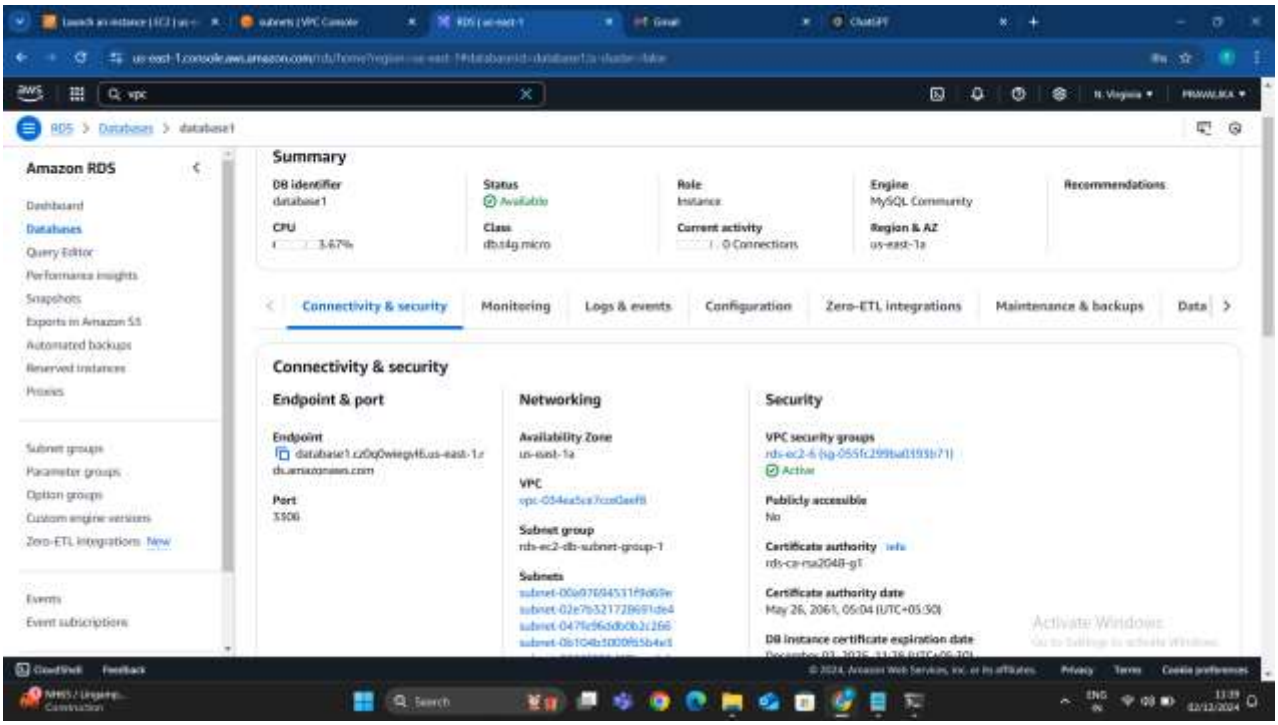
- Wait for the instance to become available.

Here database is created that is in available state



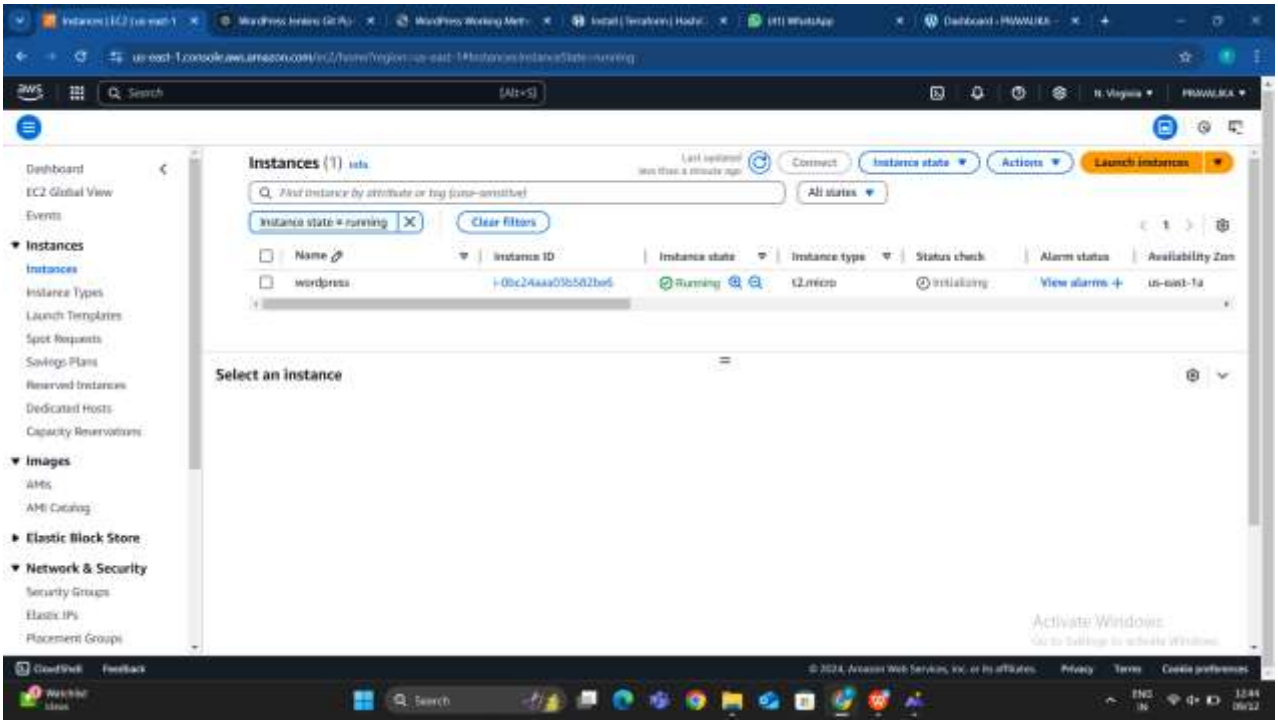
Note Down RDS Endpoint:

- Go to the **RDS instance details** page and copy the **Endpoint** (e.g., wordpress-rds.abc123.us-east-1.rds.amazonaws.com).



Step 2: Launch an EC2 Instance for WordPress

- ❖ Now create the EC2 instance by selecting EC2 services and launch the instance by selecting Amazon linux-2 version and giving the security group with SSH(22) and HTTP(80).



- ❖ Now connect the virtual server through the Git bash as shown in below.

Access the Virtual Server via Git Bash:

Open Git Bash and connect to the EC2 instance using SSH. For example

ssh -i "your-key.pem" ec2-user@your-ec2-instance-public-ip

- Replace "your-key.pem" with your private key file and your-ec2-instance-public-ip with your EC2 instance's public IP address.

- ❖ 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>.

Update the Linux Version:

Once connected to the EC2 instance, update the Linux packages to the latest version using the following command:

sudo yum -y update

- ❖ Now to allow certain traffic from EC2 instance into RDS database for that go to RDS services and select your created database.

Install MySQL:

After updating the system, install MySQL by running:

```
sudo yum -y install mysql
```

- Now go inside the created database and go to the security under this option there is a security group id click on that.

Allowing Traffic from the EC2 Instance to the RDS Database

Access the RDS Database Security Settings:

1. Navigate to the AWS Management Console and go to **RDS Services**.
2. Select the RDS database you created.

- Now go to inbound rules and click on the edit inbound rules.

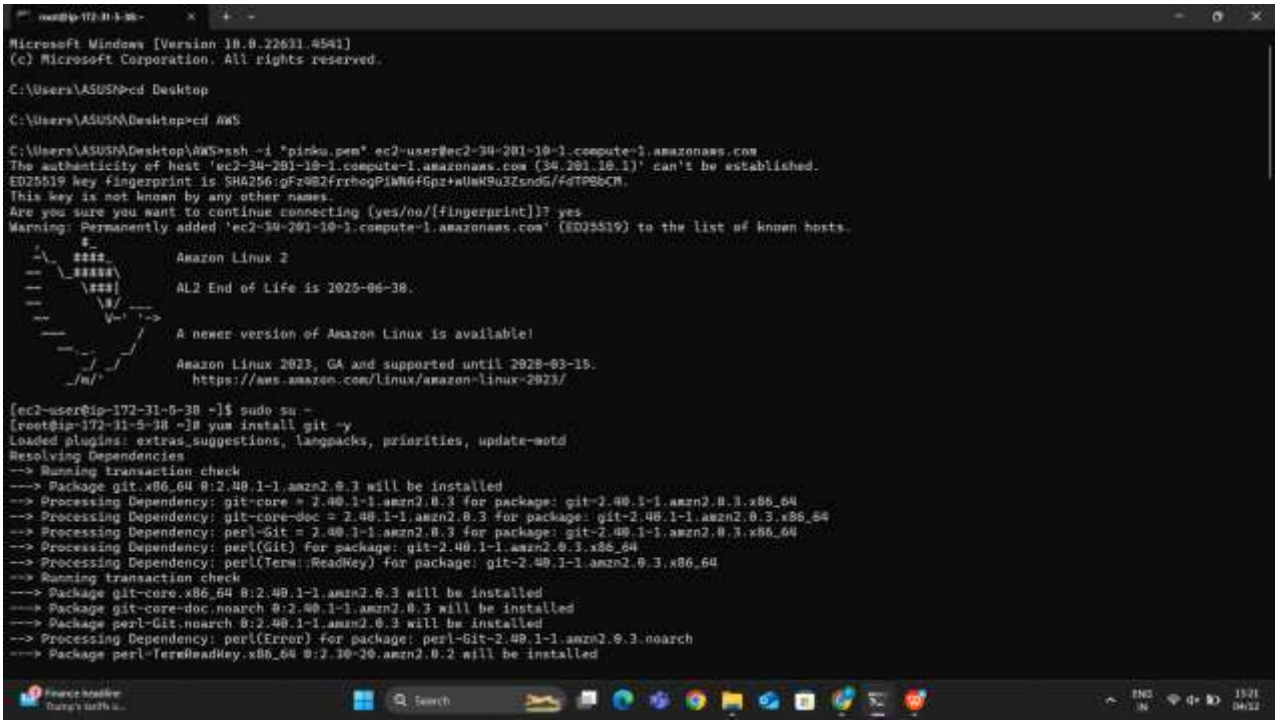
Modify the Security Group Rules:

- Under the database details, locate the **Security** section and find the **Security Group ID** associated with the RDS instance.
- Click on the **Security Group ID** link to open the security group settings.

- Now go to source option click on dropdown option select the EC2 instance security group id and click save rules.

Allow EC2 to Access RDS:

- For the **Source** field in the new rule, select the **EC2 Instance's Security Group ID** from the dropdown menu.
- Set the **Port Range** to the appropriate port for MySQL (e.g., 3306).
- Add a brief description, such as “Allow traffic from EC2 to RDS.”



Set the MySQL Host Endpoint:

- To connect to the MySQL database, you first need to export the database's endpoint address. Use the following command:

❖ 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.

❖ Replace <endpoint address> with the actual endpoint of your RDS database. You can find this in the **Endpoint** section of your RDS database in the AWS Management Console. Copy the endpoint and use it in the command.

Connect to the MySQL Database with Credentials:

- Use the following command to connect to the database by providing your username, password, and database name:

❖ Now give the credentials of database by giving a command as "mysql –user=<username> --password=<password> database-name".

· Replace <username>, <password>, and <database-name> with the actual values configured for your database.

❖ There id another command to access your database as "mysql -h <endpoint address > -u <user> -p" press eneter button it asks the password enter it and press the enter button.

❖ Now create a database user for your wordpress application and give it permission to access

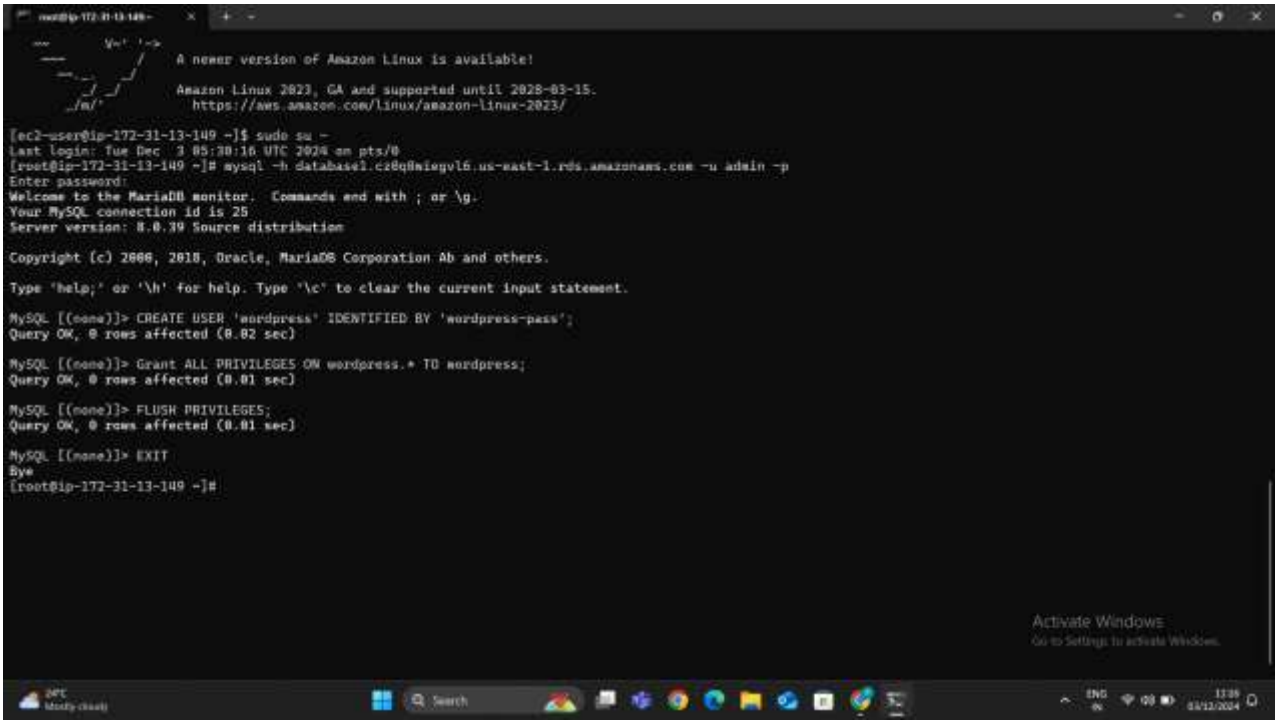
the “wordpress” database. By using this commands as

- CREATE USER 'wordpress' IDENTIFIED BY 'wordpress-pass';
- GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpress';
- FLUSH PRIVILEGES;

EXIT;

Summary Description

- **Accessing MySQL Database:** The database can be accessed using either the MYSQL_HOST variable or by directly specifying the endpoint in the connection command. Credentials such as username and password are required.
- **Creating a Database User:** A new user named wordpress is created for the WordPress application. This user is given all necessary permissions to access and manage the wordpress database.
- **Granting Permissions:** The GRANT ALL PRIVILEGES command ensures the user has full access to the database. The FLUSH PRIVILEGES command applies these changes immediately.

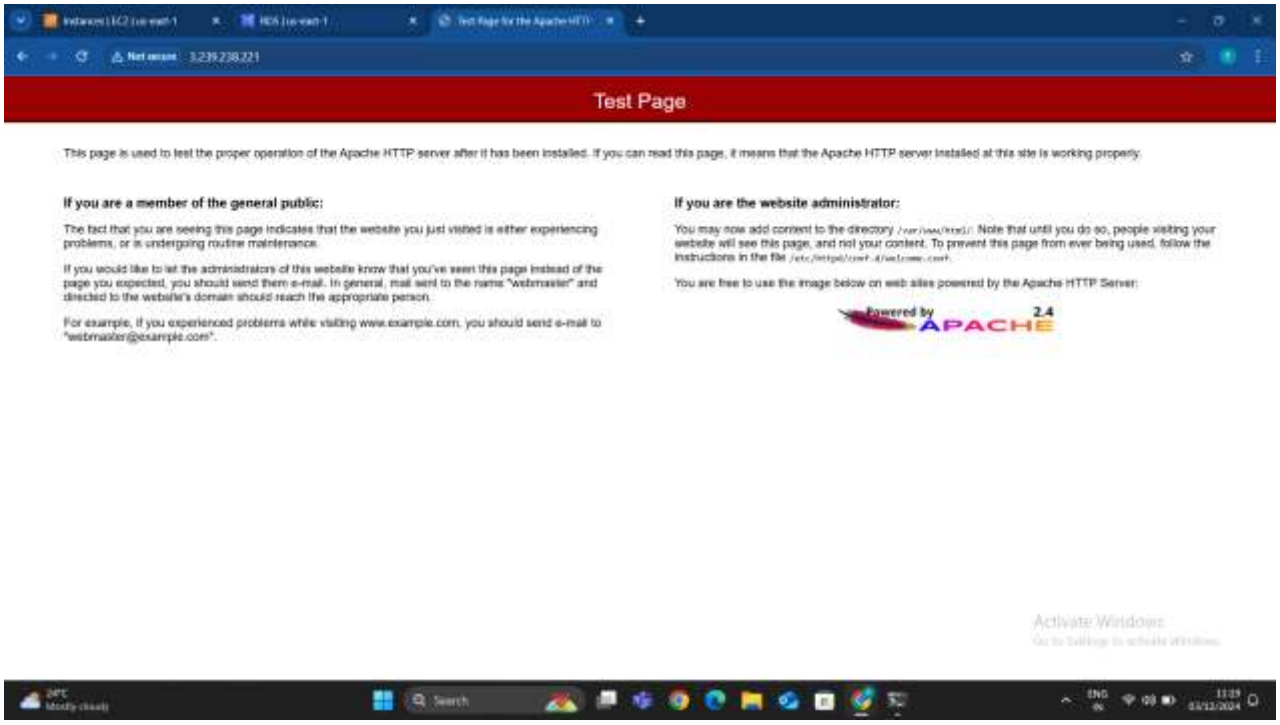


- To host word press application we need a apache web server httpd install that by giving a command as “sudo yum -y install httpd” and start and enable the httpd service by giving the commands as

“sudo service httpd start” (or) “sudo systemctl start httpd”

“sudo chkconfig httpd on” (or) “sudo systemctl enable httpd”

- Now go to EC2 instance and copy public ip and paste it on google browse it and check the official page of httpd is displaying or not.



- Now go to broser and search as download word press and click on proper link and select and copy the address link of word press download file and paste that along with wget command in git bash as “wget <address link of word press download file?”
- It gives the zip file to unzip that file by using a command as “unzip <zipfile>”
- To run wordpress web application, you have to install run time of word press web application as php language with the following command <sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2> otherwise update your EC2 instance with the following command as “<sudo yum -y update and sudo yum -y upgrade>.

```
  inflating: wordpress/wp-admin/install.php
  inflating: wordpress/wp-admin/admin-header.php
  inflating: wordpress/wp-admin/post-new.php
  inflating: wordpress/wp-admin/themes.php
  inflating: wordpress/wp-admin/options-reading.php
  inflating: wordpress/wp-trackback.php
  inflating: wordpress/wp-comments-post.php
[root@ip-172-31-13-149 ~]# ll
total 27920
-rw-r--r-- 1 root root 28585184 Nov 21 14:08 latest.zip
drwxr-xr-x 5 root root     4096 Nov 21 14:07 wordpress
[root@ip-172-31-13-149 ~]# sleep 3000
^Z
[2]+  Stopped                 sleep 3000
[root@ip-172-31-13-149 ~]# sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
Topic lamp-mariadb10.2-php7.2 has end-of-support date of 2020-11-30
Topic php7.2 has end-of-support date of 2020-11-30
Installing php-pdo, php-mysqlnd, php-fpm, php-cli, php-json, mariadb
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Cleaning repos: amzn2-core amzn2extra-docker amzn2extra-kernel-5.10 amzn2extra-lamp-mariadb10.2-php7.2 amzn2extra-php7.2
17 metadata files removed
6 sqlite files removed
0 metadata files removed
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core                                                               | 3.6 kB  00:00:00
amzn2extra-docker                                                        | 2.9 kB  00:00:00
amzn2extra-kernel-5.10                                                   | 3.0 kB  00:00:00
amzn2extra-lamp-mariadb10.2-php7.2                                       | 3.0 kB  00:00:00
amzn2extra-php7.2                                                        | 3.0 kB  00:00:00
(1/11): amzn2-core/2/x86_64/group_gz                                     | 2.7 kB  00:00:00
(2/11): amzn2-core/2/x86_64/updateinfo                                   | 1.0 MB  00:00:00
(3/11): amzn2extra-docker/2/x86_64/updateinfo                            |  20 kB  00:00:00
(4/11): amzn2extra-docker/2/x86_64/primary_db                            | 116 kB  00:00:00
(5/11): amzn2extra-lamp-mariadb10.2-php7.2/2/x86_64/updateinfo           |   76 B  00:00:00
(6/11): amzn2extra-php7.2/2/x86_64/updateinfo                            |   76 B  00:00:00
(7/11): amzn2extra-kernel-5.10/2/x86_64/updateinfo                       |   91 kB  00:00:00
(8/11): amzn2extra-lamp-mariadb10.2-php7.2/2/x86_64/primary_db           | 506 kB  00:00:00
(9/11): amzn2extra-php7.2/2/x86_64/primary_db                            | 580 kB  00:00:00
(10/11): amzn2extra-kernel-5.10/2/x86_64/primary_db                      | 1.2 MB  00:00:00
(11/11): amzn2-core/2/x86_64/primary_db                                  | 71 MB  00:00:01
```

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

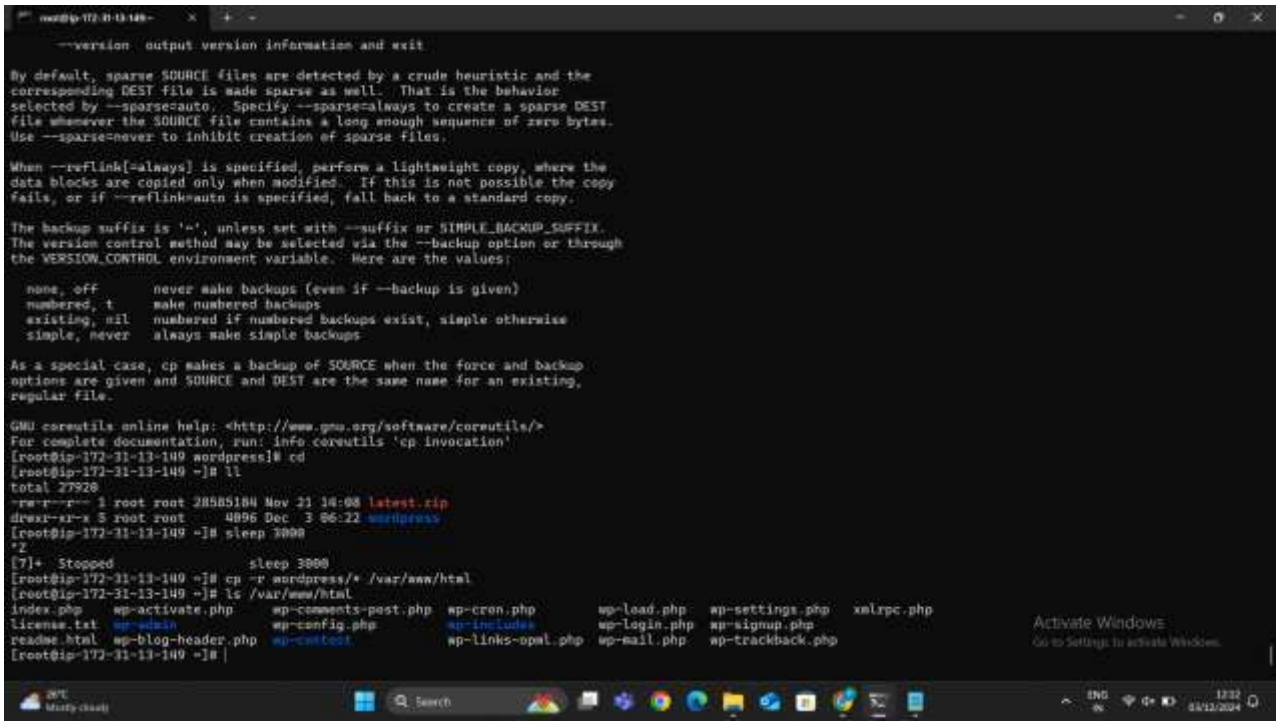
```
      --version  output version information and exit

By default, sparse SOURCE files are detected by a crude heuristic and the
corresponding DEST file is made sparse as well.  That is the behavior
selected by --sparse=auto.  Specify --sparse=always to create a sparse DEST
file whenever the SOURCE file contains a long enough sequence of zero bytes.
Use --sparse=never to inhibit creation of sparse files.

When --reflink[=always] is specified, perform a lightweight copy, where the
data blocks are copied only when modified.  If this is not possible the copy
fails, or if --reflink=auto is specified, fall back to a standard copy.

The backup suffix is '~', unless set with --suffix or SIMPLE_BACKUP_SUFFIX.
The version control method may be selected via the --backup option or through
the VERSION_CONTROL environment variable.  Here are the values:

  none, off       never make backups (even if --backup is given)
  numbered, t     make numbered backups
  existing, nil   numbered if numbered backups exist, simple otherwise
  simple, never   always make simple backups

As a special case, cp makes a backup of SOURCE when the force and backup
options are given and SOURCE and DEST are the same name for an existing,
regular file.

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
For complete documentation, run: info coreutils 'cp invocation'
[root@ip-172-31-13-149 wordpress]# cd
[root@ip-172-31-13-149 ~]# ll
total 27920
-rw-r--r-- 1 root root 28585184 Nov 21 14:08 latest.zip
drwxr-xr-x 5 root root     4096 Dec  3 06:22 wordpress
[root@ip-172-31-13-149 ~]# sleep 3000
^Z
[7]+  Stopped                 sleep 3000
[root@ip-172-31-13-149 ~]# cp -r wordpress/* /var/www/html
[root@ip-172-31-13-149 ~]# ls /var/www/html
index.php    wp-activate.php     wp-comments-post.php  wp-cron.php        wp-load.php   wp-settings.php   xmlrpc.php
license.txt  wp-admin            wp-config.php         wp-includes        wp-login.php  wp-signup.php
readme.html  wp-blog-header.php  wp-content            wp-links-opml.php  wp-mail.php   wp-trackback.php
[root@ip-172-31-13-149 ~]#
```

- Now do some configurations in word press configuration file by giving database name, username, password and host name for that execute a command as “sudo vi wp-config.php”
- Search in google “Wordpress secret key generator”. And use trhat password.

❖ Now copy this wordpress directory to the document root directory to host your web application of wordpress by giving a command as

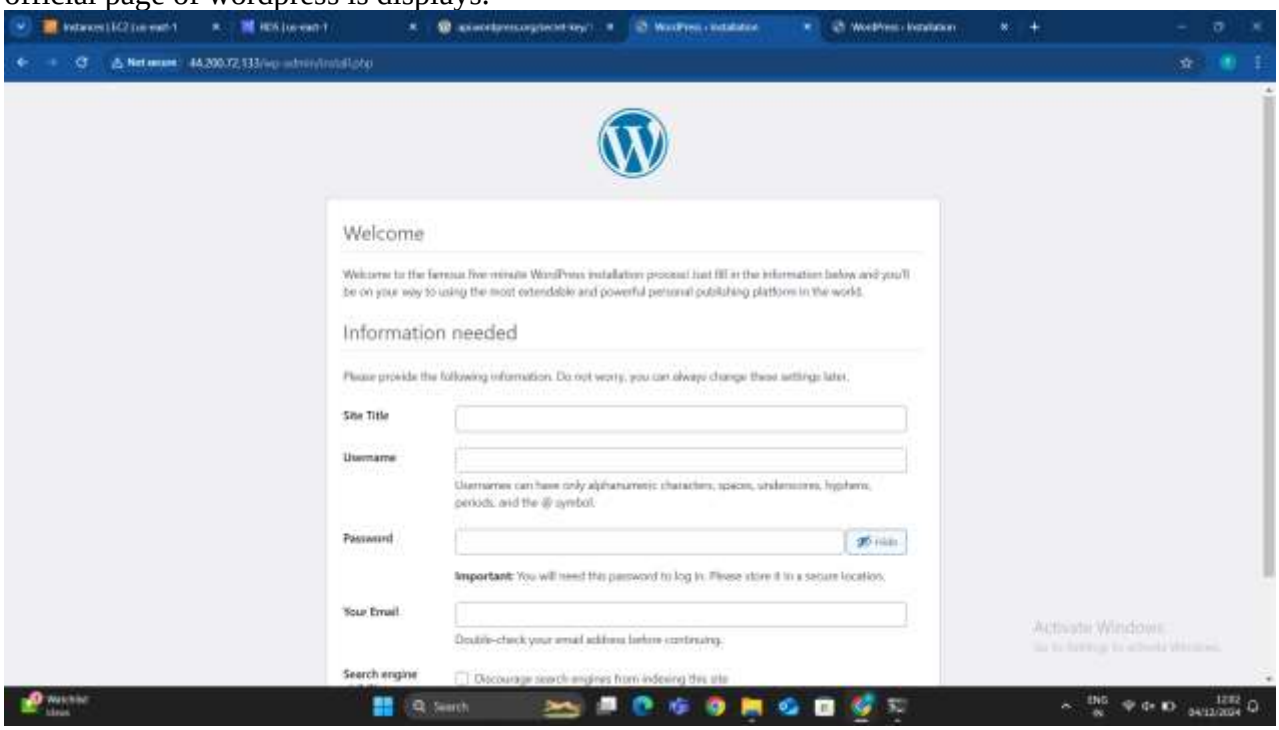
"sudo cp -r <unzip file or wordpress directory>/* /var/www/html/"

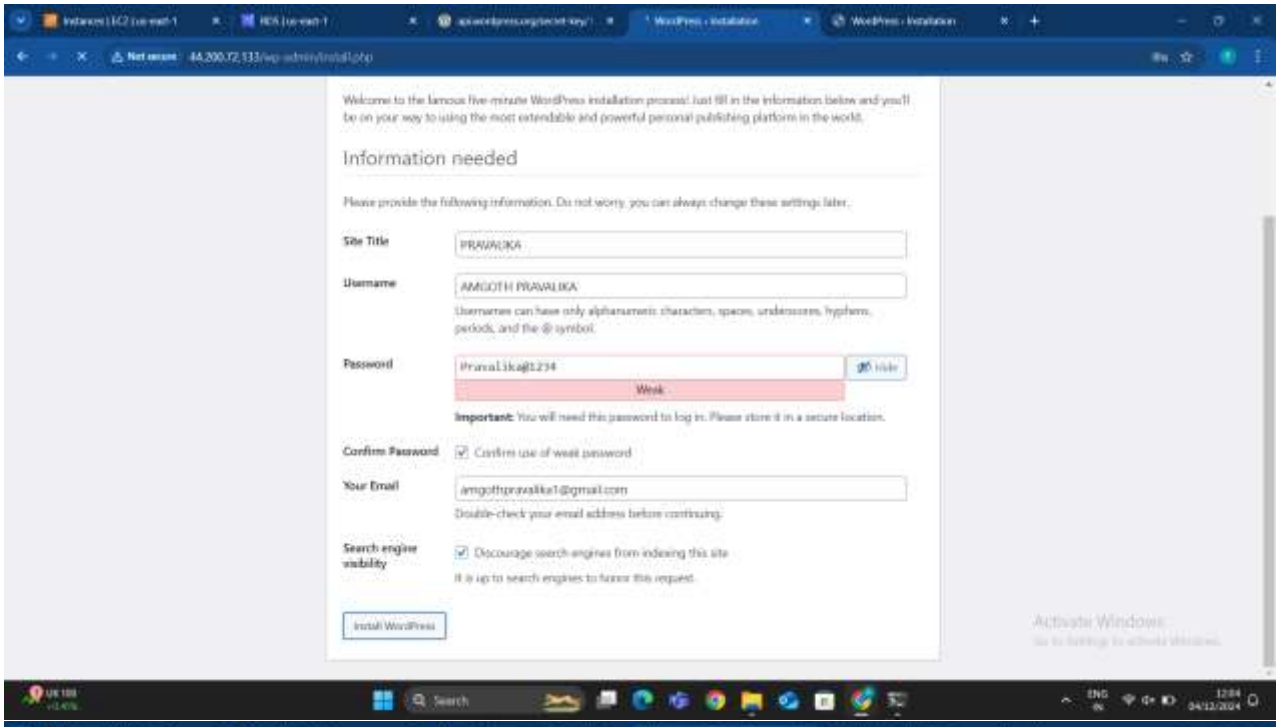
"ls /var/www/html/"

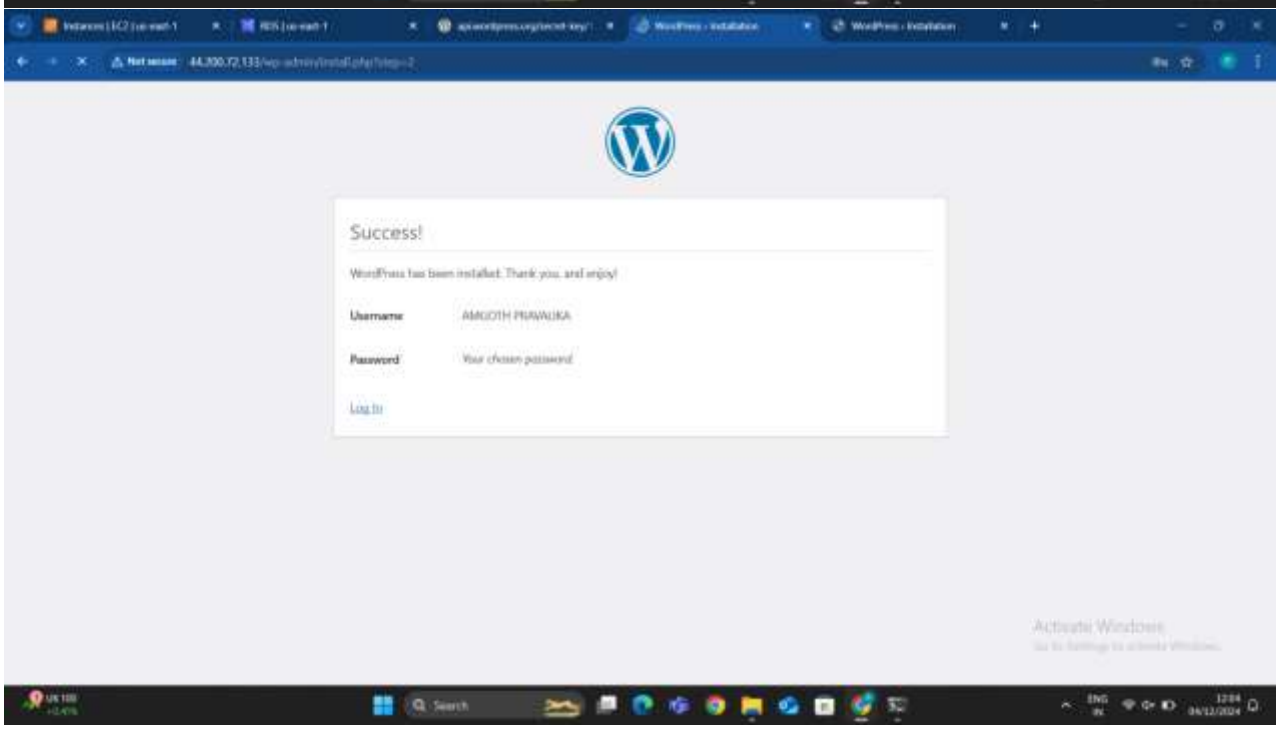


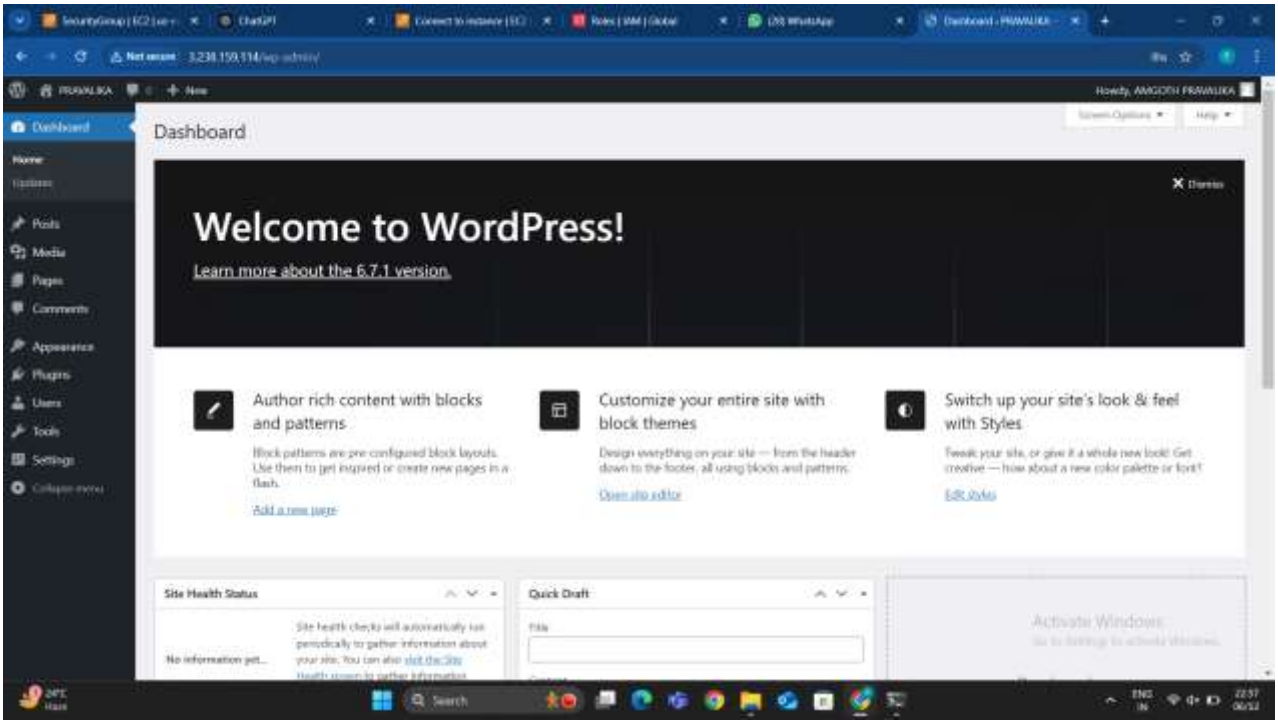
❖ Restart the httpd by giving a commandas

❖ Now go to EC2 instance and copy public ip and paste it on google browse it and check the official page of wordpress is displays.

❖ 


Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.
Information needed
Please provide the following information. Do not worry, you can always change these settings later.
Site Title
PRAVALIKA
Username
AMGOTH PRAVALIKA
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.
Password
Pravalika@1234
Hide
Weak
Important: You will need this password to log in. Please store it in a secure location.
Confirm Password
Confirm use of weak password
Your Email
amgothpravalika1@gmail.com
Double-check your email address before continuing.
Search engine visibility
Discourage search engines from indexing this site
It is up to search engines to honor this request.
Install WordPress


Success!
WordPress has been installed. Thank you, and enjoy!
Username
AMGOTH PRAVALIKA
Password
Your chosen password.
Log in



METHOD-2

CREATING AND LAUNCH AN AMAZON EC2 INSTANCE:

- ❖ First login to the AWS account with credentials.

Navigate to the EC2 Dashboard

- In the AWS Management Console, search for **EC2** in the search bar and click on **EC2** under **Services**.

Launch an EC2 Instance

Click on "Launch Instance":

1. On the EC2 Dashboard, click the **Launch Instance** button.

Name Your Instance:

1. Under the "Name and tags" section, provide a name for your EC2 instance (e.g., MyFirstInstance).

Select an Amazon Machine Image (AMI):

1. In the **Application and OS Images (Amazon Machine Image)** section, choose your AMI.
    1. For Linux-based systems, select **Amazon Linux 2** or **Ubuntu 22.04 LTS (Free Tier Eligible)**.
2. Ensure the AMI is free-tier eligible if you're using the AWS free tier.

Choose an Instance Type:

1. Select **t2.micro** (free-tier eligible) in the **Instance Type** section. This is sufficient for lightweight tasks.

Configure Key Pair (Login):

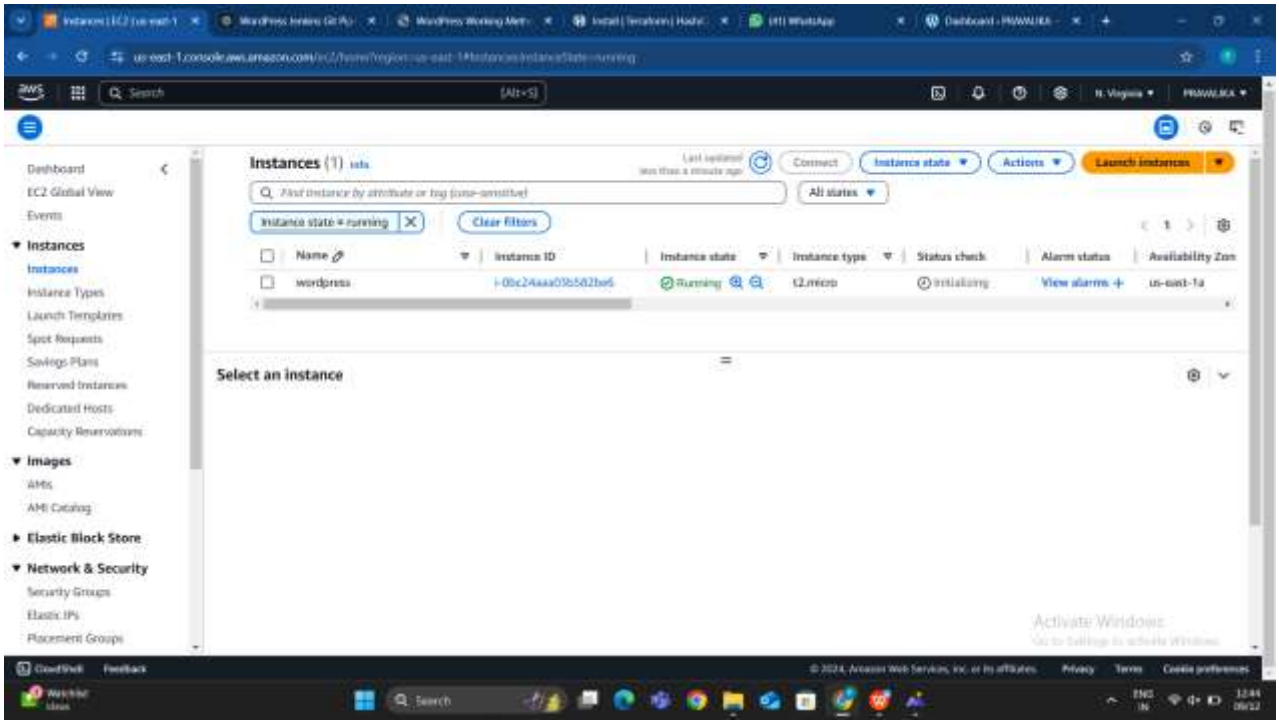
1. In the **Key pair (login)** section, select an existing key pair or create a new one:
    1. To create a new key pair, click **Create new key pair**, name it, and download the .pem file. Keep this file safe as you'll need it to connect to your instance.

Network Settings:

1. The default network settings will suffice for most cases, but ensure the following:
    1. **Auto-assign Public IP** is enabled.
    2. Add a security group rule to allow SSH (port 22) access from your local IP address.

Storage Settings:

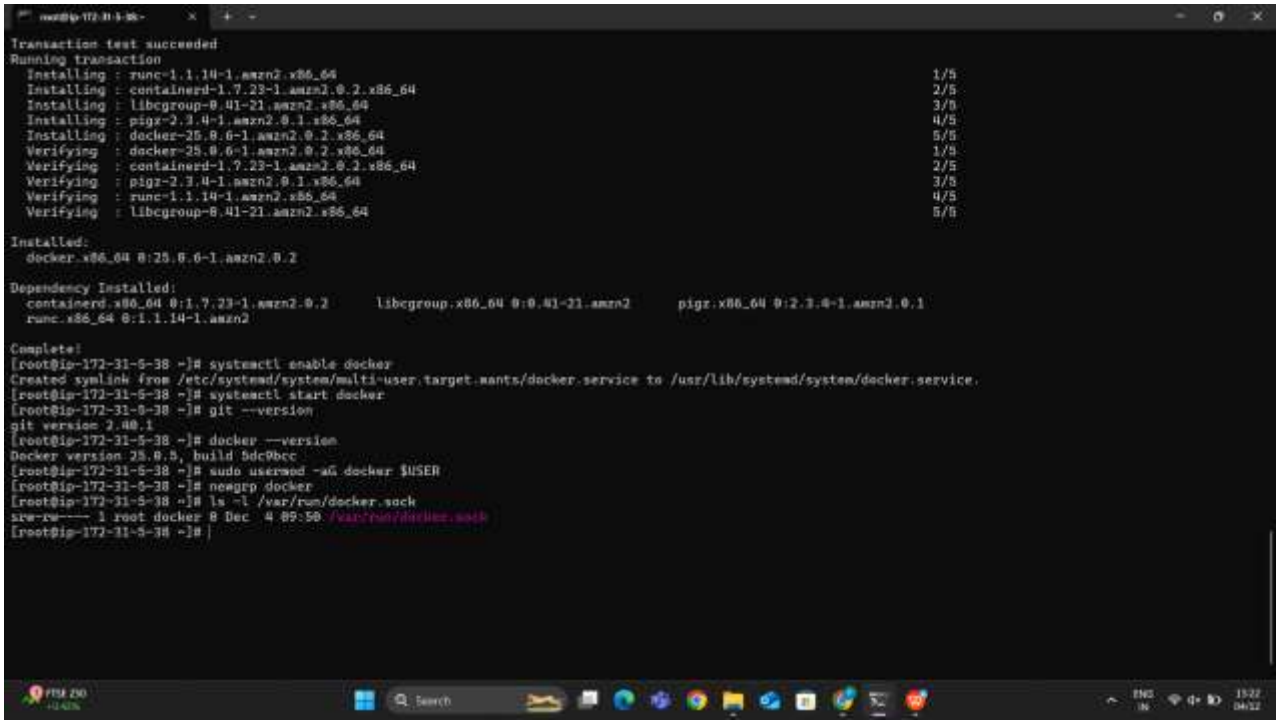
1. Keep the default storage settings (e.g., 8 GiB) unless you have specific requirements.


Instances (1) info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zon
wordpress
i-0bc24aaa09b582be6
Running
t2.micro
Initializing
View alarms +
us-east-1a
Select an instance
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Network & Security
Security Groups
Elastic IPs
Placement Groups
Activate Windows

```
Microsoft Windows [Version 10.0.22631.4541]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUSN>cd Desktop

C:\Users\ASUSN\Desktop>cd AWS

C:\Users\ASUSN\Desktop\AWS>ssh -i "pinku.pem" ec2-user@ec2-34-201-10-1.compute-1.amazonaws.com
The authenticity of host 'ec2-34-201-10-1.compute-1.amazonaws.com (34.201.10.1)' can't be established.
ED25519 key fingerprint is SHA256:gFz9B2frrhogPiWN6fGpz+mUmK9u3ZsndG/fdTPBbCM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-201-10-1.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
       __|  __|_  )
       _| (     /   Amazon Linux 2
      ___|\___|___|
      https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-5-38 ~]$ sudo su -
[root@ip-172-31-5-38 ~]# yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package git.x86_64 0:2.40.1-1.amzn2.0.3 will be installed
--> Processing Dependency: git-core = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: git-core-doc = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl-Git = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Git) for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Running transaction check
---> Package git-core.x86_64 0:2.40.1-1.amzn2.0.3 will be installed
---> Package git-core-doc.noarch 0:2.40.1-1.amzn2.0.3 will be installed
---> Package perl-Git.noarch 0:2.40.1-1.amzn2.0.3 will be installed
--> Processing Dependency: perl(Error) for package: perl-Git-2.40.1-1.amzn2.0.3.noarch
---> Package perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2 will be installed
```

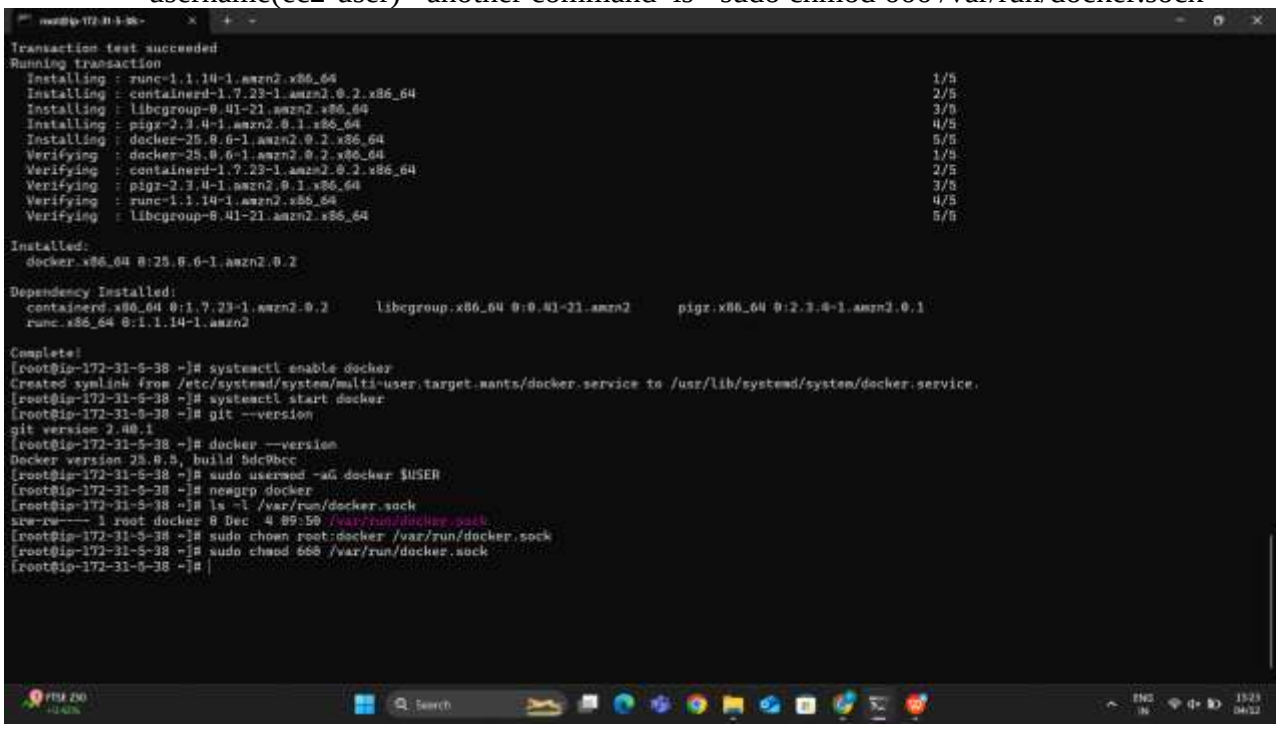
Install git, docker and setup docker environment. And provide permissions for admint socket file.

```
Transaction test succeeded
Running transaction
  Installing : runc-1.1.14-1.amzn2.x86_64                                   1/5
  Installing : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Installing : libcgroup-0.41-21.amzn2.x86_64                               3/5
  Installing : pigz-2.3.4-1.amzn2.0.1.x86_64                                4/5
  Installing : docker-25.0.6-1.amzn2.0.2.x86_64                             5/5
  Verifying  : docker-25.0.6-1.amzn2.0.2.x86_64                             1/5
  Verifying  : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Verifying  : pigz-2.3.4-1.amzn2.0.1.x86_64                                3/5
  Verifying  : runc-1.1.14-1.amzn2.x86_64                                   4/5
  Verifying  : libcgroup-0.41-21.amzn2.x86_64                               5/5

Installed:
  docker.x86_64 0:25.0.6-1.amzn2.0.2

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

Complete!
[root@ip-172-31-5-38 ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@ip-172-31-5-38 ~]# systemctl start docker
[root@ip-172-31-5-38 ~]# git --version
git version 2.40.1
[root@ip-172-31-5-38 ~]# docker --version
Docker version 25.0.5, build 5dc9bcc
[root@ip-172-31-5-38 ~]#
```



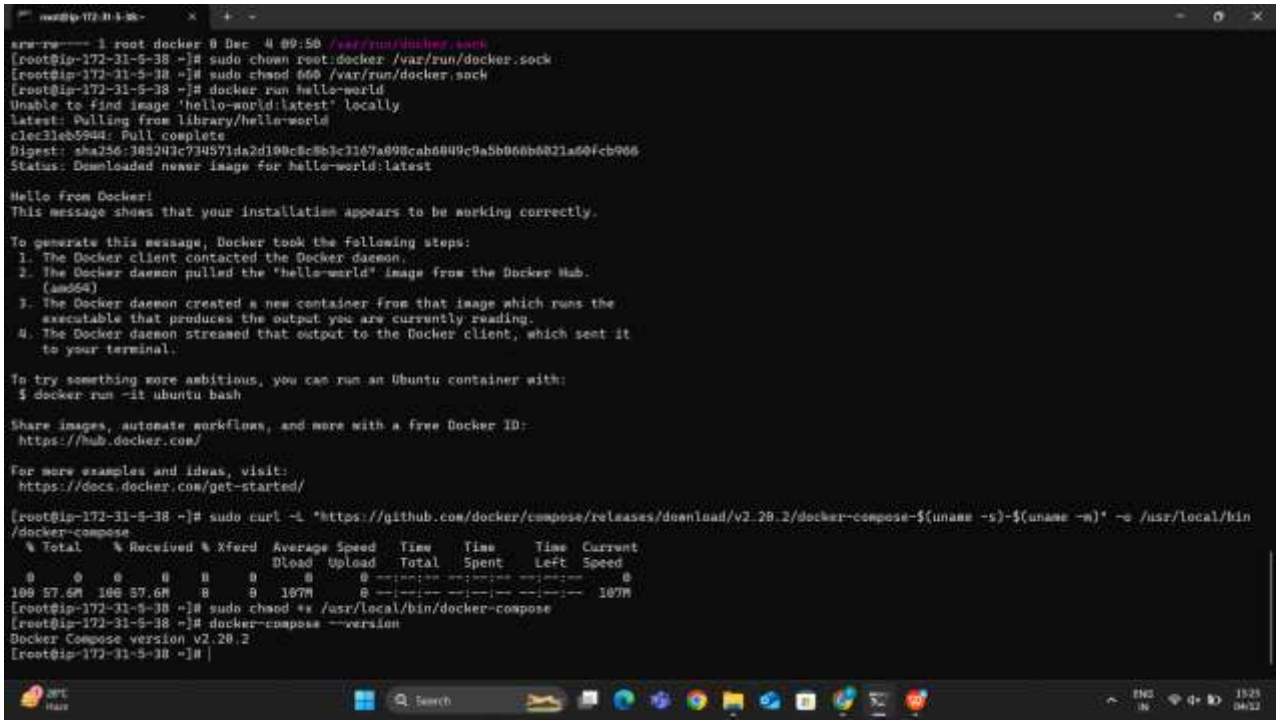
```
Transaction test succeeded
Running transaction
  Installing : runc-1.1.14-1.amzn2.x86_64                                   1/5
  Installing : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Installing : libcgroup-0.41-21.amzn2.x86_64                               3/5
  Installing : pigz-2.3.4-1.amzn2.0.1.x86_64                                4/5
  Installing : docker-25.0.6-1.amzn2.0.2.x86_64                             5/5
  Verifying  : docker-25.0.6-1.amzn2.0.2.x86_64                             1/5
  Verifying  : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Verifying  : pigz-2.3.4-1.amzn2.0.1.x86_64                                3/5
  Verifying  : runc-1.1.14-1.amzn2.x86_64                                   4/5
  Verifying  : libcgroup-0.41-21.amzn2.x86_64                               5/5

Installed:
  docker.x86_64 0:25.0.6-1.amzn2.0.2

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

Complete!
[root@ip-172-31-5-38 ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@ip-172-31-5-38 ~]# systemctl start docker
[root@ip-172-31-5-38 ~]# git --version
git version 2.40.1
[root@ip-172-31-5-38 ~]# docker --version
Docker version 25.0.5, build 5dc9bcc
[root@ip-172-31-5-38 ~]# sudo usermod -aG docker $USER
[root@ip-172-31-5-38 ~]# newgrp docker
[root@ip-172-31-5-38 ~]# ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Dec  4 09:50 /var/run/docker.sock
[root@ip-172-31-5-38 ~]#
```

❖ Now give permissions to add limited linux user account to docker group by using a command as “<sudo usermod -aG username(ec2-user)> or <sudo usermod -a -G username(ec2-user)> another command is <sudo chmod 666 /var/run/docker.sock>



```
Transaction test succeeded
Running transaction
  Installing : runc-1.1.14-1.amzn2.x86_64                                   1/5
  Installing : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Installing : libcgroup-0.41-21.amzn2.x86_64                               3/5
  Installing : pigz-2.3.4-1.amzn2.0.1.x86_64                                4/5
  Installing : docker-25.0.6-1.amzn2.0.2.x86_64                             5/5
  Verifying  : docker-25.0.6-1.amzn2.0.2.x86_64                             1/5
  Verifying  : containerd-1.7.23-1.amzn2.0.2.x86_64                         2/5
  Verifying  : pigz-2.3.4-1.amzn2.0.1.x86_64                                3/5
  Verifying  : runc-1.1.14-1.amzn2.x86_64                                   4/5
  Verifying  : libcgroup-0.41-21.amzn2.x86_64                               5/5

Installed:
  docker.x86_64 0:25.0.6-1.amzn2.0.2

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

Complete!
[root@ip-172-31-5-38 ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@ip-172-31-5-38 ~]# systemctl start docker
[root@ip-172-31-5-38 ~]# git --version
git version 2.40.1
[root@ip-172-31-5-38 ~]# docker --version
Docker version 25.0.5, build 5dc9bcc
[root@ip-172-31-5-38 ~]# sudo usermod -aG docker $USER
[root@ip-172-31-5-38 ~]# newgrp docker
[root@ip-172-31-5-38 ~]# ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Dec  4 09:50 /var/run/docker.sock
[root@ip-172-31-5-38 ~]# sudo chown root:docker /var/run/docker.sock
[root@ip-172-31-5-38 ~]# sudo chmod 660 /var/run/docker.sock
[root@ip-172-31-5-38 ~]#
```

❖ Install docker compose.(go to browser and use docker compose installation)



```
srw-rw---- 1 root docker 0 Dec  4 09:50 /var/run/docker.sock
[root@ip-172-31-5-38 ~]# sudo chown root:docker /var/run/docker.sock
[root@ip-172-31-5-38 ~]# sudo chmod 660 /var/run/docker.sock
[root@ip-172-31-5-38 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fcb966
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@ip-172-31-5-38 ~]# sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin
/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 57.6M  100 57.6M    0     0   107M      0 --:--:-- --:--:-- --:--:--  107M
[root@ip-172-31-5-38 ~]# sudo chmod +x /usr/local/bin/docker-compose
[root@ip-172-31-5-38 ~]# docker-compose --version
Docker Compose version v2.20.2
[root@ip-172-31-5-38 ~]#
```

Apply Executable Permissions to the Docker Compose Binary

- ❖ Apply executable permissions to the binary by using command as <Sudo chmod +x /usr/local/bin/docker-compose>

What This Does:

- The chmod +x command grants the executable permissions to the Docker Compose binary file located at /usr/local/bin/docker-compose.
- This ensures that the binary can be executed as a program on the system.

- ❖ Now create the symbolic link by using command as <ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose>

- **What This Does:**
  - The ln -s command creates a symbolic link (shortcut) to the Docker Compose binary.
  - By linking it to /usr/bin/docker-compose, the system can recognize and execute Docker Compose from anywhere without needing the full path.

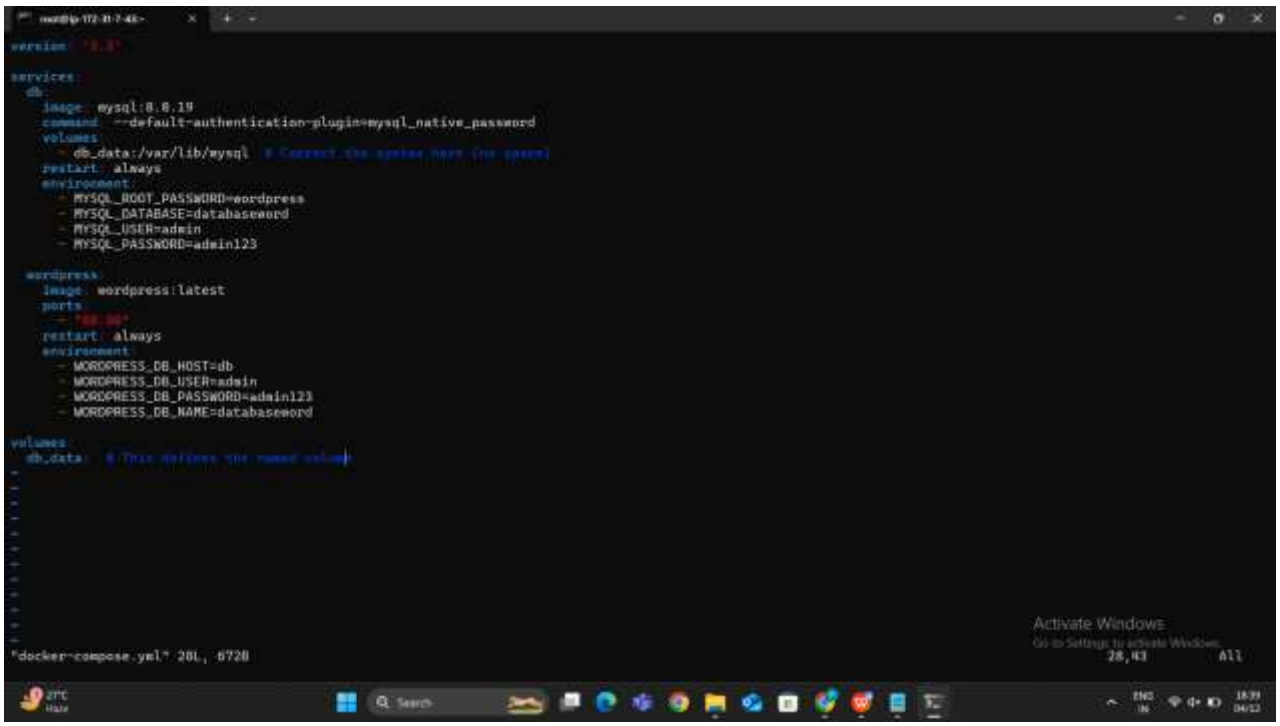
- ❖ create a file with the name of docker-compose.yml.

What is a docker-compose.yml **File?**

- It is a YAML configuration file used by Docker Compose to define and run multi-container Docker applications.
- This file specifies services, networks, and volumes required for the application.

Summary Description

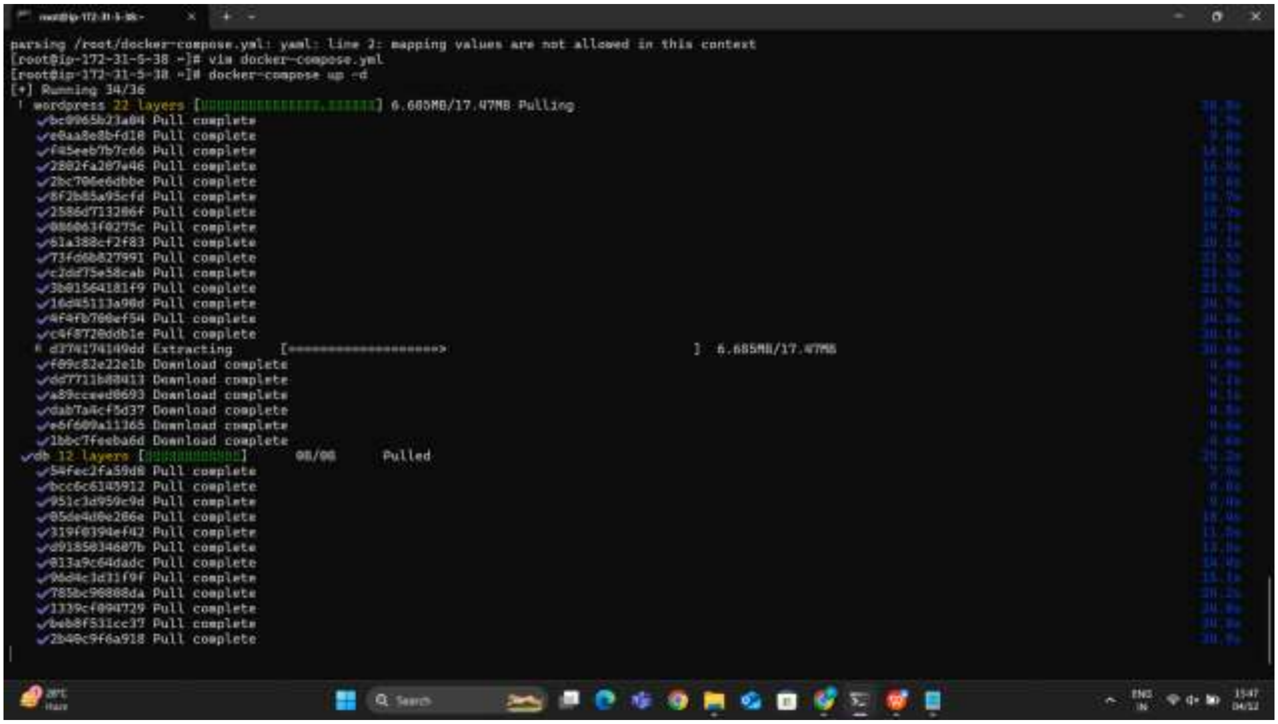
- **Granting Permissions:** Applying executable permissions to Docker Compose ensures it can run as a program.
- **Creating a Symbolic Link:** A symbolic link simplifies access to the Docker Compose binary from any directory.
- **Defining the** docker-compose.yml **File:** This file is critical for defining the services, networks, and volumes needed for containerized applications.



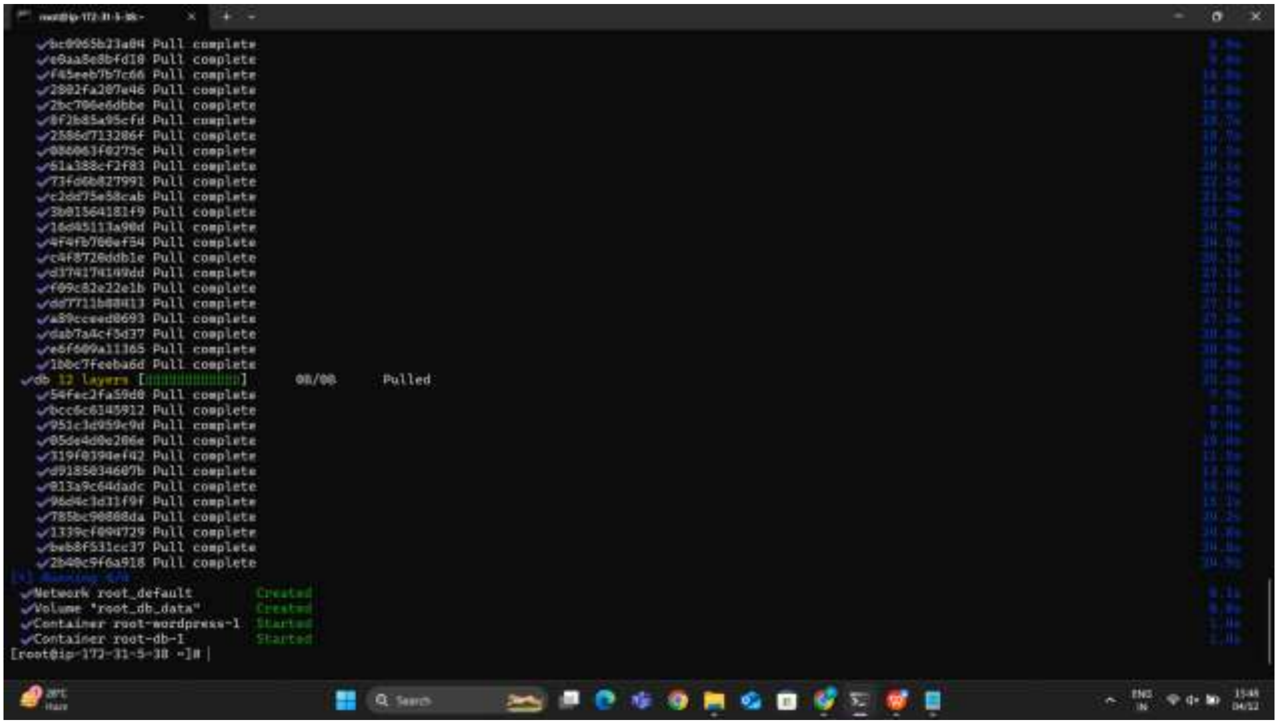
❖ Now pull the images of MYSQL and WORDPRESS we have to execute this created docker-conpose.yml file by using command as <docker-compose up -d>

What This Does:

- The docker-compose up command reads the docker-compose.yml file and starts all the services defined in it.
- The -d flag runs the containers in detached mode, meaning they will run in the background.

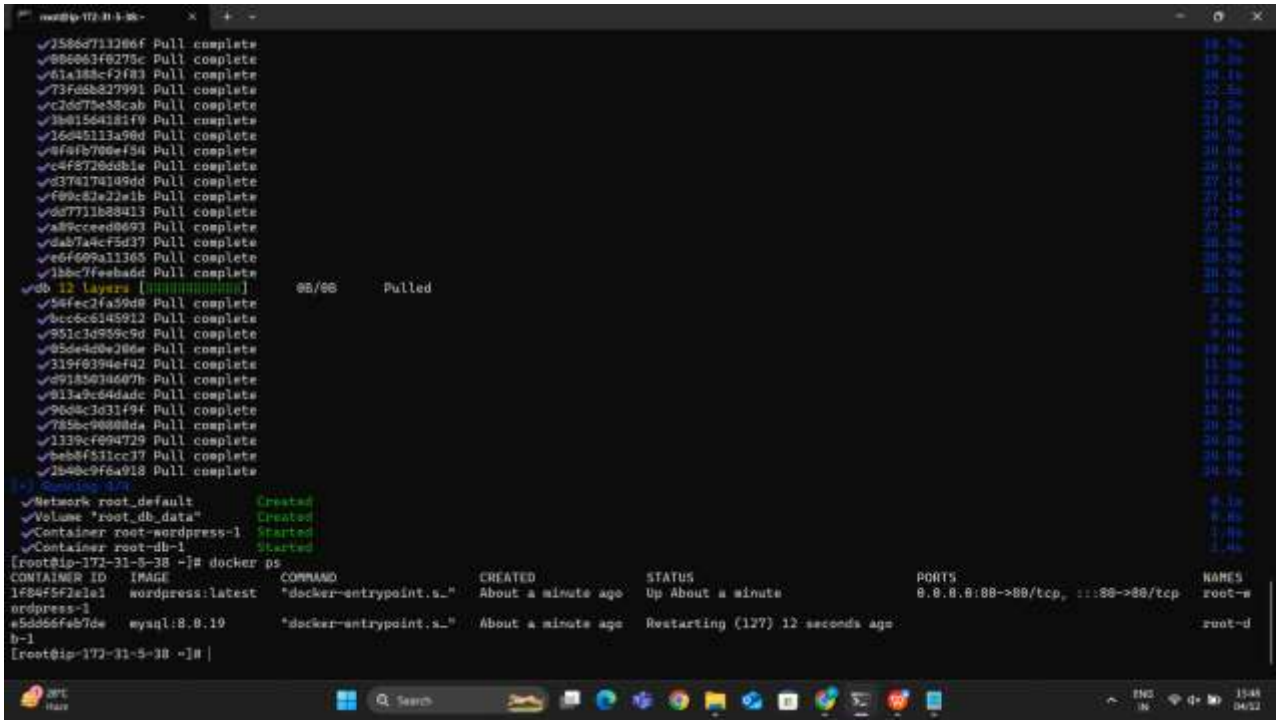


```
parsing /root/docker-compose.yml: yaml: line 2: mapping values are not allowed in this context
[root@ip-172-31-5-38 ~]# vim docker-compose.yml
[root@ip-172-31-5-38 ~]# docker-compose up -d
[+] Running 34/36
 ! wordpress 22 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣀⣿⣿⣿⣿⣿⣿] 6.685MB/17.47MB Pulling
   ✓bc0065b23a04 Pull complete
   ✓e0aa8e8bfd18 Pull complete
   ✓f45eeb7b7c66 Pull complete
   ✓2802fa207e46 Pull complete
   ✓2bc706e6dbbe Pull complete
   ✓8f2b85a95cfd Pull complete
   ✓2586d713206f Pull complete
   ✓086063f0275c Pull complete
   ✓61a388cf2f83 Pull complete
   ✓73fd6b827991 Pull complete
   ✓c2dd75e58cab Pull complete
   ✓3b01564181f9 Pull complete
   ✓16d45113a90d Pull complete
   ✓4f4fb700ef54 Pull complete
   ✓c4f8720ddb1e Pull complete
   ⠿ d37417414 9dd Extracting      [====================>                              ]  6.685MB/17.47MB
   ✓f09c82e22e1b Download complete
   ✓dd7711b88413 Download complete
   ✓a89cceed8693 Download complete
   ✓dab7a4cf5d37 Download complete
   ✓e6f609a11365 Download complete
   ✓1bbc7feeba6d Download complete
 ✓db 12 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿]      0B/0B      Pulled
   ✓54fec2fa59d8 Pull complete
   ✓bcc6c6145912 Pull complete
   ✓951c3d959c9d Pull complete
   ✓05de4d0e206e Pull complete
   ✓319f0394ef42 Pull complete
   ✓d9185034607b Pull complete
   ✓013a9c64dadc Pull complete
   ✓96d4c3d31f9f Pull complete
   ✓785bc90808da Pull complete
   ✓1339cf094729 Pull complete
   ✓beb8f531cc37 Pull complete
   ✓2b40c9f6a918 Pull complete
```

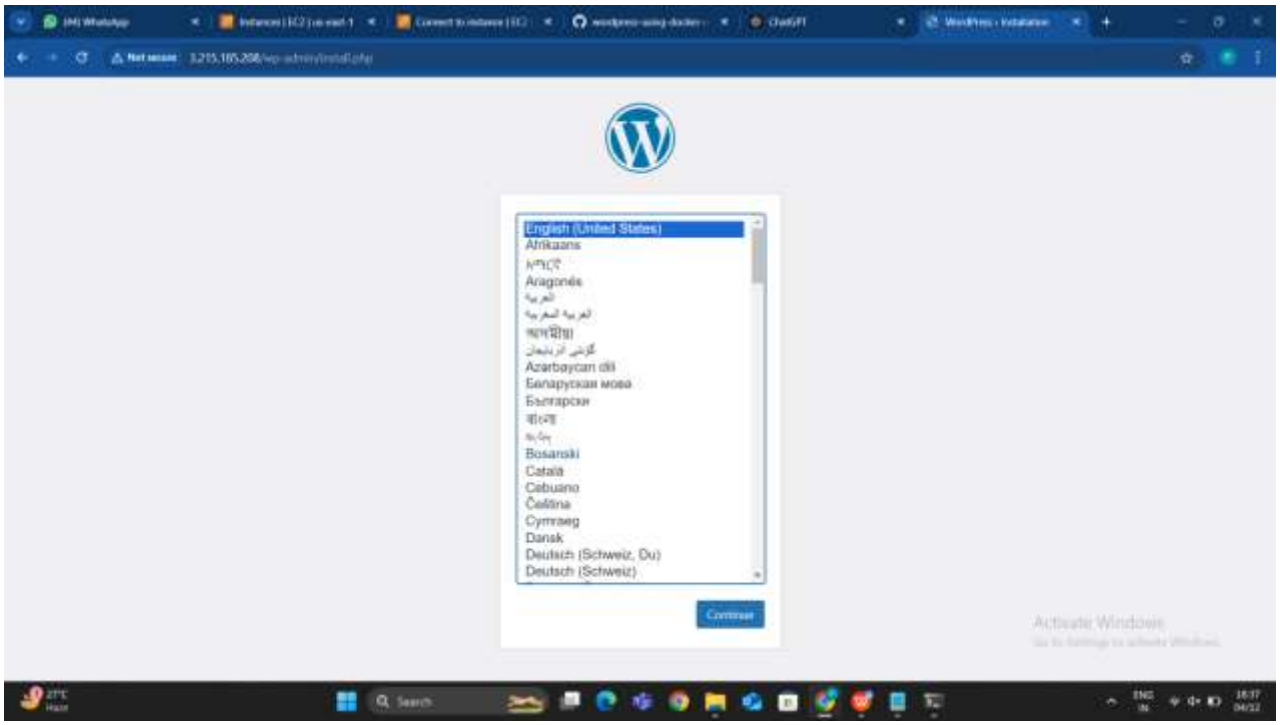


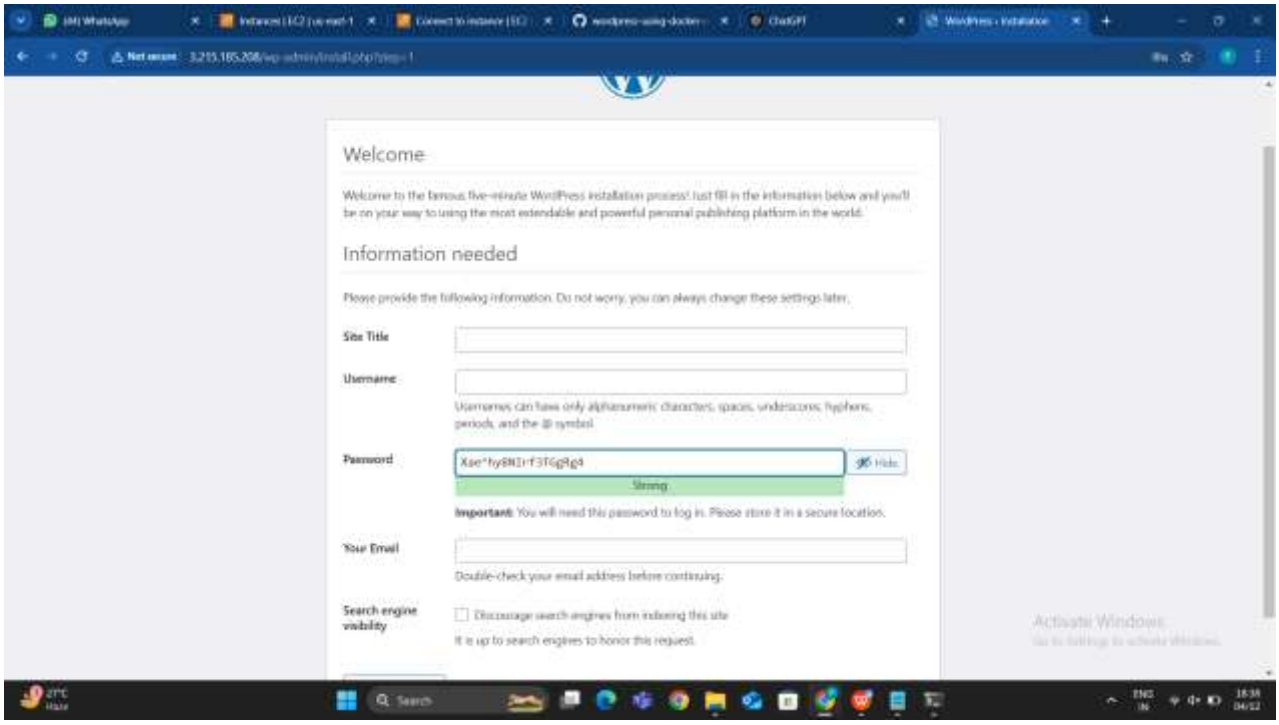
```
   ✓bc0065b23a04 Pull complete
   ✓e0aa8e8bfd18 Pull complete
   ✓f45eeb7b7c66 Pull complete
   ✓2802fa207e46 Pull complete
   ✓2bc706e6dbbe Pull complete
   ✓8f2b85a95cfd Pull complete
   ✓2586d713206f Pull complete
   ✓086063f0275c Pull complete
   ✓61a388cf2f83 Pull complete
   ✓73fd6b827991 Pull complete
   ✓c2dd75e58cab Pull complete
   ✓3b01564181f9 Pull complete
   ✓16d45113a90d Pull complete
   ✓4f4fb700ef54 Pull complete
   ✓c4f8720ddb1e Pull complete
   ✓d37417414 9dd Pull complete
   ✓f09c82e22e1b Pull complete
   ✓dd7711b88413 Pull complete
   ✓a89cceed8693 Pull complete
   ✓dab7a4cf5d37 Pull complete
   ✓e6f609a11365 Pull complete
   ✓1bbc7feeba6d Pull complete
 ✓db 12 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿]      0B/0B      Pulled
   ✓54fec2fa59d8 Pull complete
   ✓bcc6c6145912 Pull complete
   ✓951c3d959c9d Pull complete
   ✓05de4d0e206e Pull complete
   ✓319f0394ef42 Pull complete
   ✓d9185034607b Pull complete
   ✓013a9c64dadc Pull complete
   ✓96d4c3d31f9f Pull complete
   ✓785bc90808da Pull complete
   ✓1339cf094729 Pull complete
   ✓beb8f531cc37 Pull complete
   ✓2b40c9f6a918 Pull complete
[+] Running 4/4
 ✓Network root_default          Created
 ✓Volume "root_db_data"         Created
 ✓Container root-wordpress-1    Started
 ✓Container root-db-1           Started
[root@ip-172-31-5-38 ~]# |
```

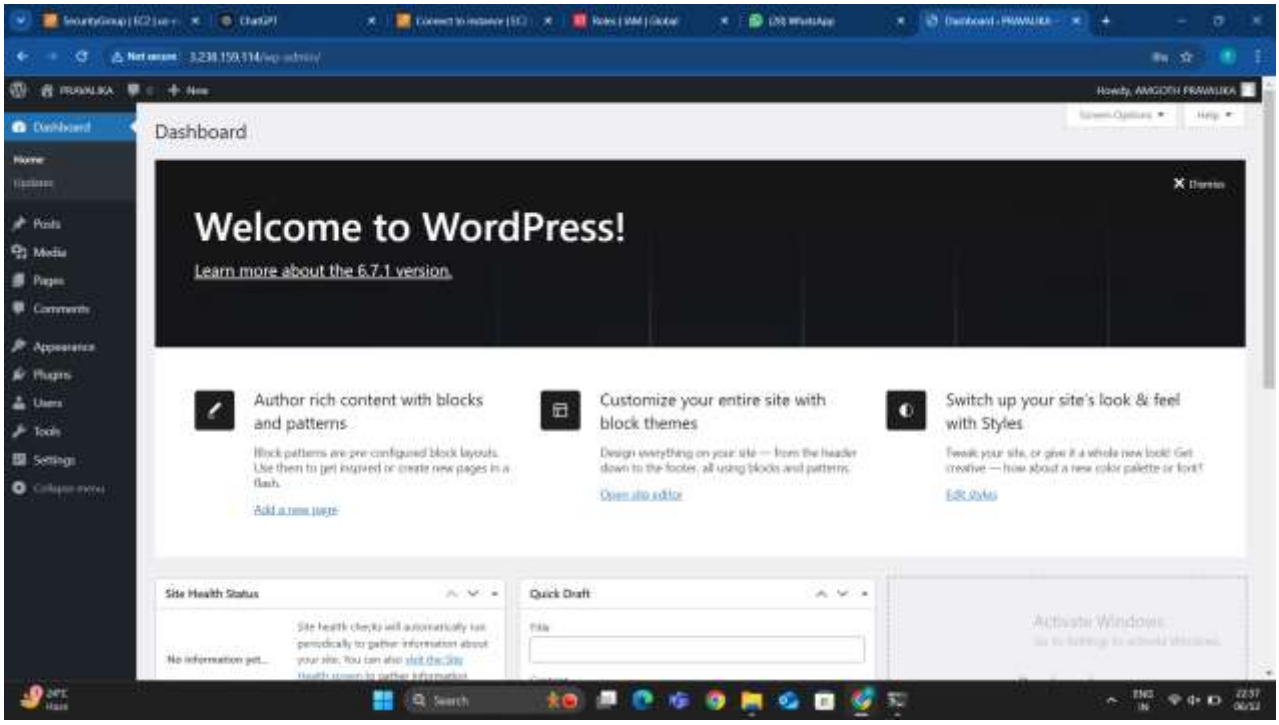
Check for containers using docker ps command



After creation of the containers now access the instance public instance in the browser and u will get the out put







METHOD-3:

Deploy WordPress web application by using git and jenkins

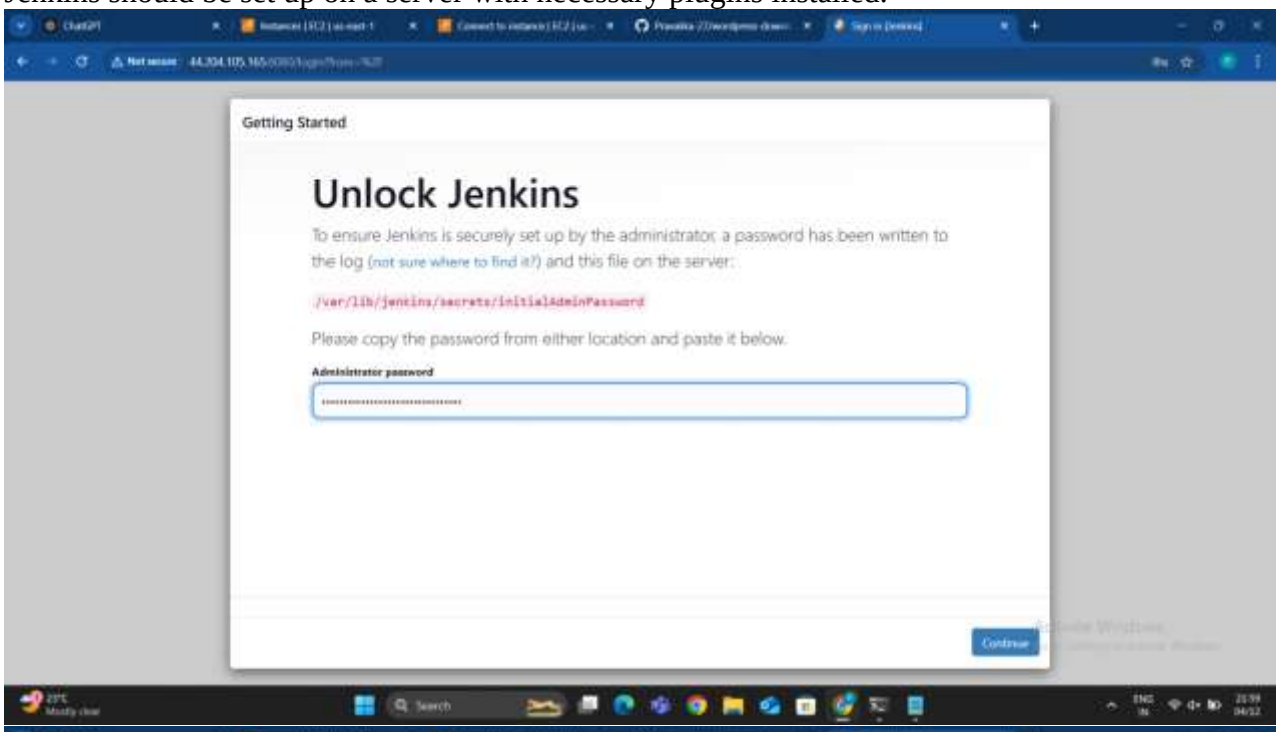
Prerequisites Setup:

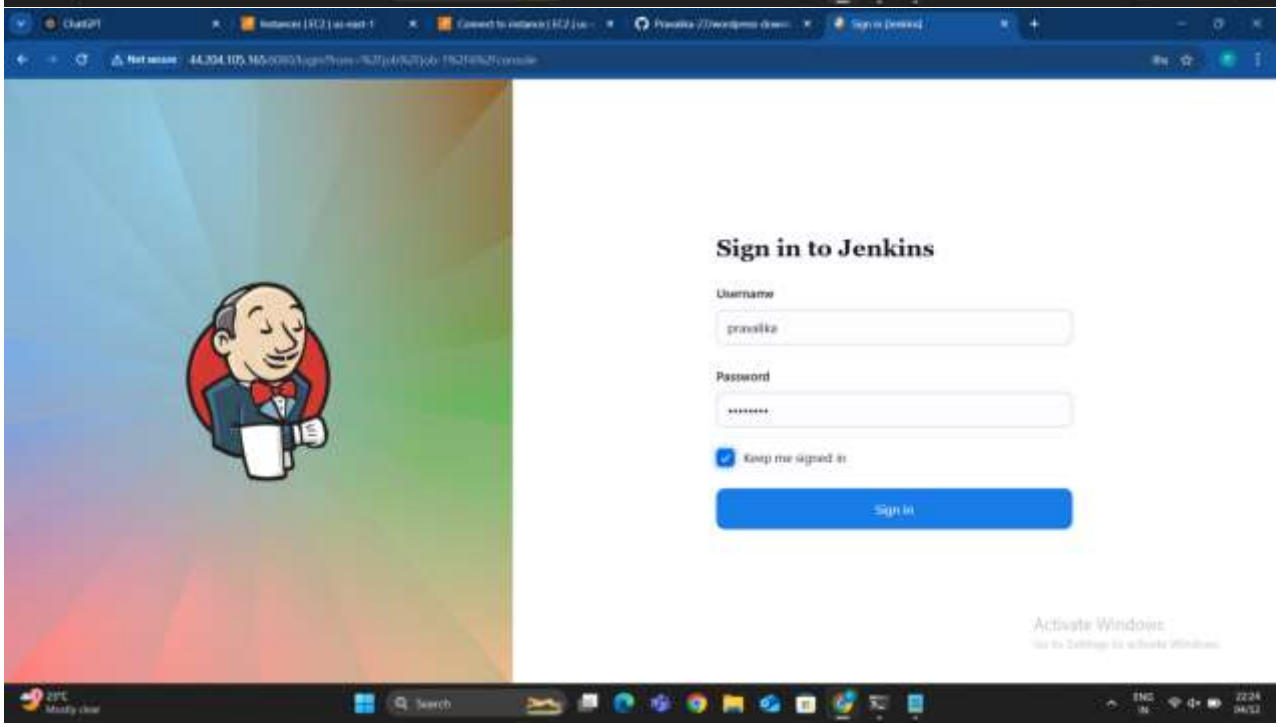
1. Git Repository:

Use platforms like GitHub, GitLab, or Bitbucket to host your WordPress files.

Jenkins Installation:

Jenkins should be set up on a server with necessary plugins installed.





Git Plugin: For pulling the WordPress code from the repository.

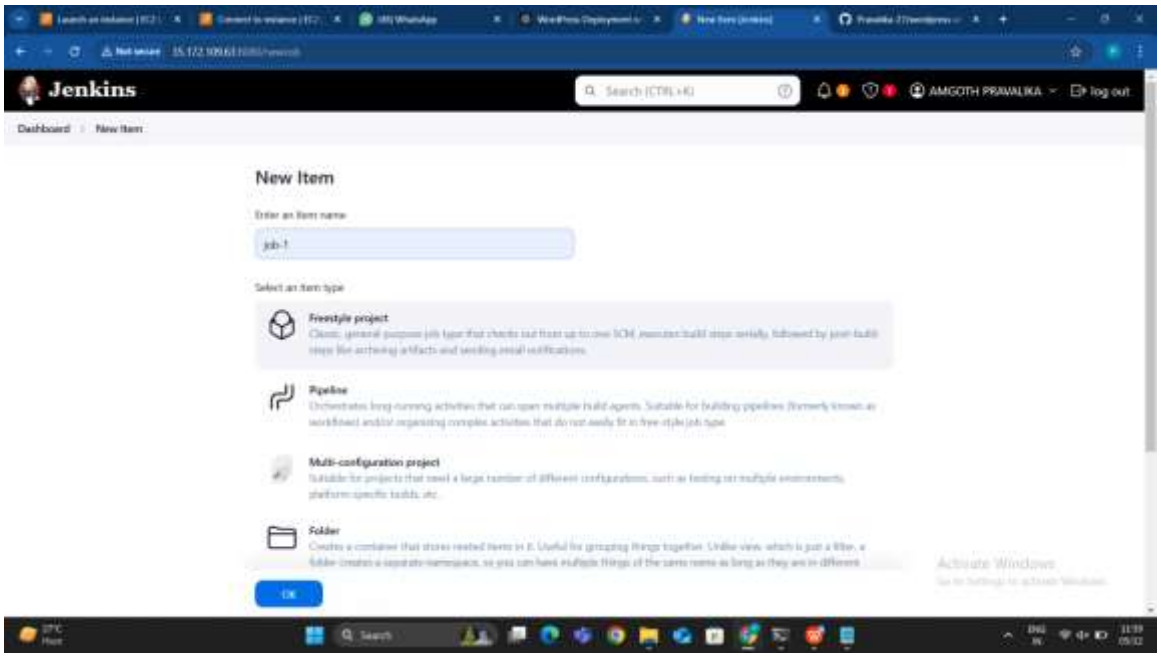
Publish Over SSH (optional): To copy files to the target server

Web Server: A server with PHP and MySQL configured to host WordPress.
Database Setup: Ensure the MySQL database for WordPress is pre-configured.
Access Credentials: SSH access to the web server.

Step 2: Configure Jenkins

Create a New Job:

1. In Jenkins, click on **"New Item"**.
2. Select **"Pipeline"** or **"Freestyle Project"**, and give it a name.



Set Up Source Code Management:

1. Under **"Source Code Management"**, select **Git**.
2. Provide your repository URL and credentials.

Create a Jenkins Job

Add a New Job:

1. Go to the Jenkins dashboard and click **New Item**.
2. Choose **Freestyle Project**, name it (e.g., WordPress_Deployment), and click **OK**.

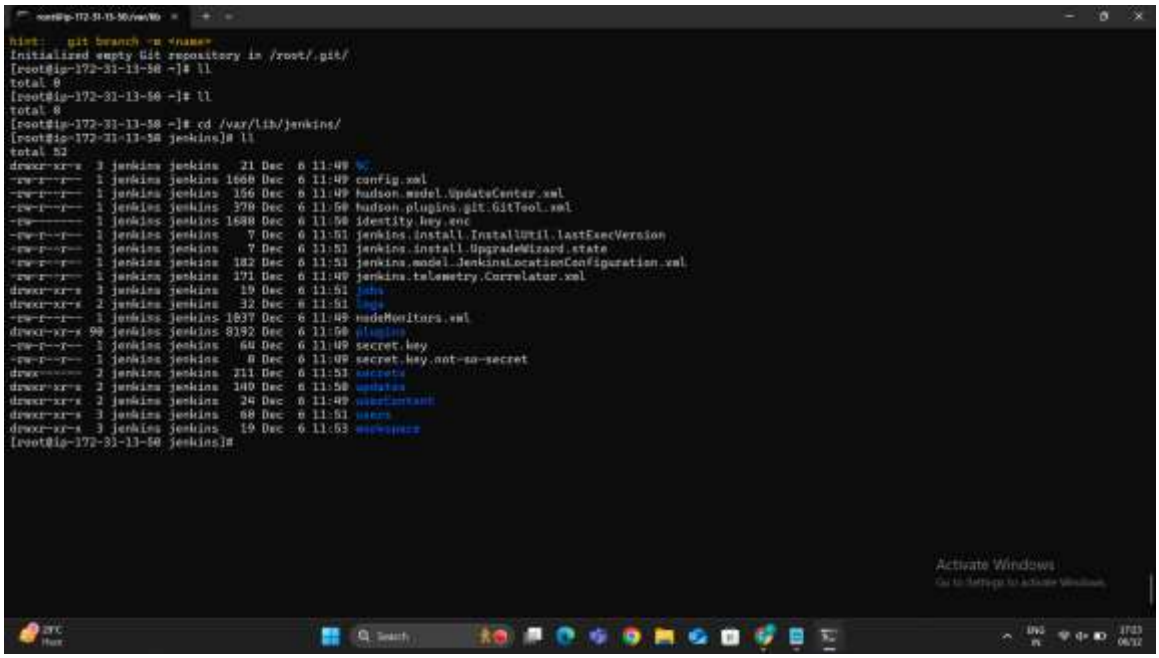
Configure Git Repository:

1. Under **Source Code Management**, select **Git**.

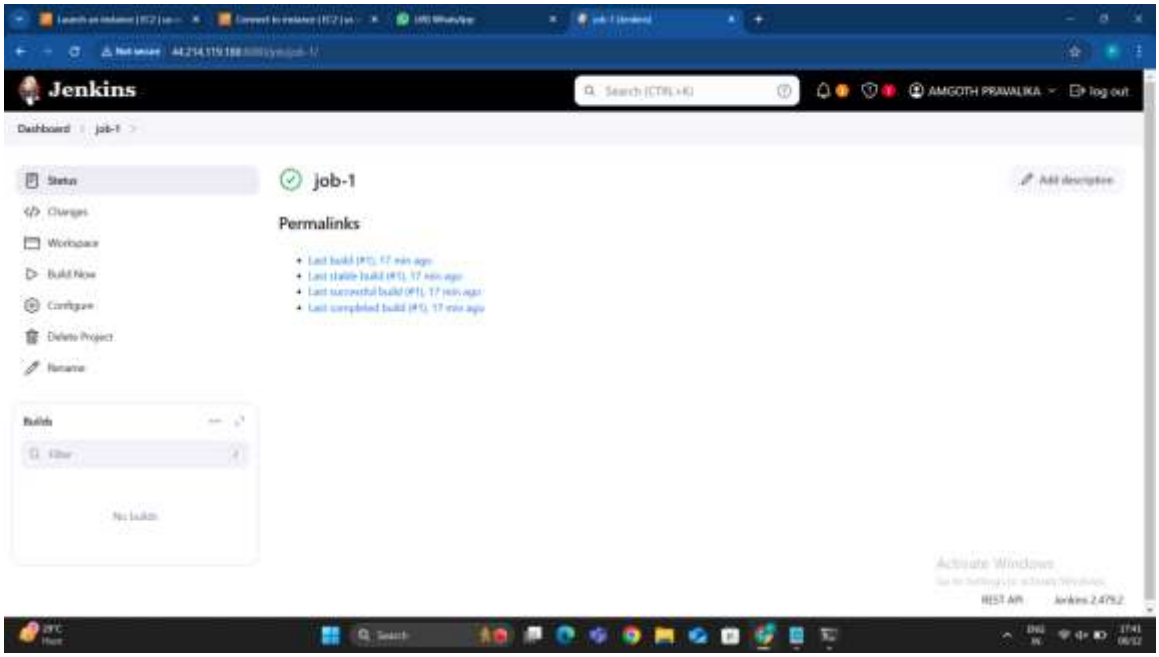
2. Enter your Git repository URL and credentials if required

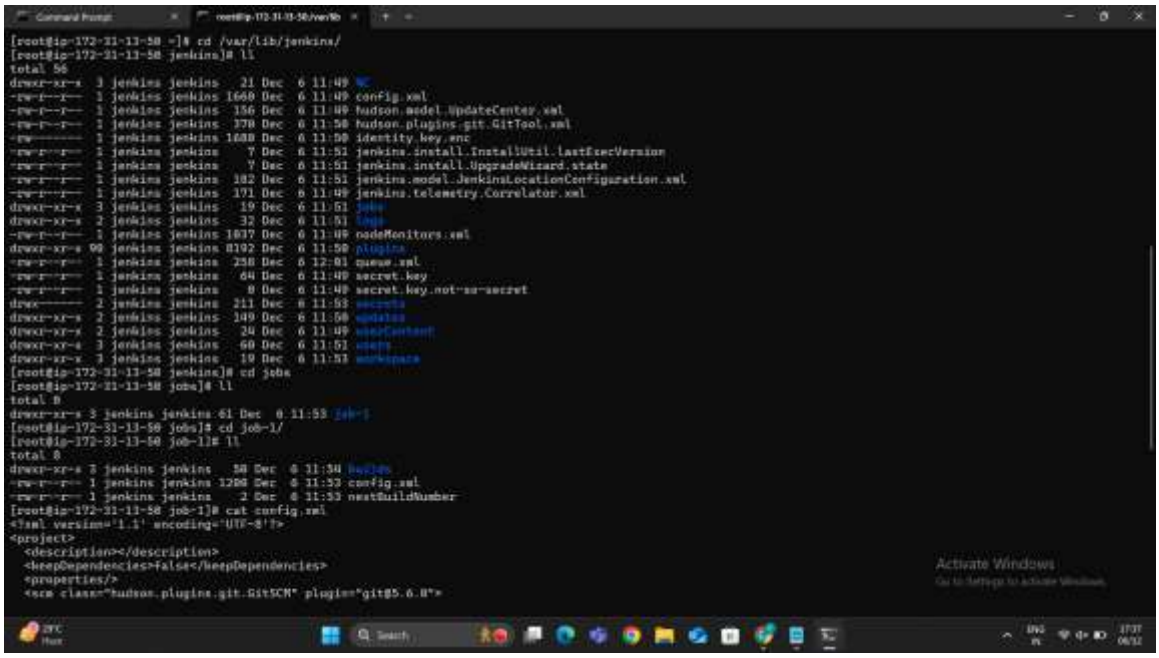
Install Plugins (if not installed):

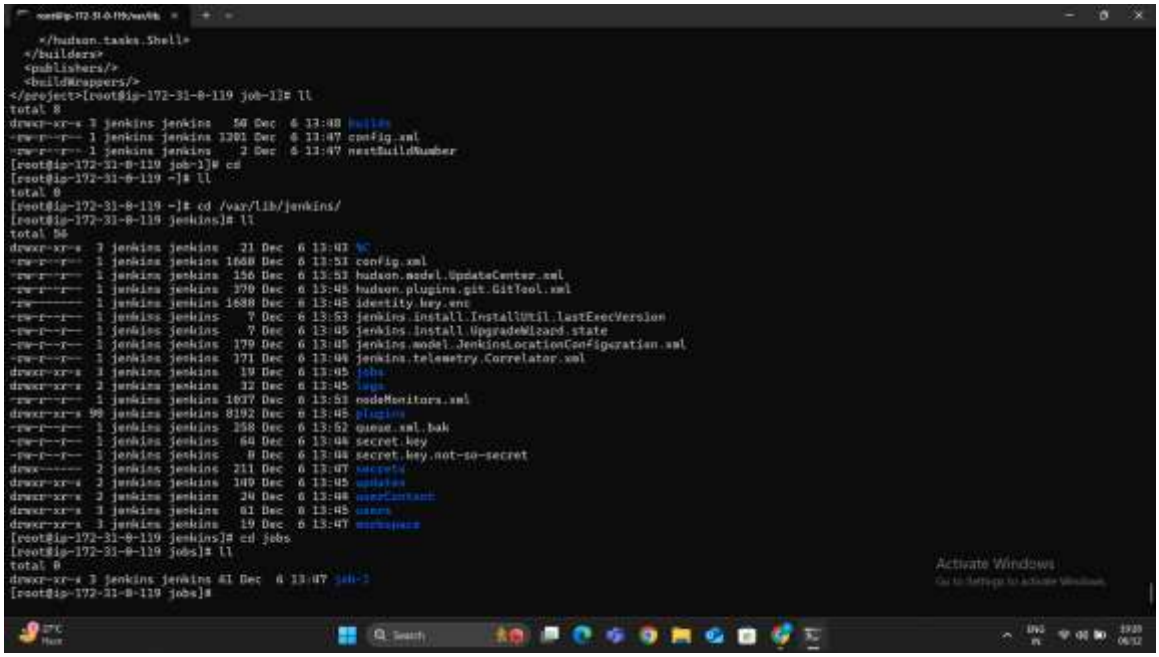
1. Navigate to **Manage Jenkins → Plugin Manager → Available Plugins**.
2. Install:
    1. **Git Plugin**
    2. **SSH Pipeline Steps** or **Publish Over SSH Plug**



-build the job



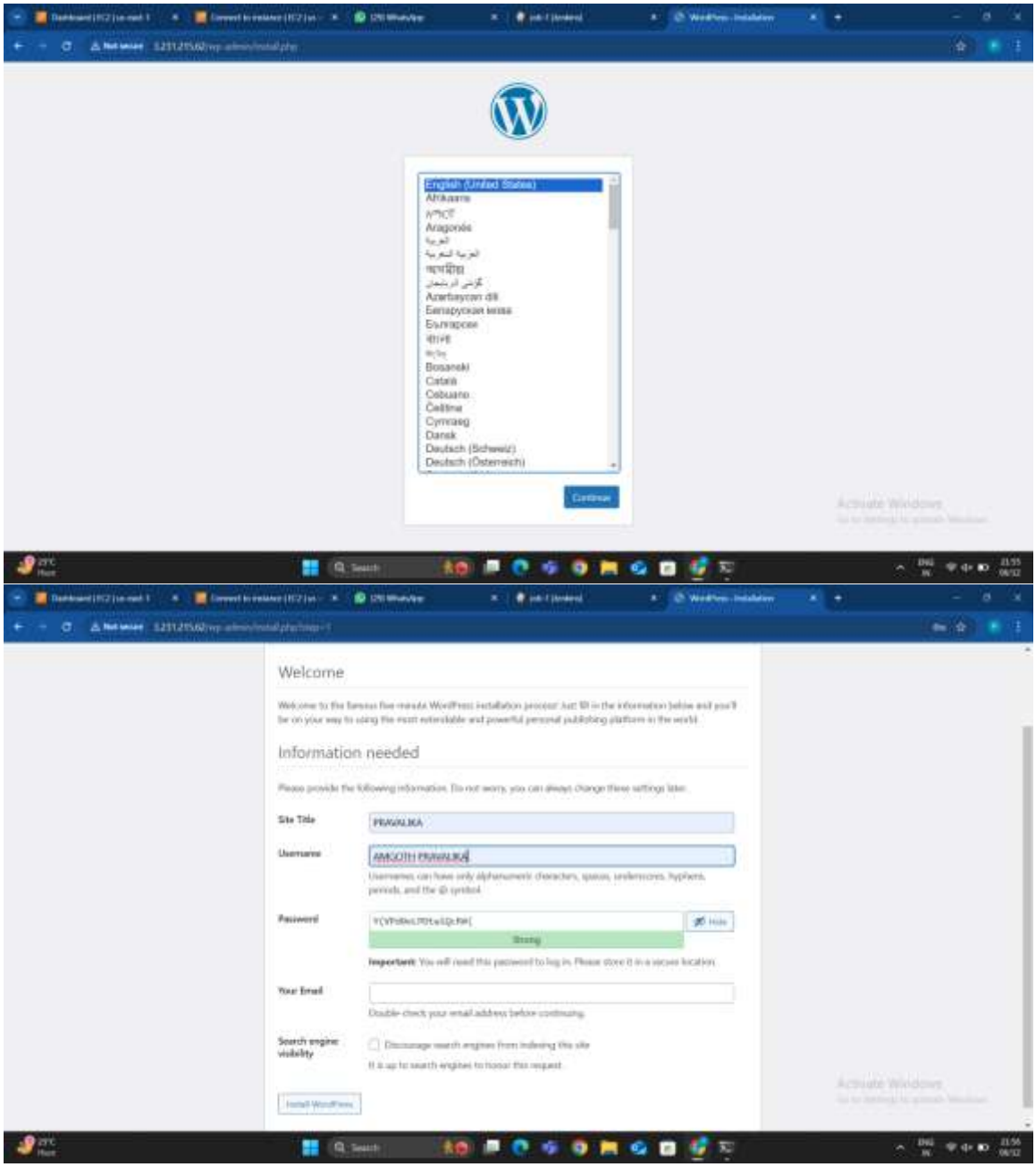


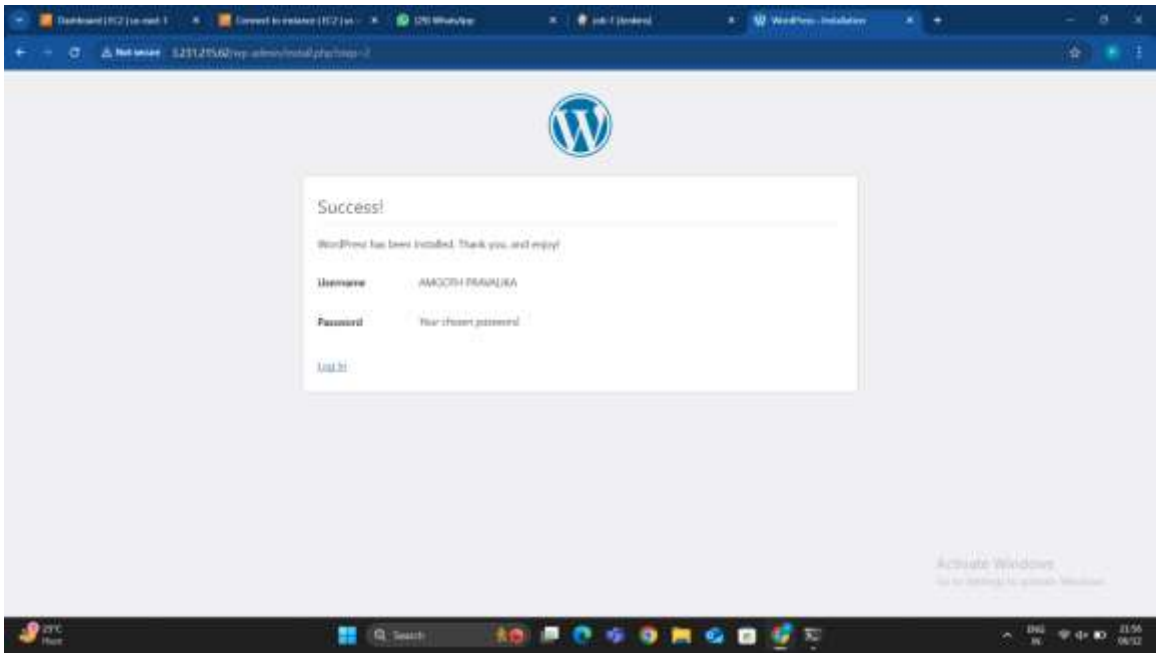


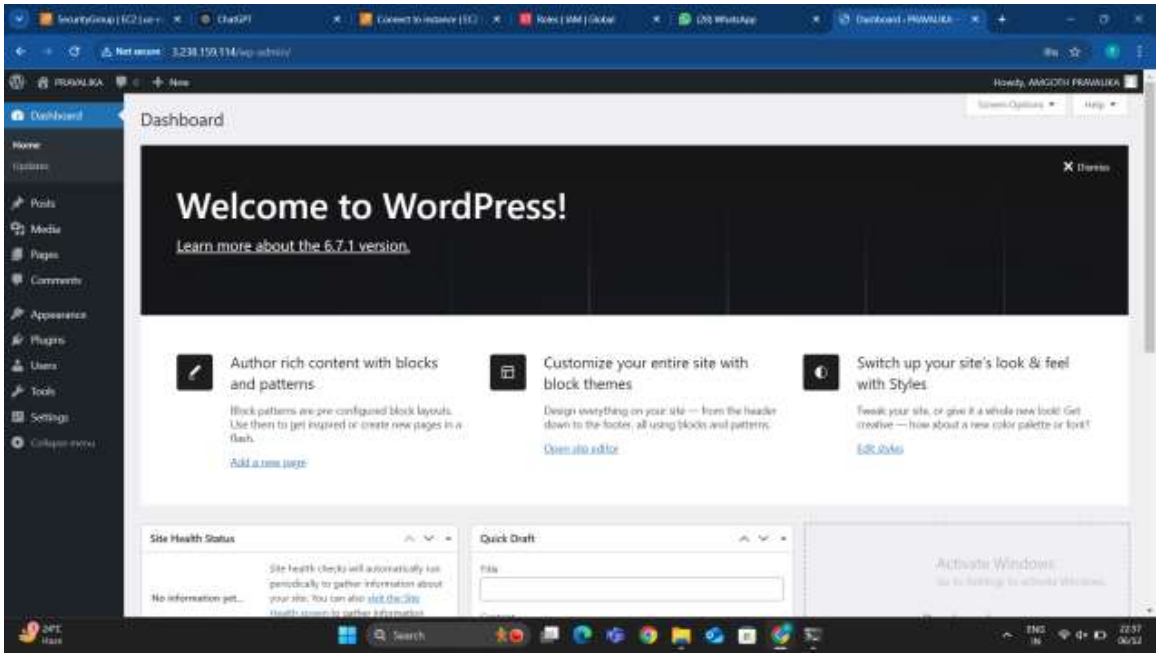
Summary Description

1. **Install Jenkins:** Set up Jenkins on your server and install required plugins.
2. **Prepare Git Repository:** Push your WordPress application files to a Git repository.
3. **Create a Jenkins Job:** Add the Git URL, configure the build step with deployment commands, and save the job.
4. **Run the Job:** Trigger the job to pull the latest code and deploy the WordPress application using Docker Compose.

Now access the public ip of the instance in the browser you will get the output.







METHOD-4:

Deploy WordPress web application by using userdata of EC2 instance?

Deploying a WordPress web application using **User Data** in an **AWS EC2 instance** involves automating the server setup during instance initialization. This allows the instance to be fully configured and ready to host WordPress without manual intervention.

Step 1: Prerequisites

1. **AWS Account**: Ensure you have access to an AWS account.
2. **IAM Role**: Create an IAM role with appropriate permissions (e.g., AmazonS3FullAccess, AmazonEC2FullAccess).
3. **Key Pair**: Create a key pair to connect to the instance later, if needed.
4. **Security Group**: Configure a security group with:
    1. **Inbound Rules**:
        1. Allow HTTP (Port 80) and HTTPS (Port 443).
        2. Allow SSH (Port 22) for troubleshooting (optional).
5. **MySQL Database**: Prepare an RDS instance or install MySQL on the same EC2 instance.

Step 1: Launch an EC2 Instance

Log in to AWS Management Console:

1. Navigate to the **EC2 Dashboard**.

Create a New EC2 Instance:

1. Click on **Launch Instance**.
2. Provide a name for the instance (e.g., WordPressServer).

Choose an Amazon Machine Image (AMI):

1. Select a Linux-based AMI, such as **Amazon Linux 2** (free tier eligible).

Select an Instance Type:

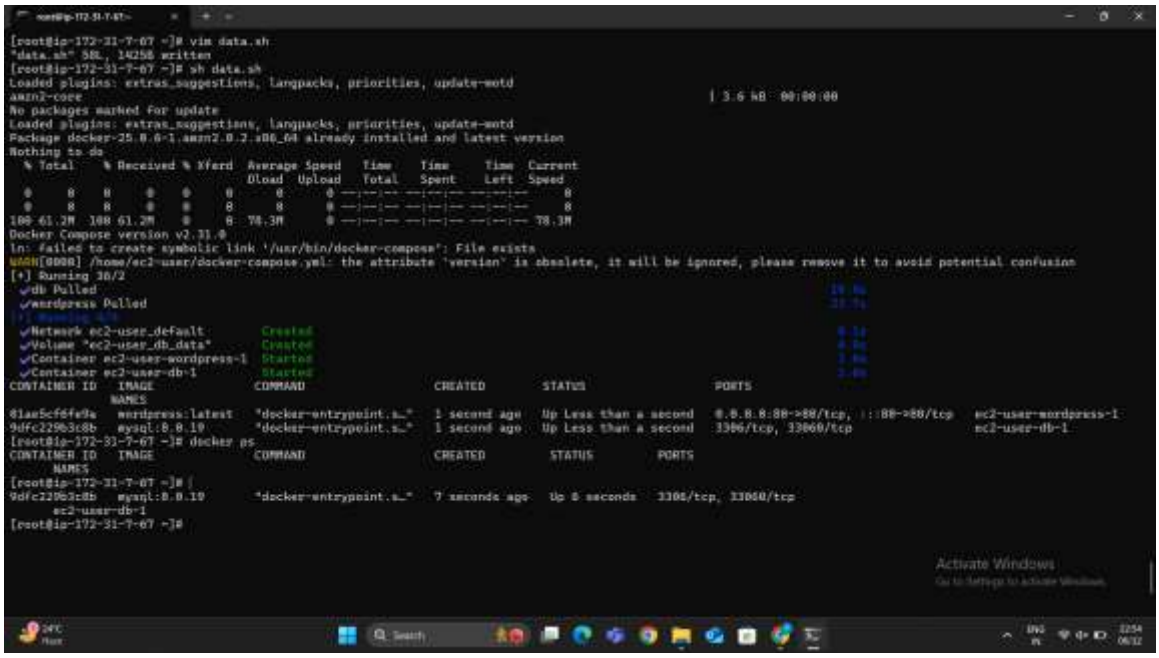
1. Choose **t2.micro** (free tier eligible) or another type based on your needs.

Configure Key Pair:

1. Select an existing key pair or create a new one for SSH access.

Set Up Security Group:

1. Open the following ports:
    1. **Port 22 (SSH)** for remote access.
    2. **Port 80 (HTTP)** for the web application.

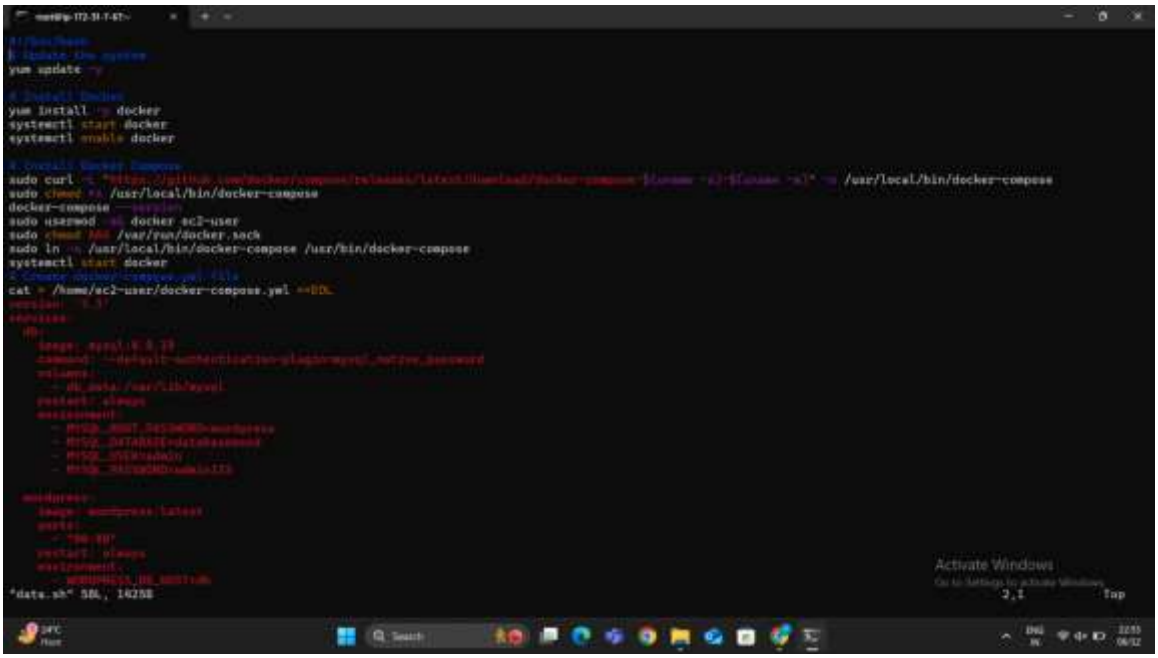


Step 2: Add User Data Script

- In the **Advanced Details** section, add the following user data script to automate the WordPress and LAMP stack installation:

The User Data script will:

- Install required software (Apache, PHP, MySQL client, etc.).
- Download and configure WordPress.
- Set permissions and restart the server.



Step 3: Launch EC2 Instance

1. **Login to AWS Console**:
    1. Go to **EC2 Dashboard** and click **Launch Instances**.
2. **Configure Instance Details**:

    1. Select an appropriate **AMI** (e.g., Amazon Linux 2).
    2. Choose an instance type (e.g., t2.micro for free tier).
3. **Add User Data**:

    1. Under the **"Advanced Details"** section, paste the User Data script.
4. **Add Storage**:

    1. Allocate enough storage (e.g., 20GB) for the WordPress files.
5. **Assign Security Group**:

    1. Use the security group with HTTP, HTTPS, and SSH allowed.
6. **Review and Launch**:

    1. Assign an IAM role and launch the instance.

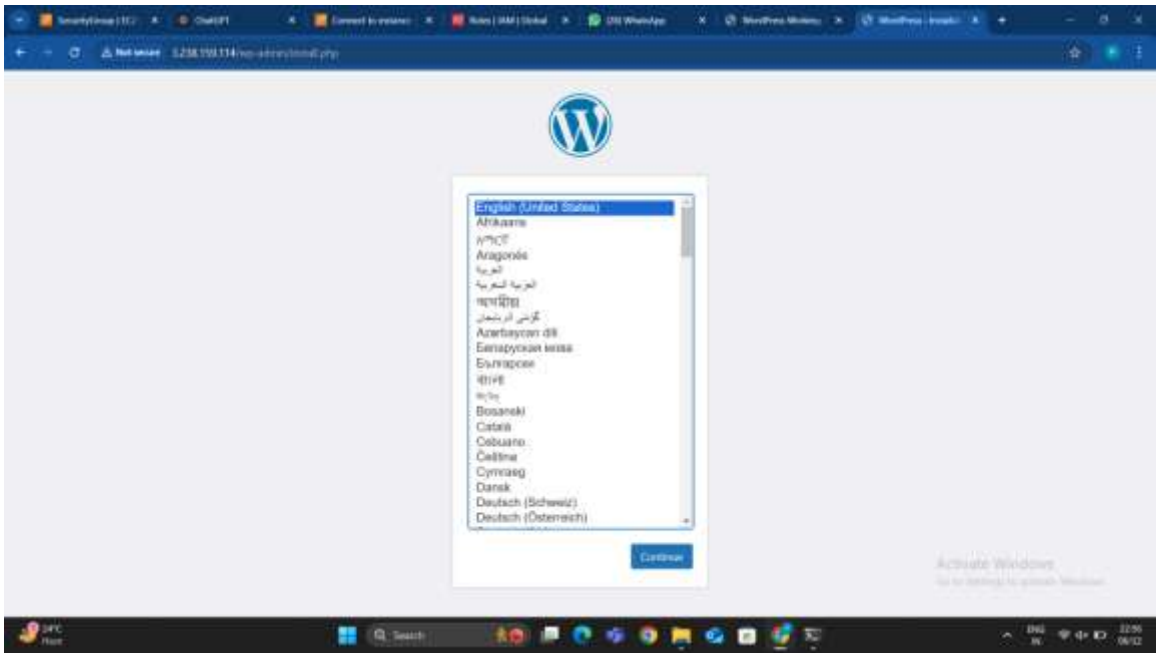
Step 4: Access the WordPress Application

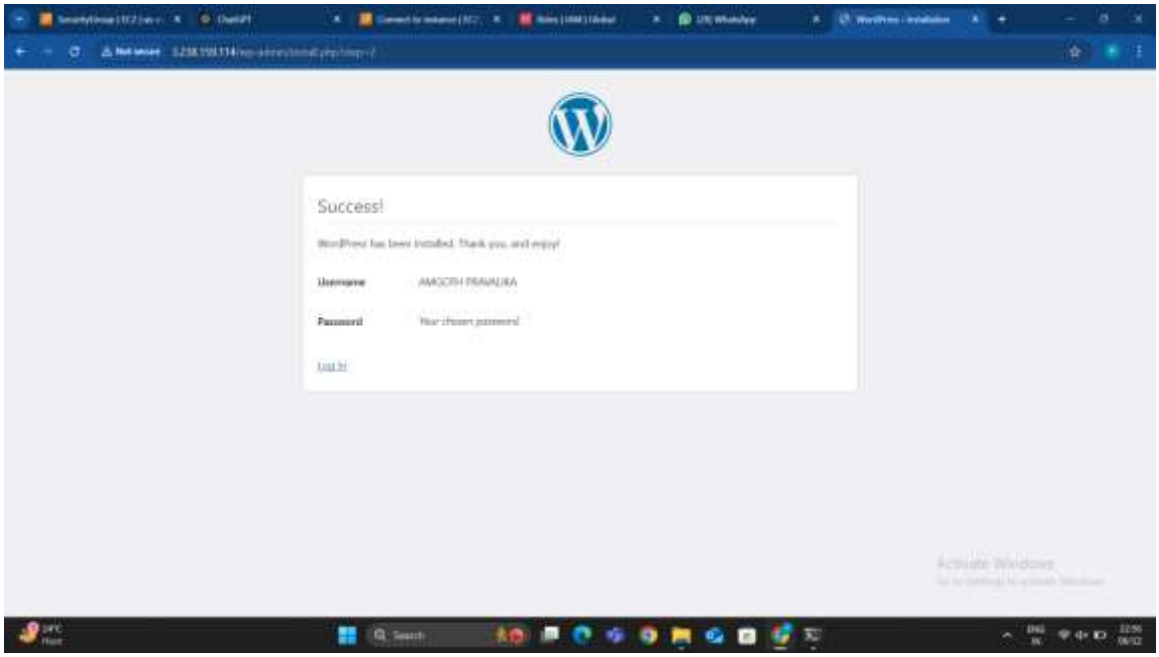
1. Wait for the EC2 instance to initialize.
2. Obtain the **Public IP** of the instance from the EC2 dashboard.
3. Open a browser and navigate to http://<Public-IP>:

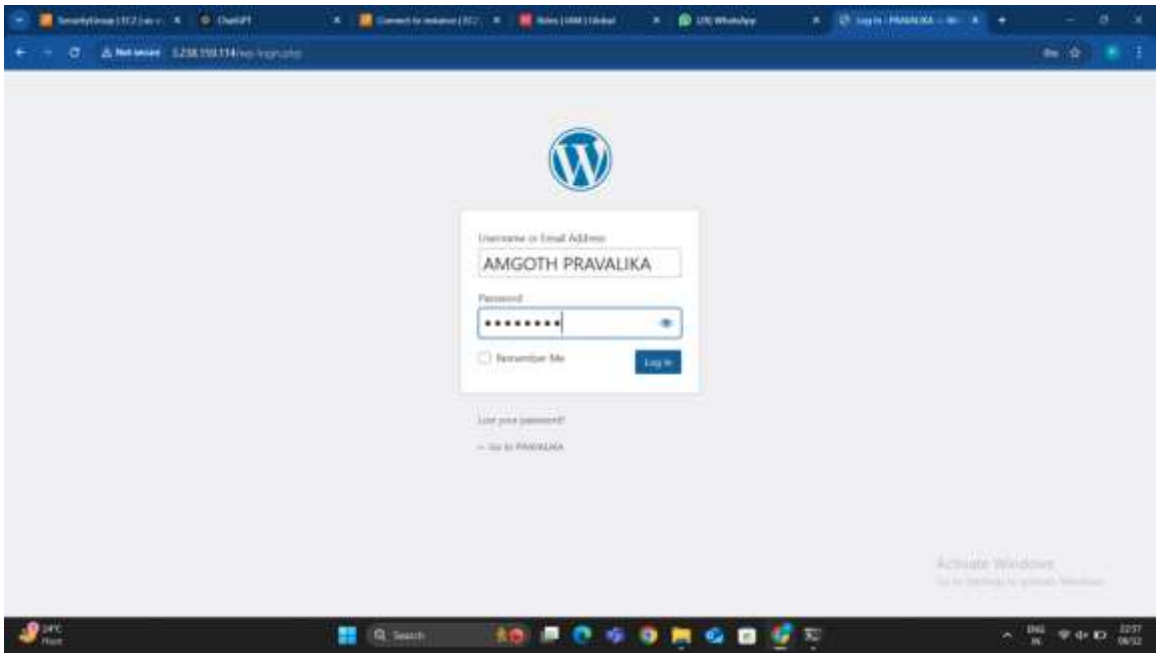
    1. You should see the WordPress setup wizard.

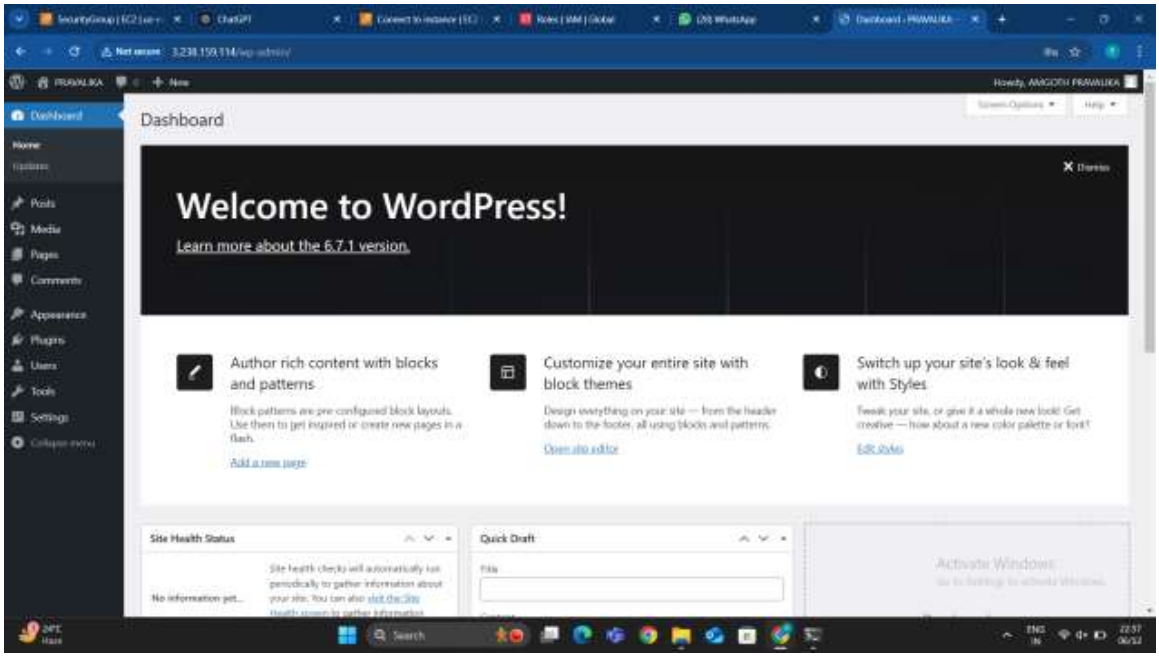
Summary

1. **User Data Script:** Automates the installation of Apache, MySQL, PHP, and WordPress.
2. **Database Configuration:** Creates a wordpress database and a user (wp_user) with necessary privileges.
3. **Automatic Deployment:** The wp-config.php file is configured with database details, enabling a ready-to-use WordPress site.
4. **Access:** The WordPress site is accessible through the EC2 instance's public IP.


English (United States)
Afrikaans
Aragonés
Azərbaycan dili
Беларуская мова
Bosanski
Català
Cebuano
Čeština
Cymraeg
Dansk
Deutsch (Schweiz)
Deutsch (Österreich)
Continue


Success!
Username
Password

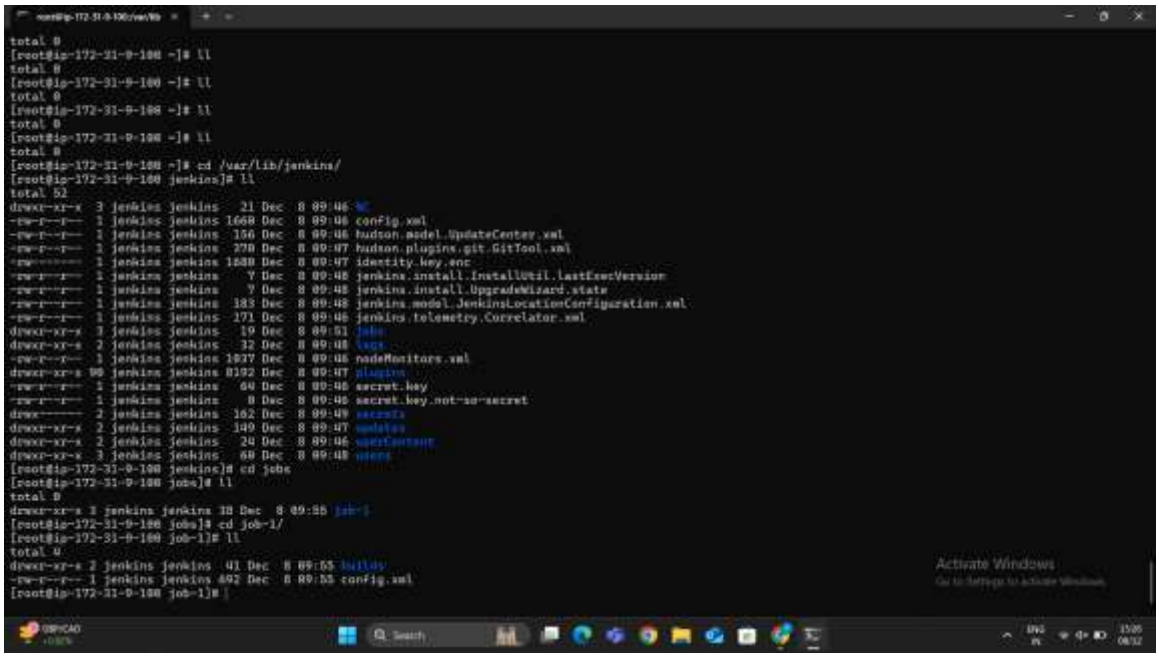




METHOD-5:

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

To deploy a **WordPress web application** using **Git** and **Jenkins Execute Shell (Bash Script)**, follow these steps:



Step 1: Install and Configure Jenkins

Install Jenkins:

1. Ensure Jenkins is installed on your system. Follow the steps in the Jenkins installation guide if not already done.

Install Git Plugin:

1. Go to **Manage Jenkins → Manage Plugins → Available Plugins**, search for the **Git Plugin**, and install it.

Step 2: Prepare Your Git Repository

Push WordPress Files to Git:

Ensure the repository contains the following:

1. docker-compose.yml file for WordPress and MySQL setup.
2. Any necessary configuration files, like wp-config.php (optional if it's dynamically generated).

Step 3: Create a Jenkins Job

Go to Jenkins Dashboard:

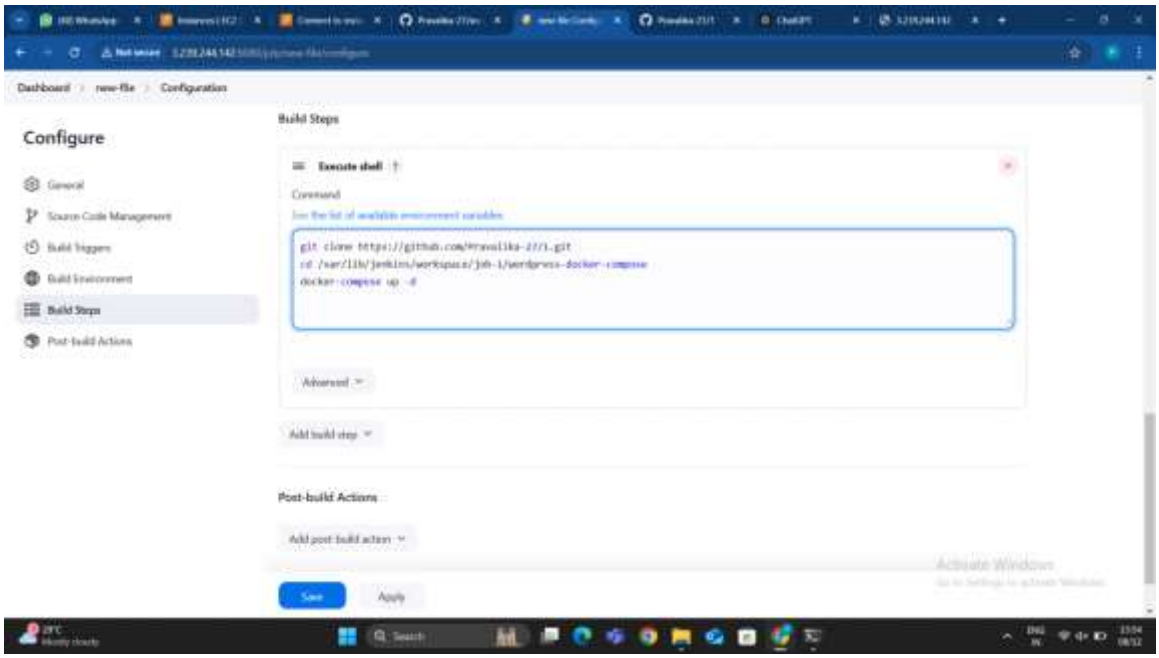
1. Click **New Item**, name the job (e.g., Deploy_WordPress), and select **Freestyle Project**.

Configure Source Code Management:

1. Select **Git**.
2. Add the repository URL (e.g., https://github.com/username/my-wordpress-project.git).
3. Specify the branch to build (e.g., main).

Add Execute Shell (Bash Script):

1. In the **Build** section, add an **Execute Shell** step.
2. Paste the following Bash script into the shell section.



Save and Build

1. Save the job configuration.
2. Click **Build Now** to trigger the deployment.

Step 4: Save and Run the Job

Save the Job:

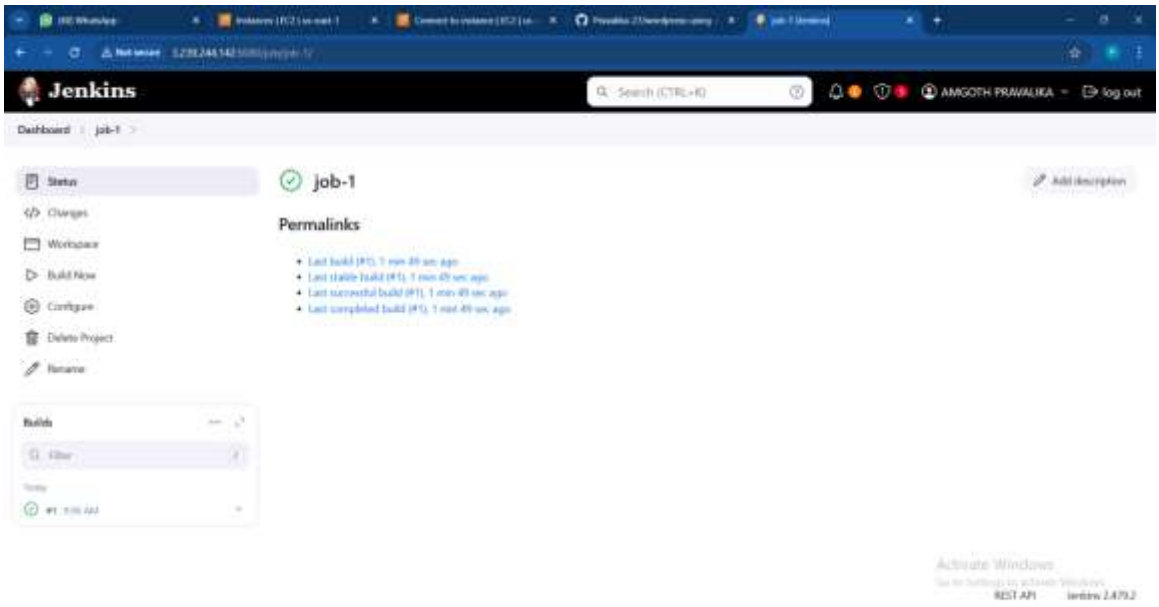
1. Click **Save** to save your Jenkins job configuration.

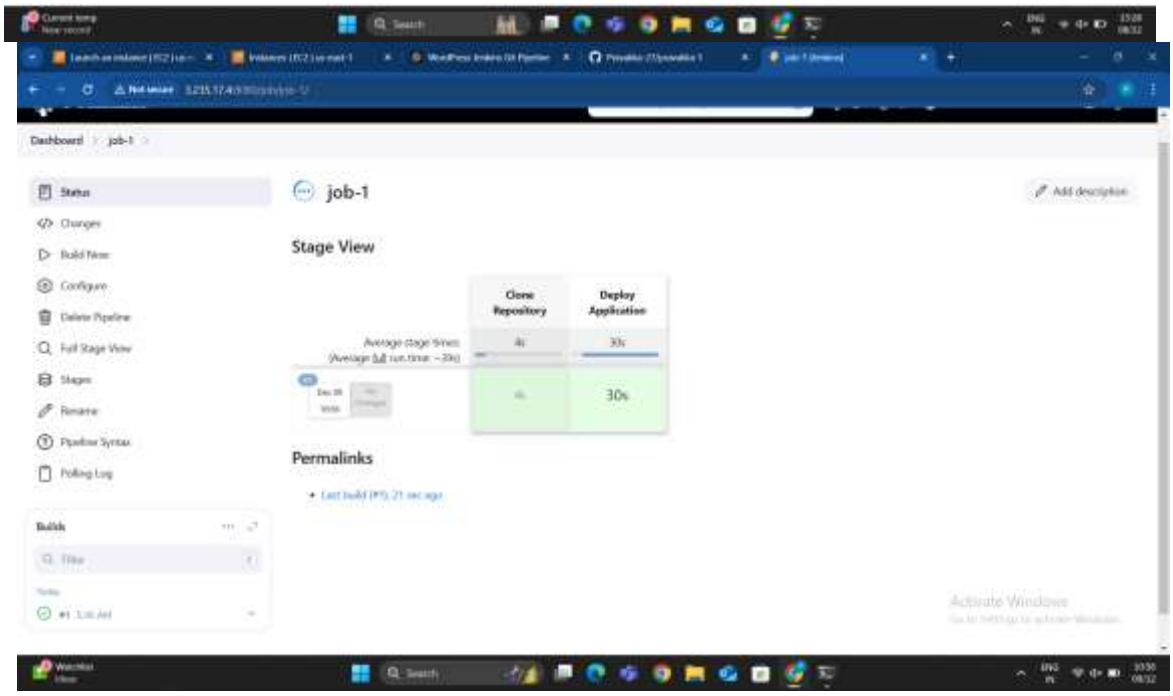
Trigger the Build:

1. Go to the job dashboard and click **Build Now**.

Monitor the Build:

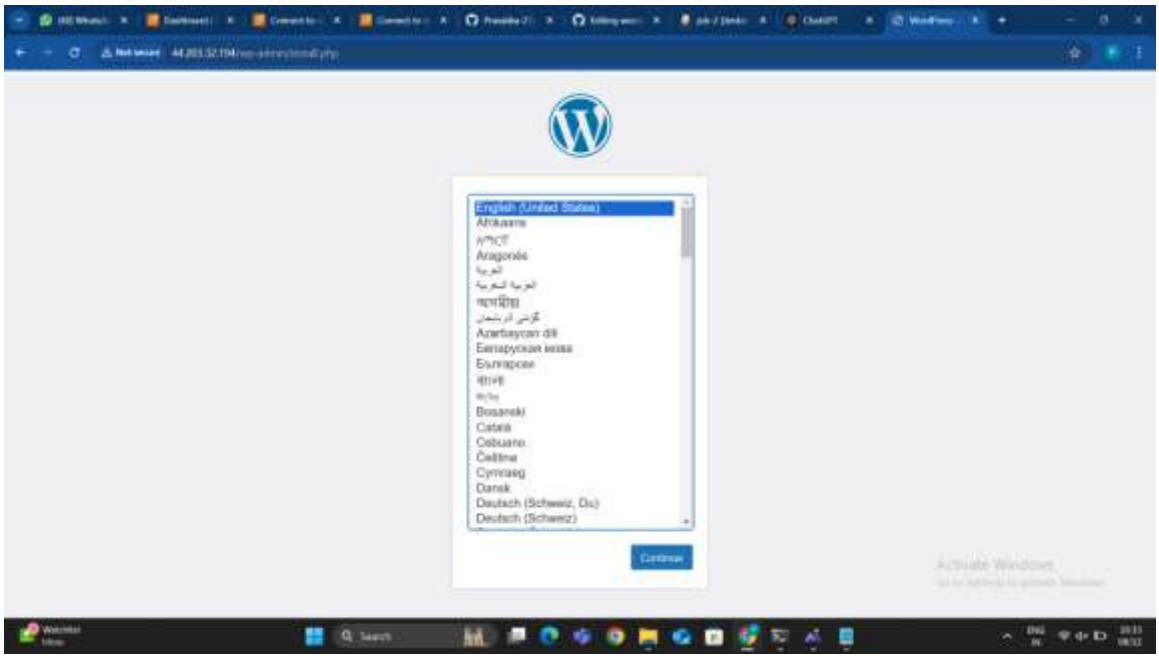
1. Click on the **Build Number → Console Output** to monitor the job's progress.

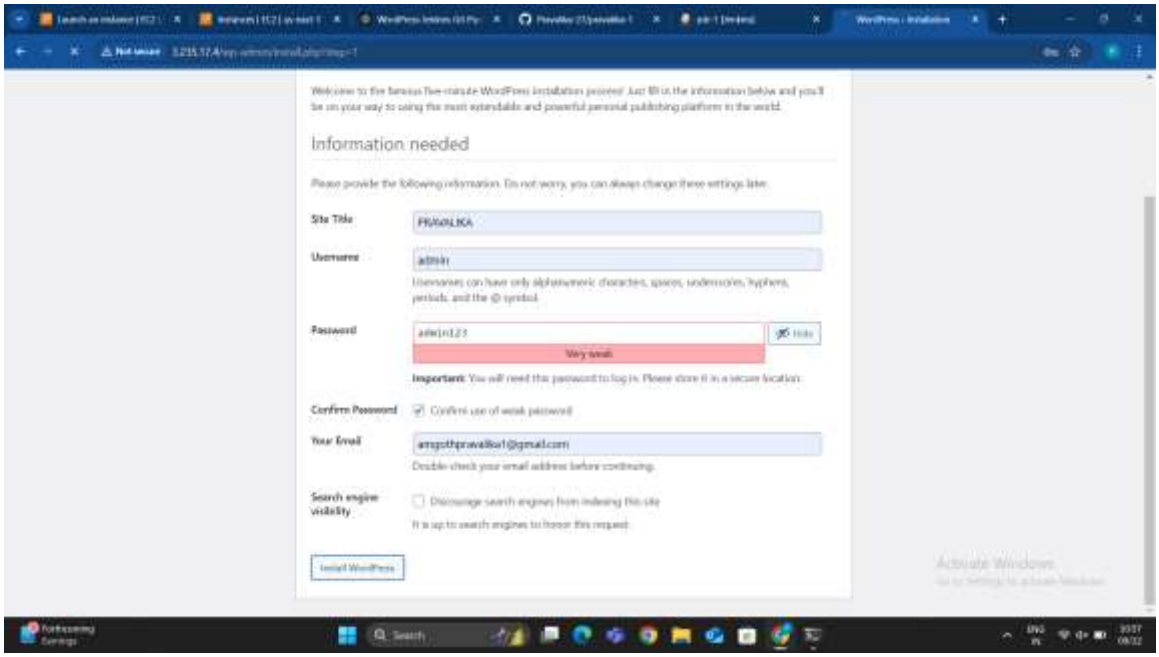


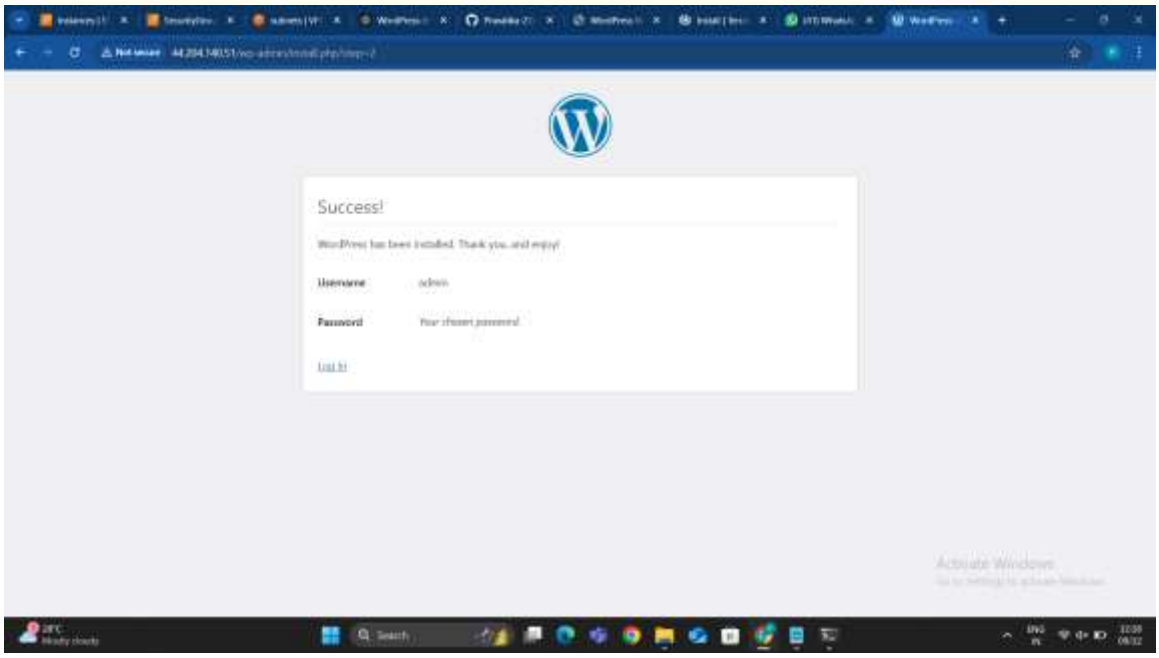


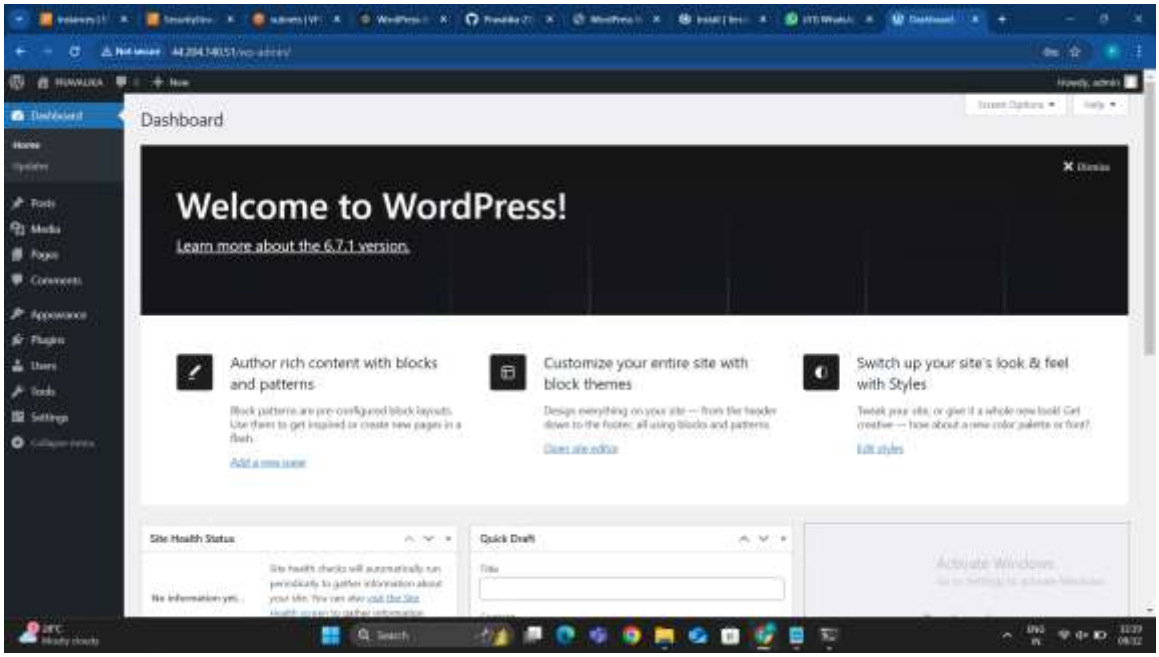
Verify the Deployment

1. Access the application in your browser: http://<server-ip>/wordpress.
2. Complete the WordPress setup wizard.









Explanation of the Bash Script

1. **Pulls Latest Code:** Fetches the latest version of the code from the specified Git branch.
2. **Installs Docker and Docker Compose:** Ensures the required tools are available.
3. **Stops Existing Containers:** Shuts down any previously running WordPress and MySQL containers.
4. **Deploys WordPress:** Uses docker-compose up -d to start the WordPress and MySQL containers.
5. **Verification:** Lists running containers to confirm deployment.

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?

Here's the **step-by-step process** to deploy a WordPress web application using **Git** and **Jenkins Execute Shell (Bash Script)** with a **Jenkins Pipeline** that includes **Build Periodically** and **Poll SCM** to check for GitHub changes:

Step 1: **Set Up Git Repository**

Prepare WordPress Files:

- Organize WordPress files (wp-content, wp-config.php, etc.) in a Git repository.
- Push the files to a remote repository (e.g., GitHub

  Ensure the repository is accessible:

  1. Use SSH keys or a Jenkins credential if your repository is private.

Step 2: Install and Configure Jenkins

Install Required Plugins:

1. Go to **Manage Jenkins → Manage Plugins**.
2. Install:
   1. **Git Plugin** (to pull the repository).
   2. **Pipeline Plugin** (to create pipelines).

Create a Jenkins Credential (if the repository is private):

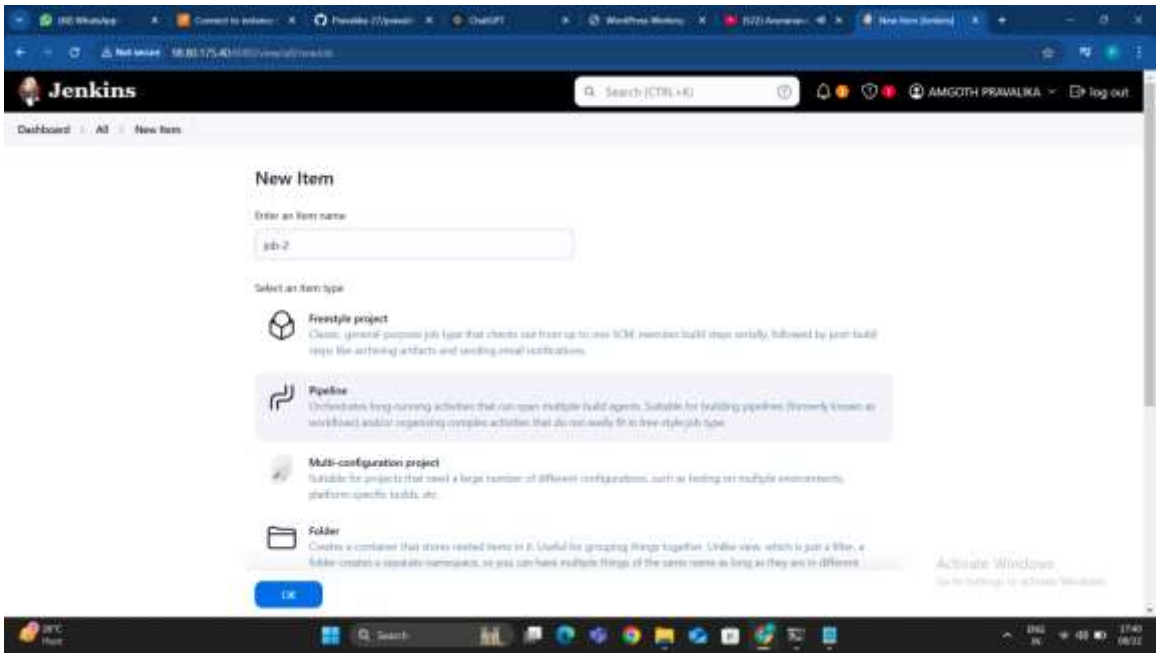
- Navigate to **Manage Jenkins → Manage Credentials**.
- Add your GitHub credentials (username/password or SSH key).

Create a Pipeline Job

1. **Go to Jenkins Dashboard → New Item**.
2. **Enter a Name**, select **Pipeline**, and click **OK**.

Configure the Pipeline

1. **In the Pipeline Configuration Page**:
   1. Go to the **Pipeline** section.



2. Select **Pipeline script** and paste the following:

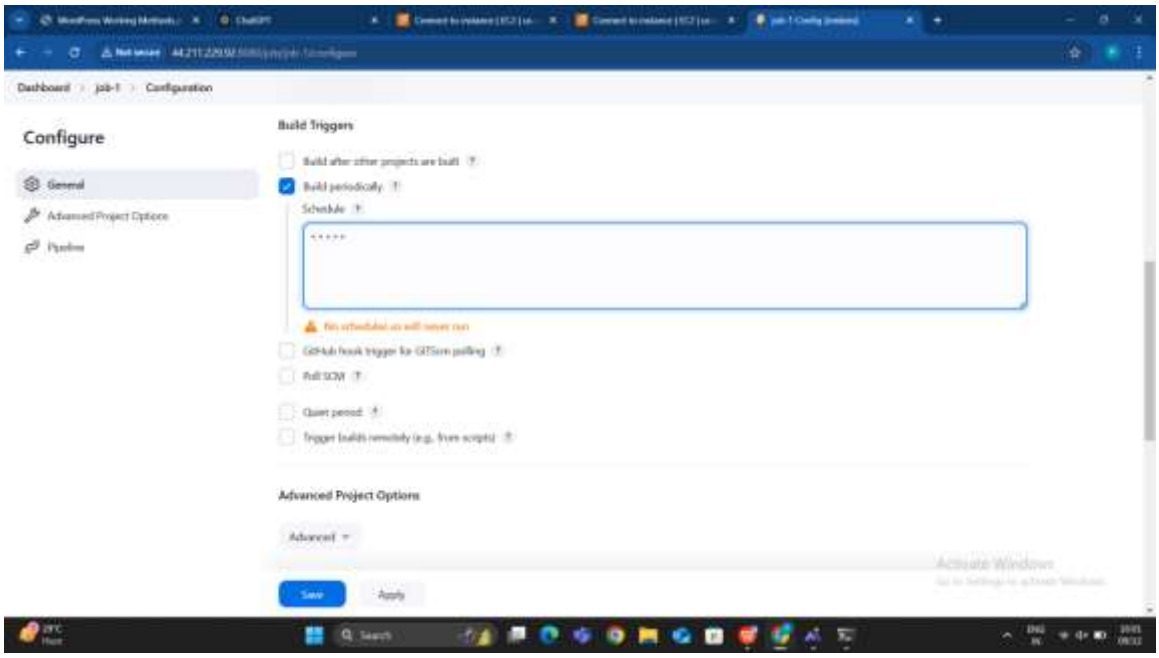
Add Triggers

1. Build Periodically

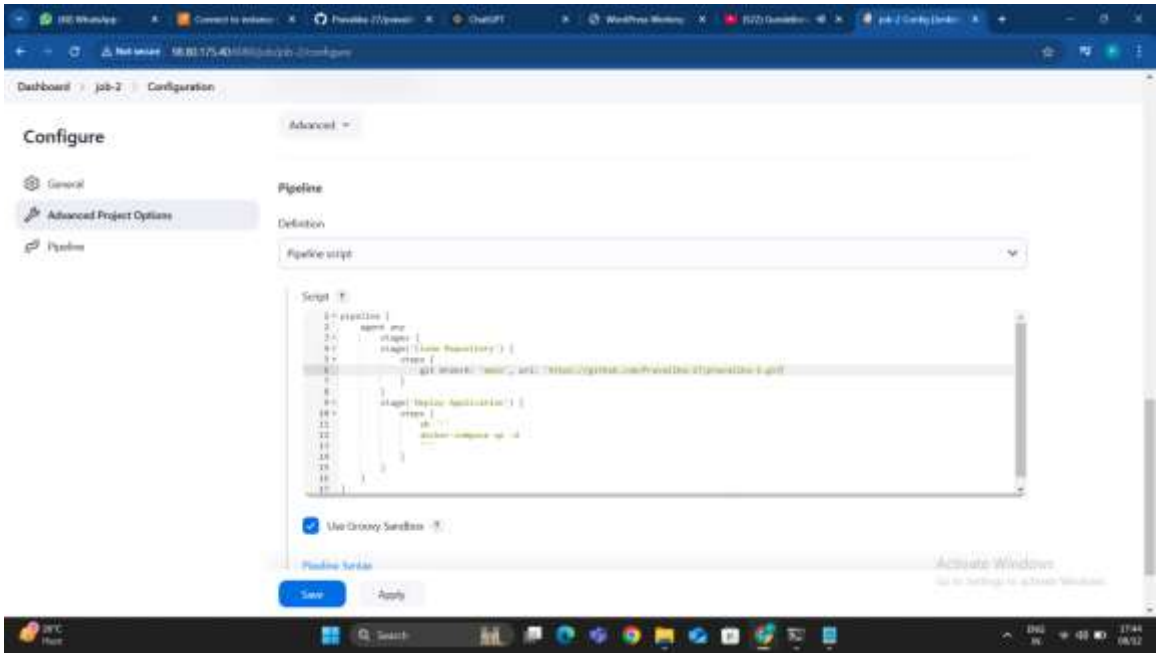
- This will trigger builds automatically at specific intervals.
- In the **Pipeline Configuration**, include:
  - cron('H/5 * * * *')
  - The H/5 means it will run every 5 minutes (replace 5 with your desired interval).

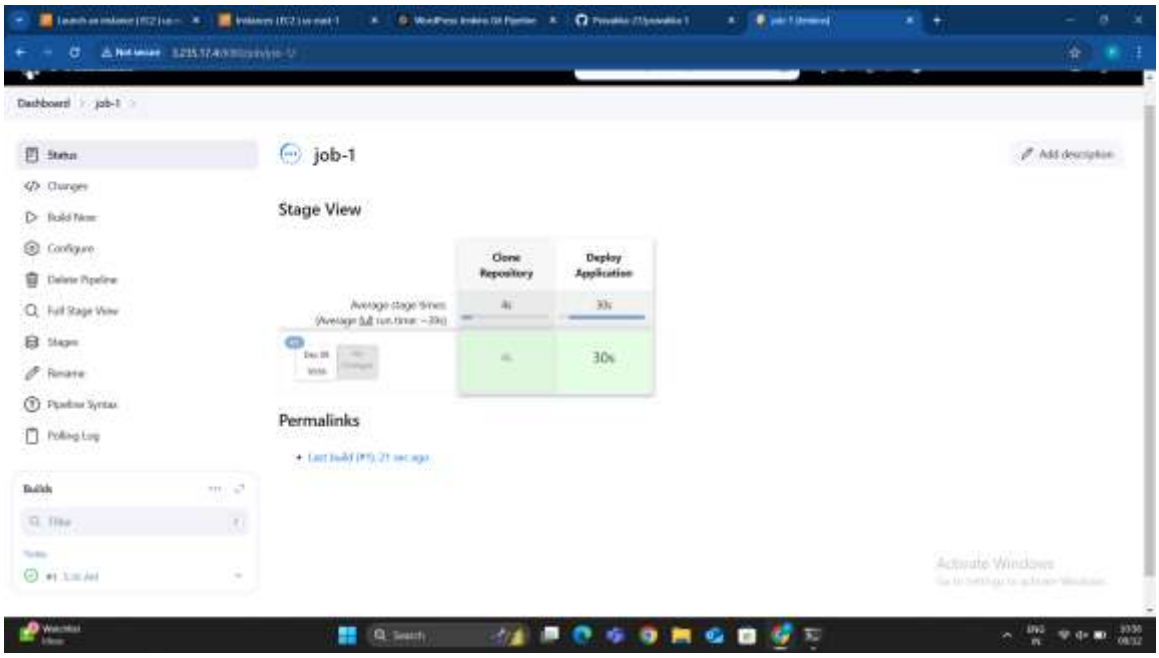
Save and Build

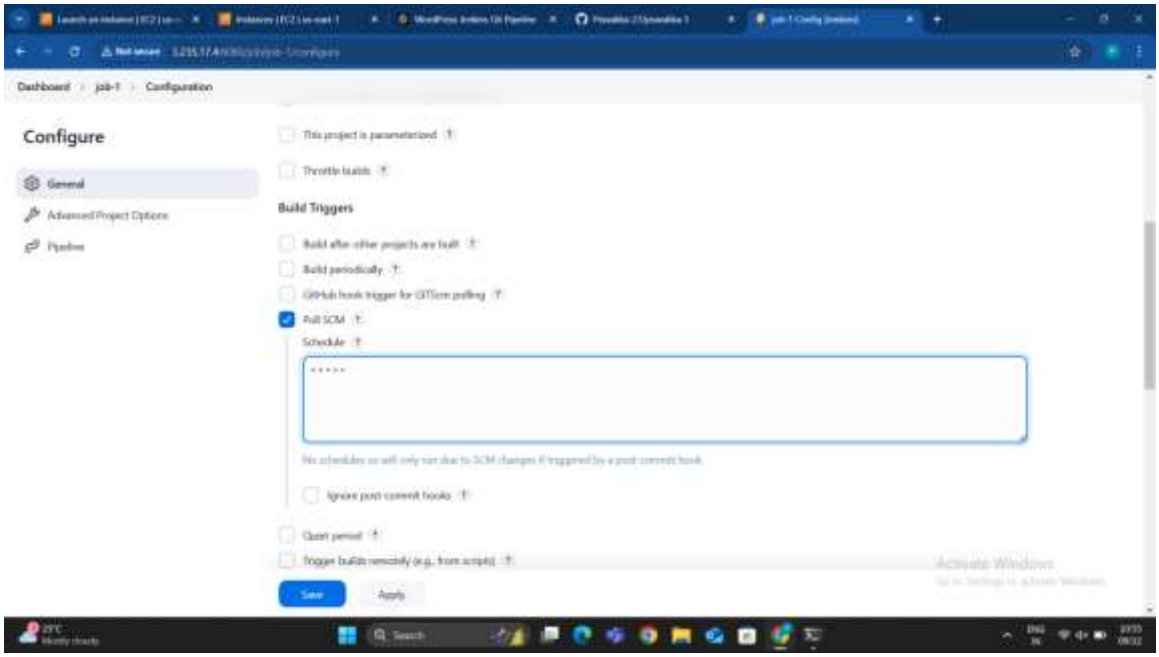
1. Save the pipeline job.
2. Click **Build Now** to trigger the first build.
3. Jenkins will:
   1. Periodically trigger builds based on the **cron schedule**.
   2. Poll the Git repository for changes and trigger builds if updates are detected.

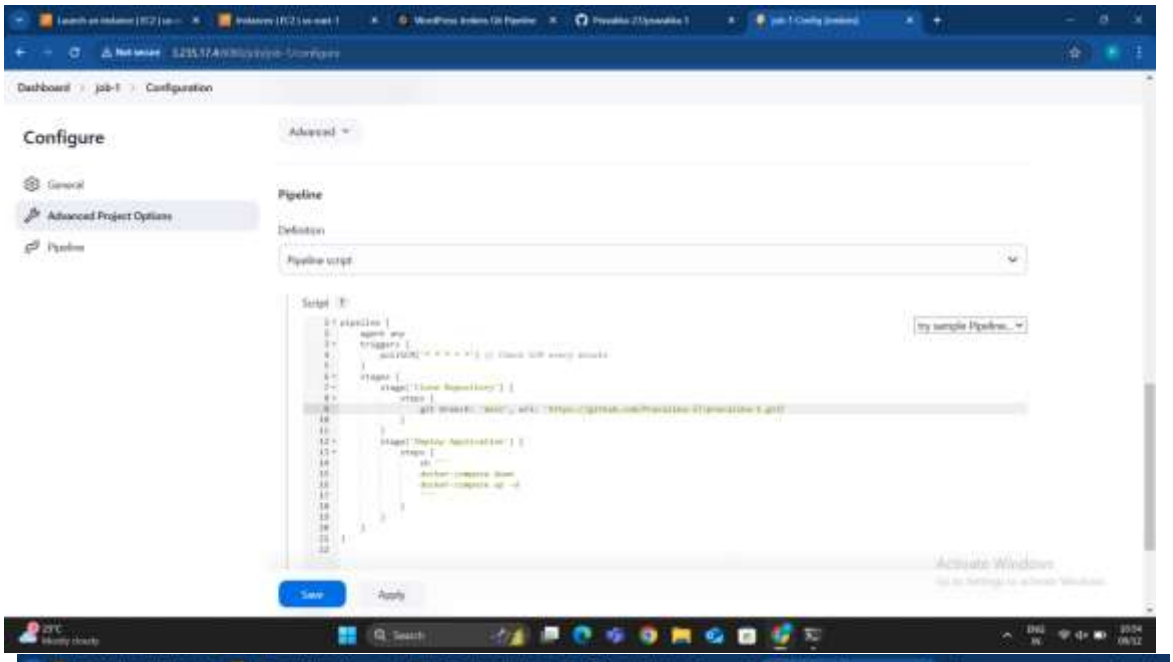


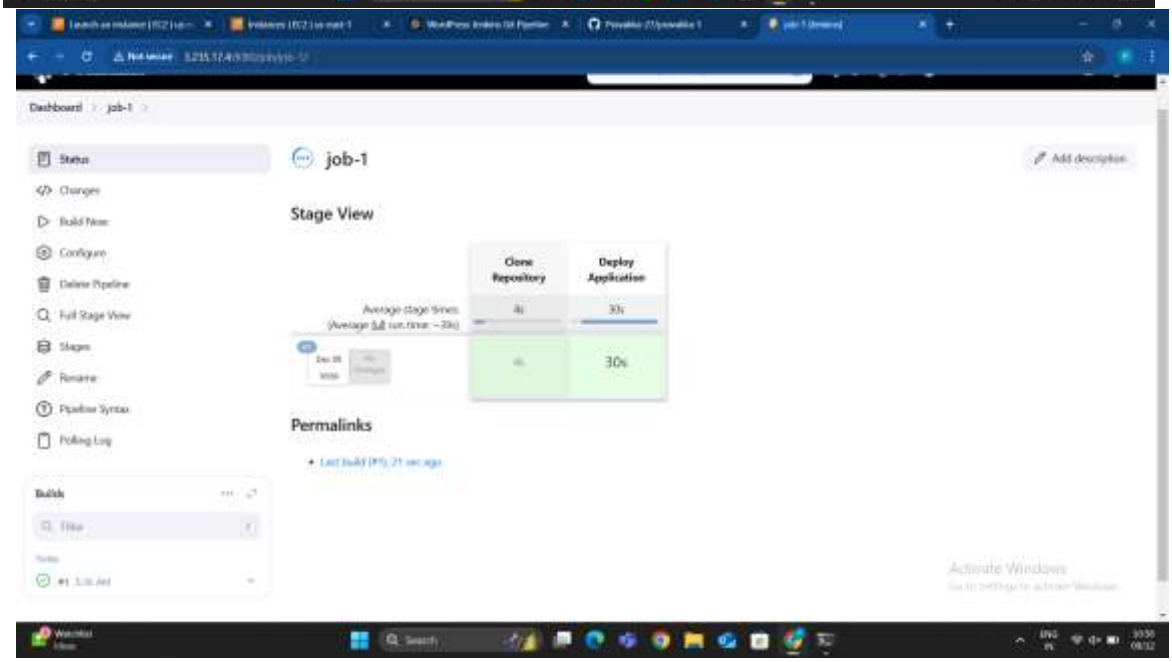
Write pipeline that clone the repository which consist of the docker-compose file and add the command docker-compose up -d to run the compose file




Dashboard
job-1
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Polling Log
Builds
job-1
Add description
Stage View
Clone Repository
Deploy Application
Average stage times:
(Average full run time: ~30s)
4s
30s
30s
Permalinks
Last build (#1), 21 sec ago
Activate Windows


Dashboard
job-1
Configuration
Configure
General
Advanced Project Options
Pipeline
This project is parameterized
Throttle builds
Build Triggers
Build after other projects are built
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Schedule
* * * * *
Ignore post-commit hooks
Quiet period
Trigger builds remotely (e.g., from scripts)
Save
Apply
Activate Windows





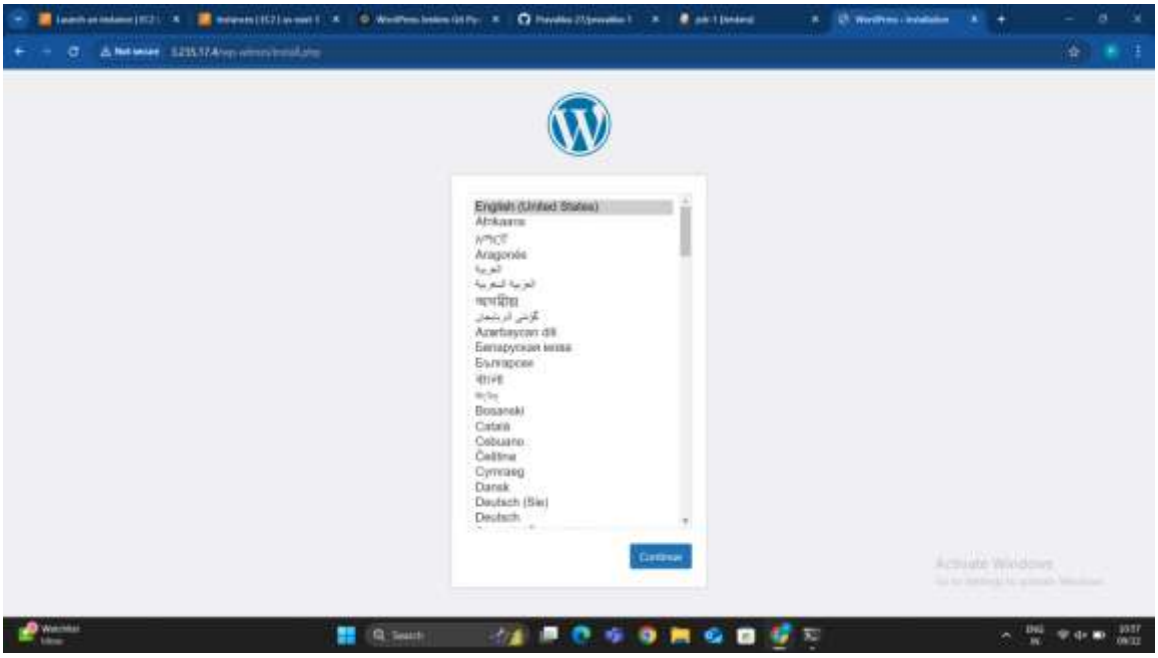
Verify Changes

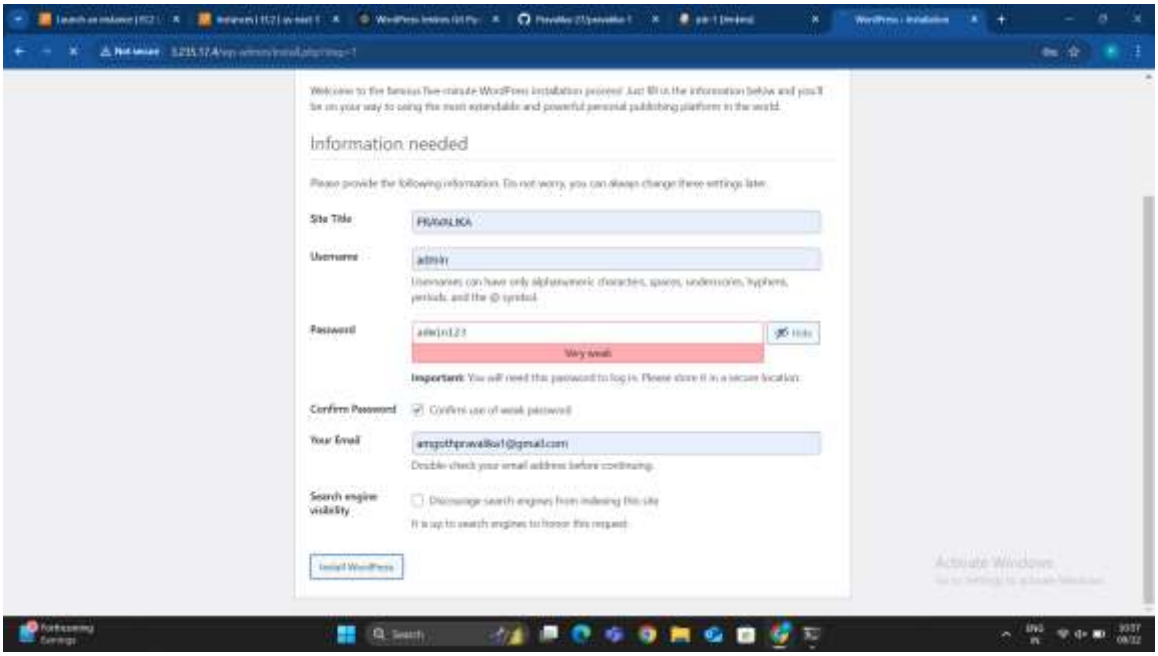
1. Make a change in the GitHub repository (e.g., edit a file, add a new theme, etc.).
2. Push the change to the repository

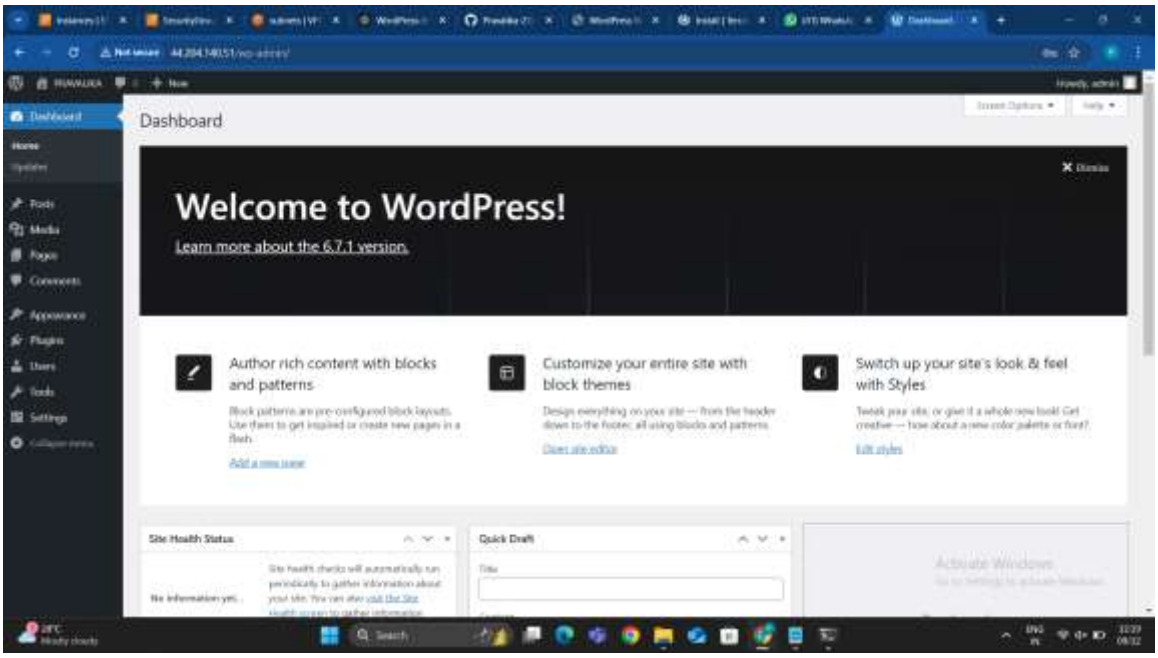
1. Monitor the Jenkins **Build History**:
    1. Check if Jenkins automatically triggered a build via **Poll SCM**.
    2. View the **Console Output** for logs to verify successful deployment.

Verify WordPress Application

1. Open your browser and visit your site: http://<server-ip>/wordpress.
2. Ensure the changes are reflected in the WordPress application.\







METHOD:7

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

Deploying a WordPress web application using **Terraform** involves automating the creation of an **EC2 instance** with the necessary configuration and a User Data script to set up WordPress. Below is the step-by-step process:

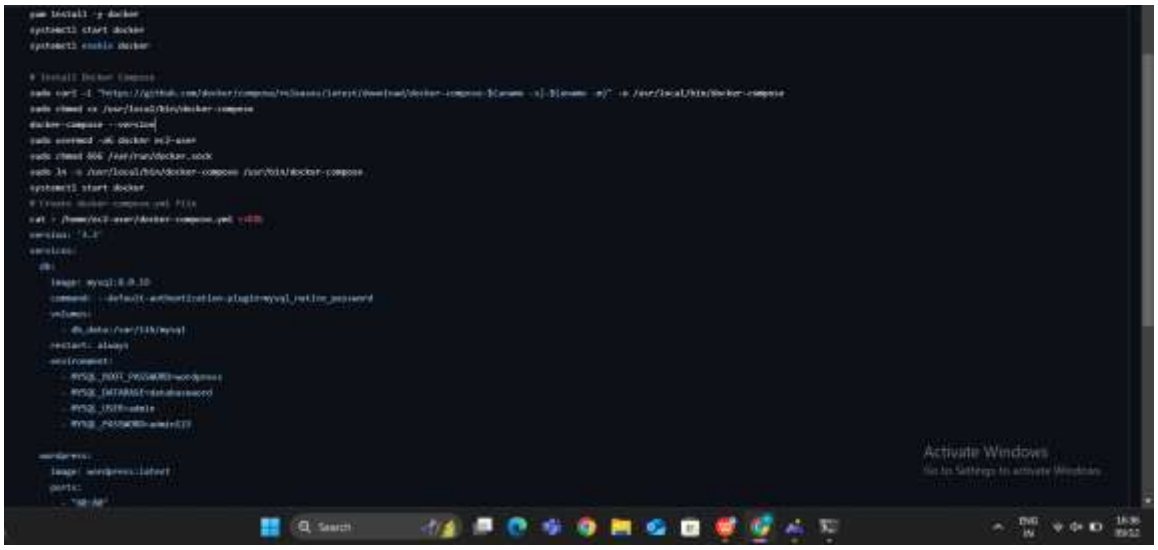
Step 1: Prerequisites

1. **AWS Account**: Ensure you have an AWS account with programmatic access.
2. **Install Terraform**:
    1. Download Terraform from here.
    2. Install it on your local machine
3. **IAM Role**:
    1. Create an IAM role with AmazonEC2FullAccess permissions or use existing access keys.
4. **Key Pair**:
    1. Create an AWS EC2 key pair for SSH access to the instance.
5. **Basic Networking**:
    1. A VPC with subnets, internet gateway, and route tables (Terraform can also provision these).

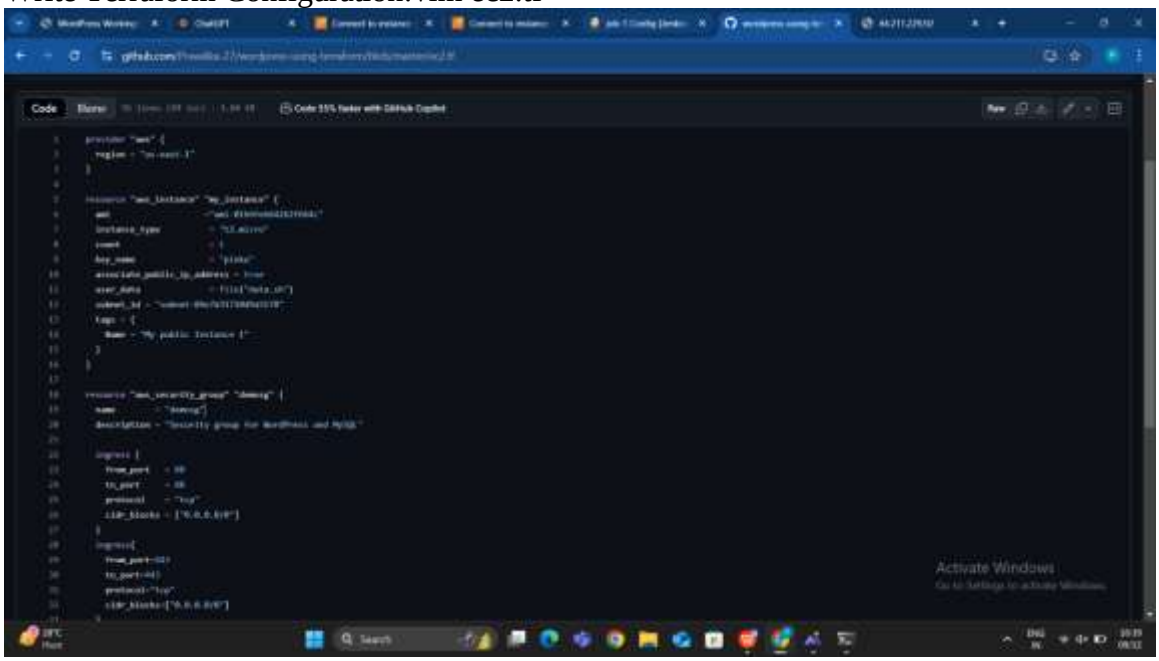
Step 2: Create the User Data Script

The User Data script automates the installation and configuration of WordPress.

Vim data.sh:



Write Terraform Configuration:vim ec2.tf



Initialize Terraform

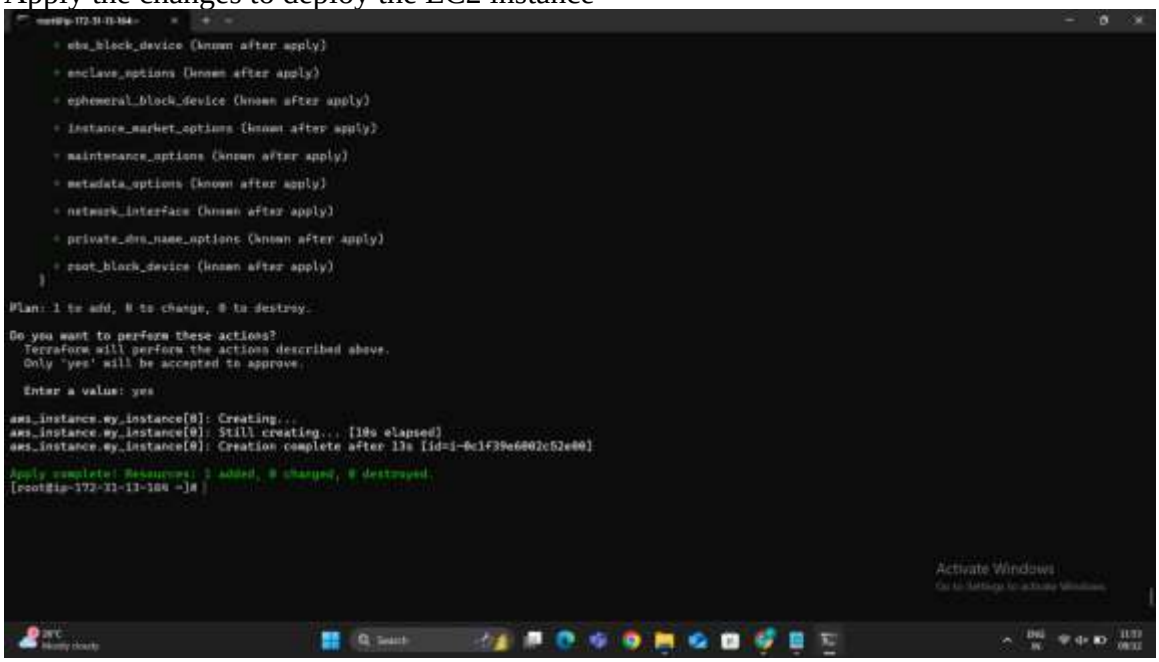
1. Open a terminal in the directory containing ec2.tf and user_data.sh
2. Run the following command:terraform init

Apply Terraform Configuration

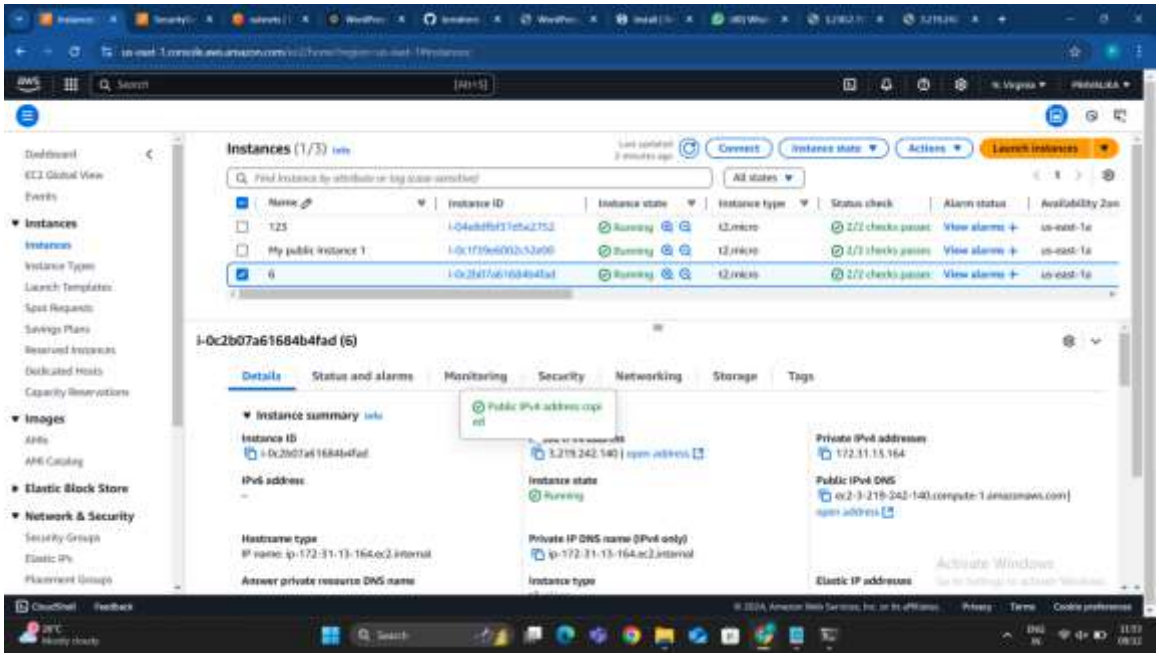
Validate the configuration using terraform validate command

Generate an execution plan:using terraform plan command

Apply the changes to deploy the EC2 instance

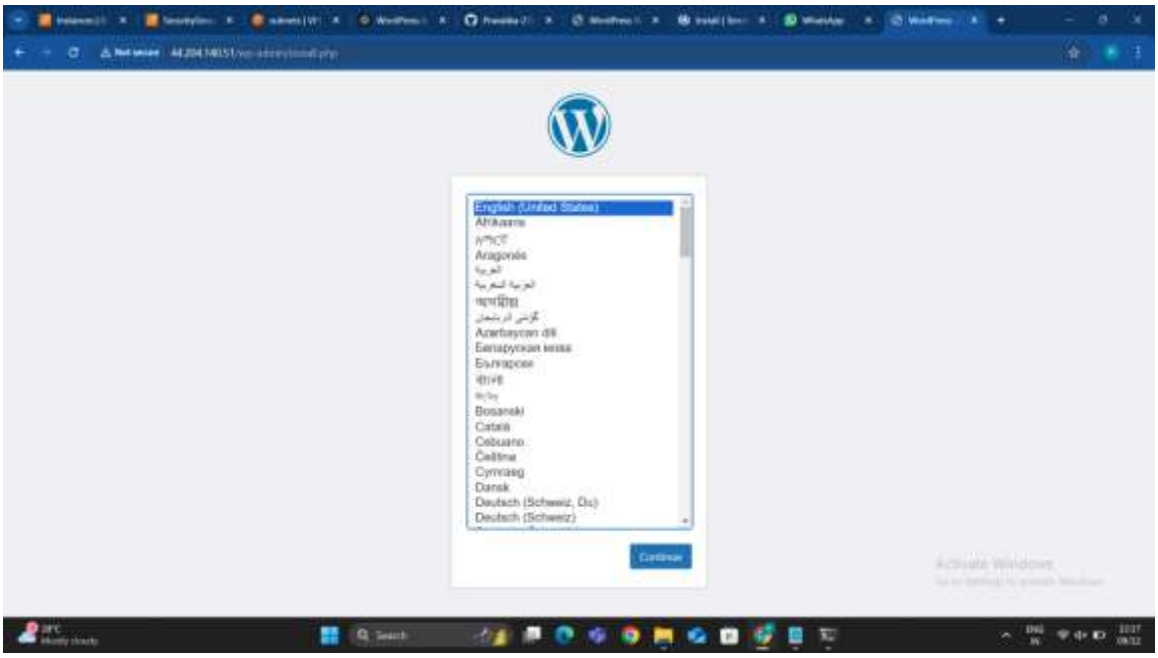


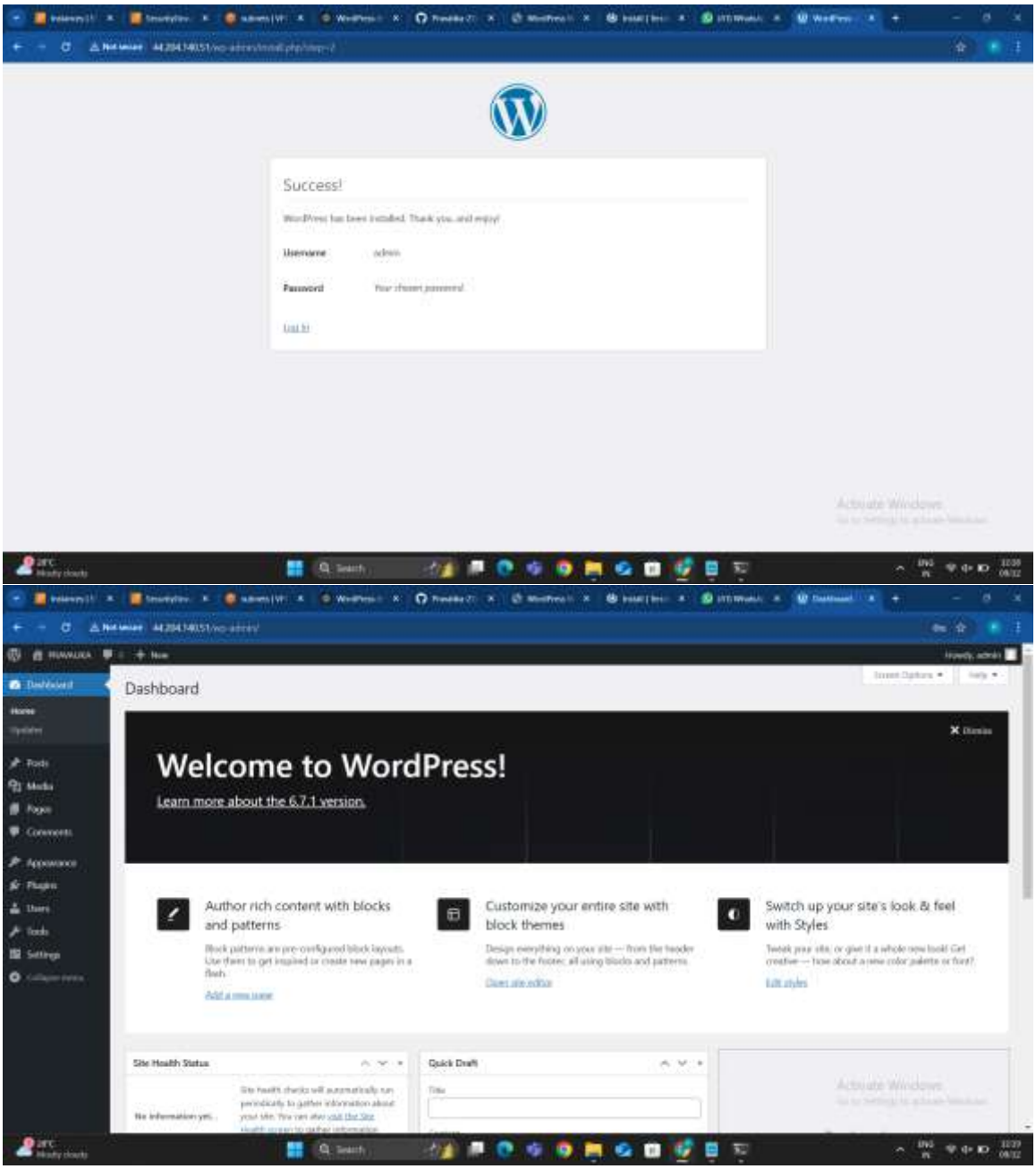
Verify the creation of ec2 instance



Verify Deployment

1. After successful execution, Terraform will output the public IP of the instance.
2. Open a browser and navigate to http://<public-ip>.
3. You should see the WordPress setup wizard.


English (United States)
Afrikaans
Aragonés
Azərbaycan dili
Bosanski
Català
Cebuano
Čeština
Cymraeg
Dansk
Deutsch (Schweiz, Du)
Deutsch (Schweiz)
Continue



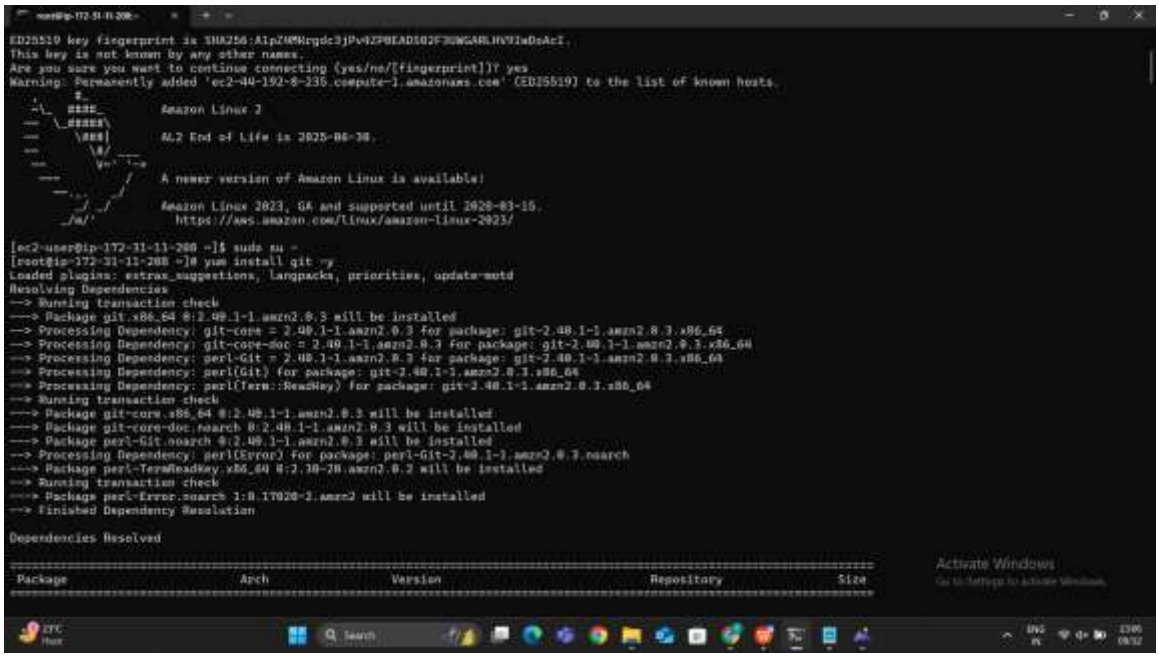
METHOD:8

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.

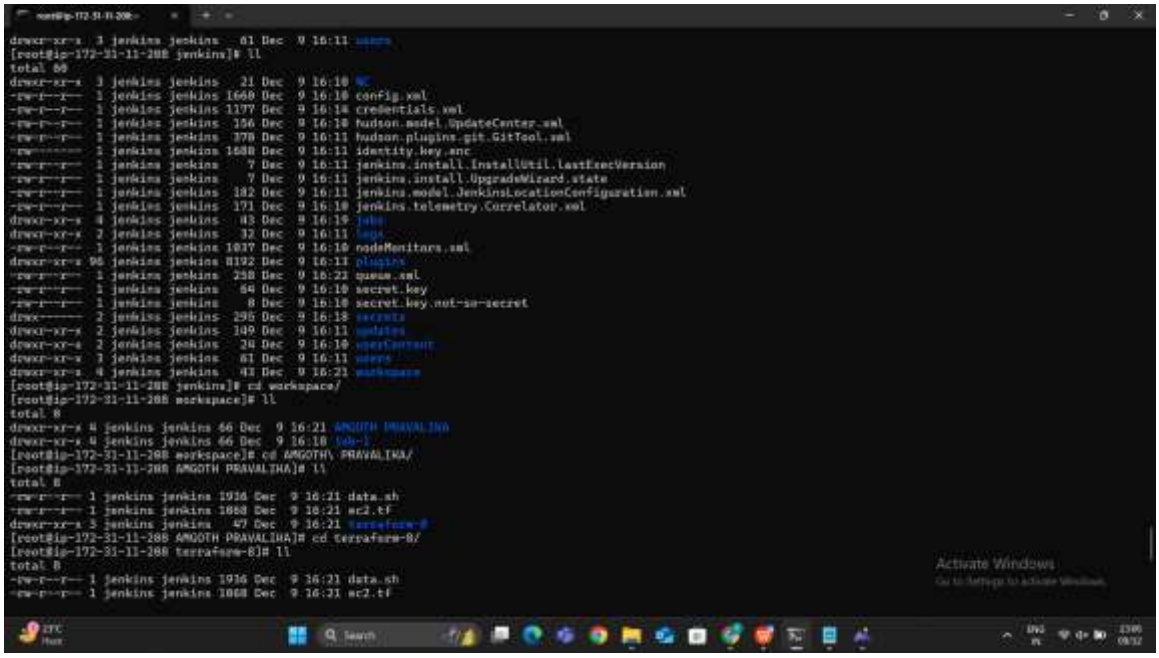
To deploy a WordPress application using a **Jenkins freestyle job**, **Terraform**, and the Terraform scripts in your GitHub repository (ec2.tf and data.sh), follow the step-by-step process below.

Prerequisites

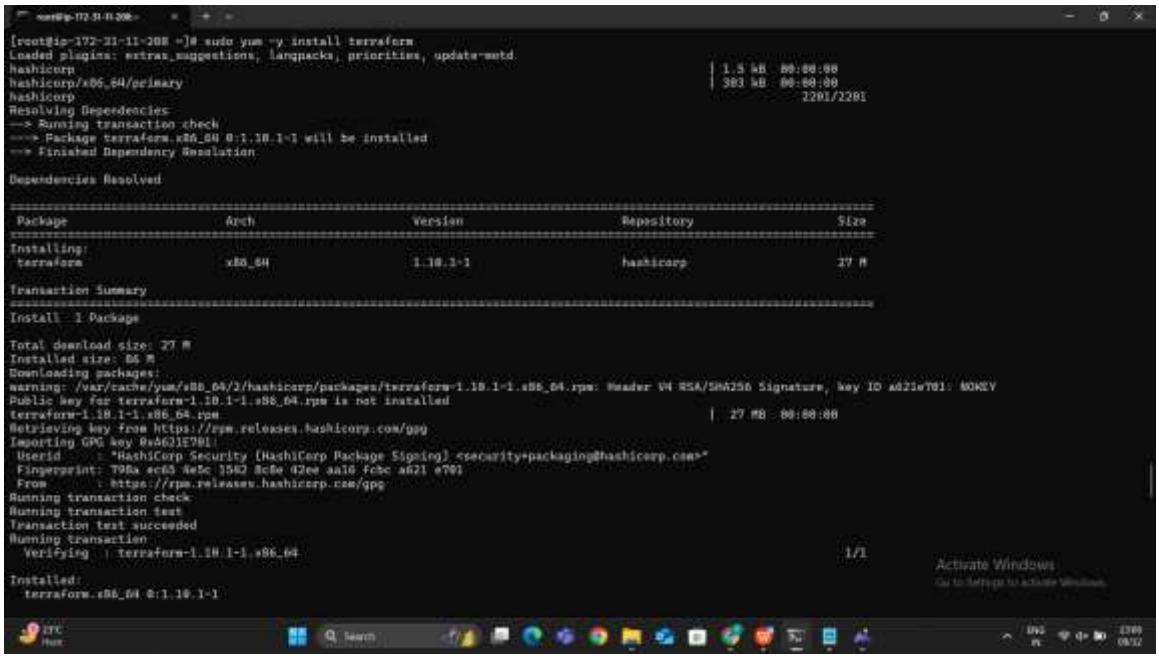
1. **AWS Account** with proper IAM credentials.



2. **Jenkins Server** installed and running.



3. **Terraform** installed (can be installed via Jenkins).



4. Your GitHub repository URL: https://github.com/Pravalika-27/terraform-8.git.
5. AWS credentials set up using the **AWS Credentials Plugin** in Jenkins.

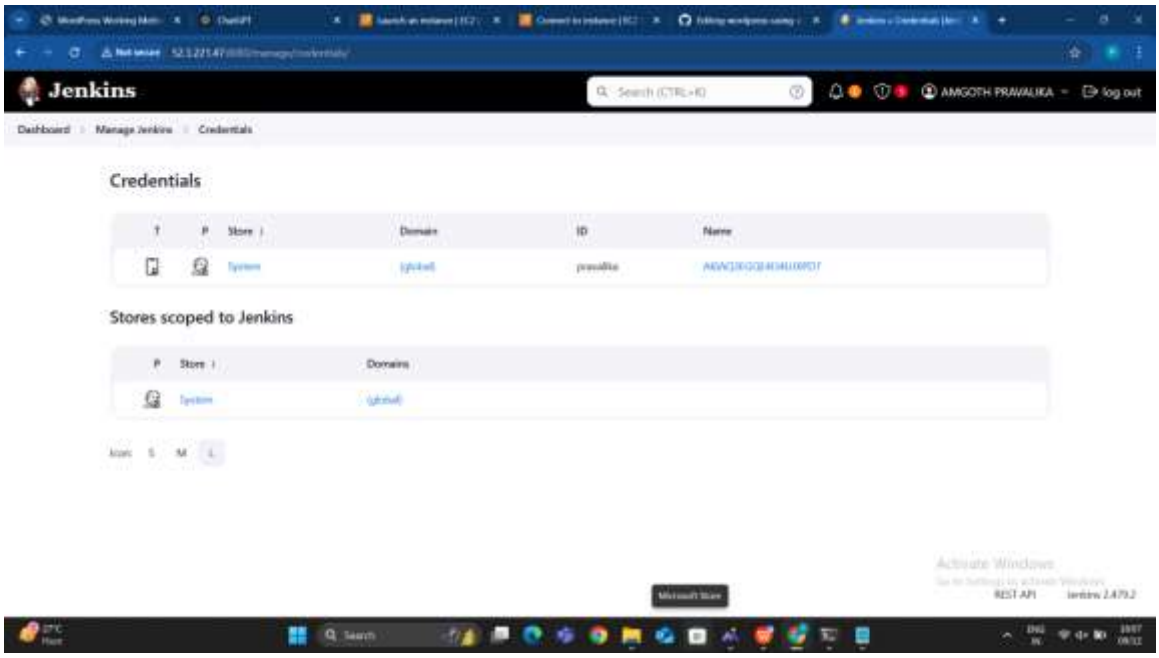
Step 1: Configure AWS Credentials in Jenkins

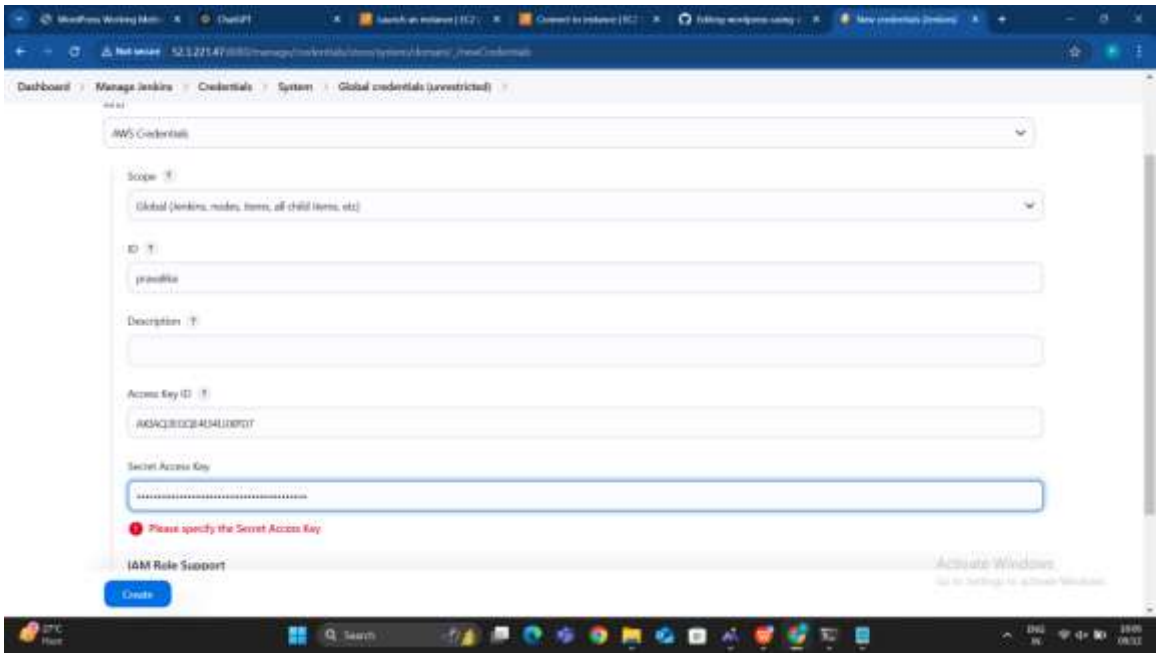
Install the **AWS Credentials Plugin**:

1. Go to **Manage Jenkins** > **Manage Plugins** > **Available Plugins**.
2. Search for AWS Credentials and install it.

Add AWS Credentials:

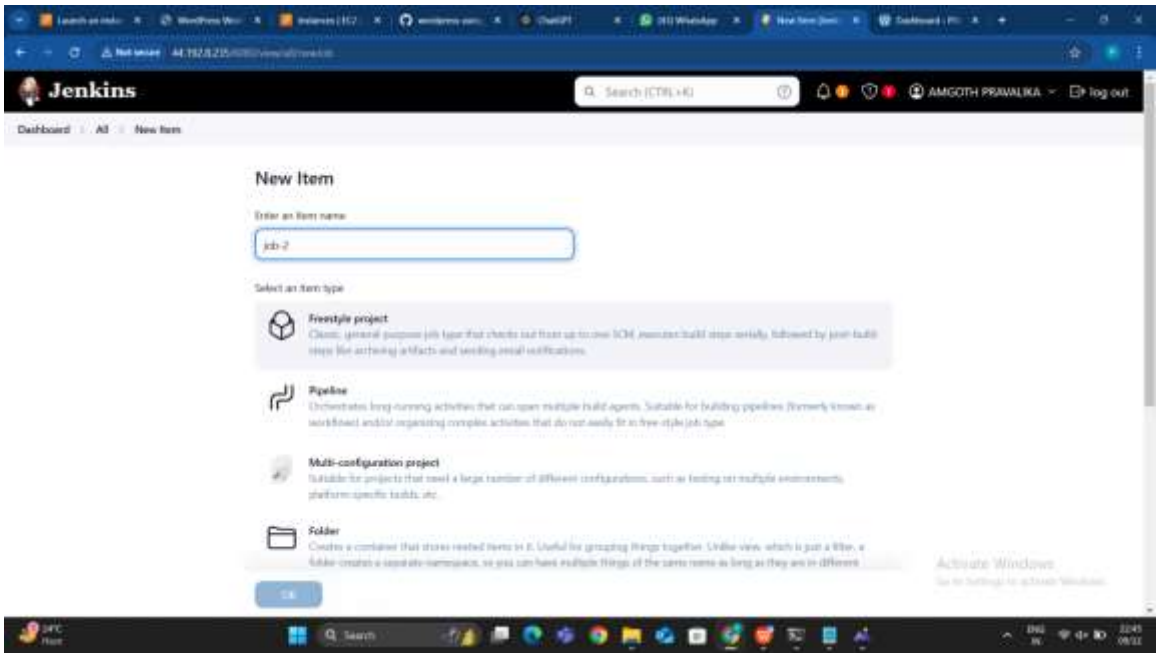
- Go to **Manage Jenkins** > **Manage Credentials**.
- Select a credentials domain (e.g., **Global**).
- Click **Add Credentials** > **AWS Credentials**.
- Enter your **AWS Access Key ID** and **Secret Access Key**.
- Give the credentials an ID (e.g., aws-credentials).





Step 2: Create a Jenkins Freestyle Job

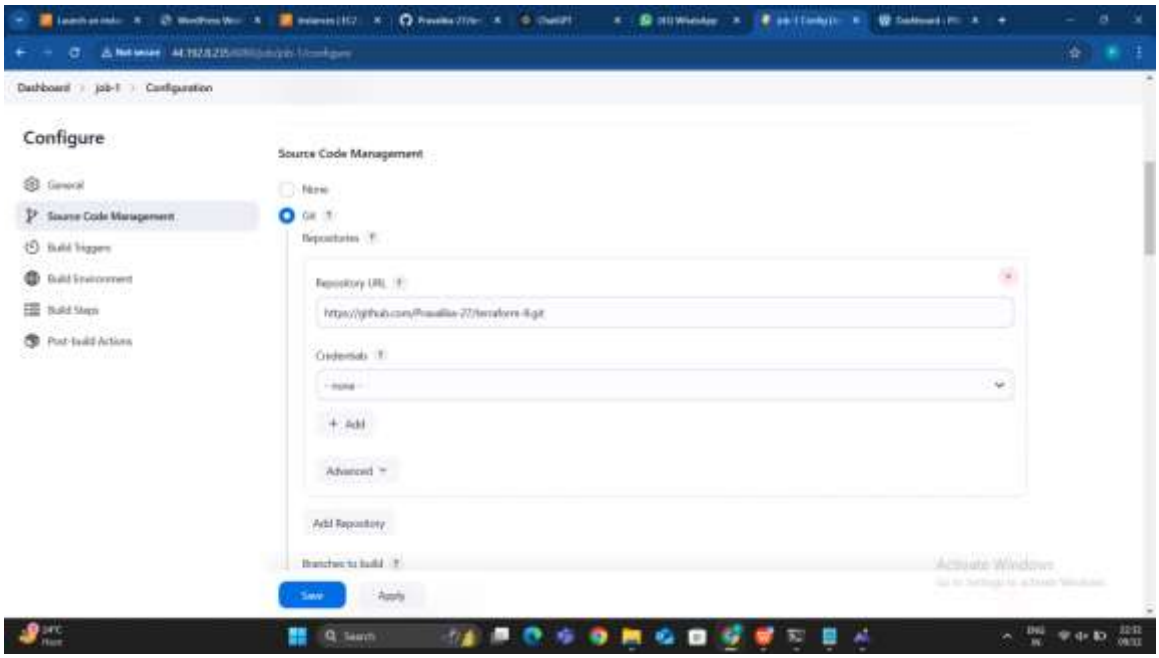
1. Go to **Jenkins Dashboard** > **New Item**.
2. Enter the name for your job (e.g., Deploy WordPress) and select **Freestyle Project**.
3. Click **OK**.



Step 3: Configure the Job

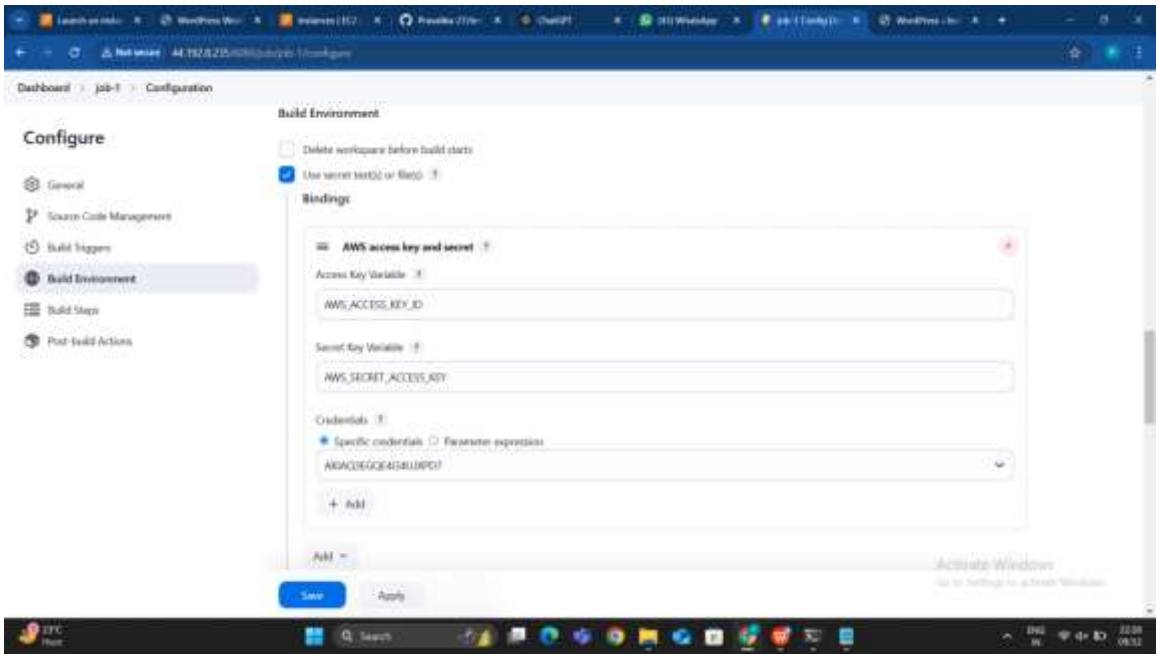
Step 3.1: Add Git Repository

1. Under the **Source Code Management** section, select **Git**.
2. Enter your GitHub repository URL: https://github.com/Pravalika-27/terraform-8.git.
3. If the repository is private, provide GitHub credentials by clicking **Add**.



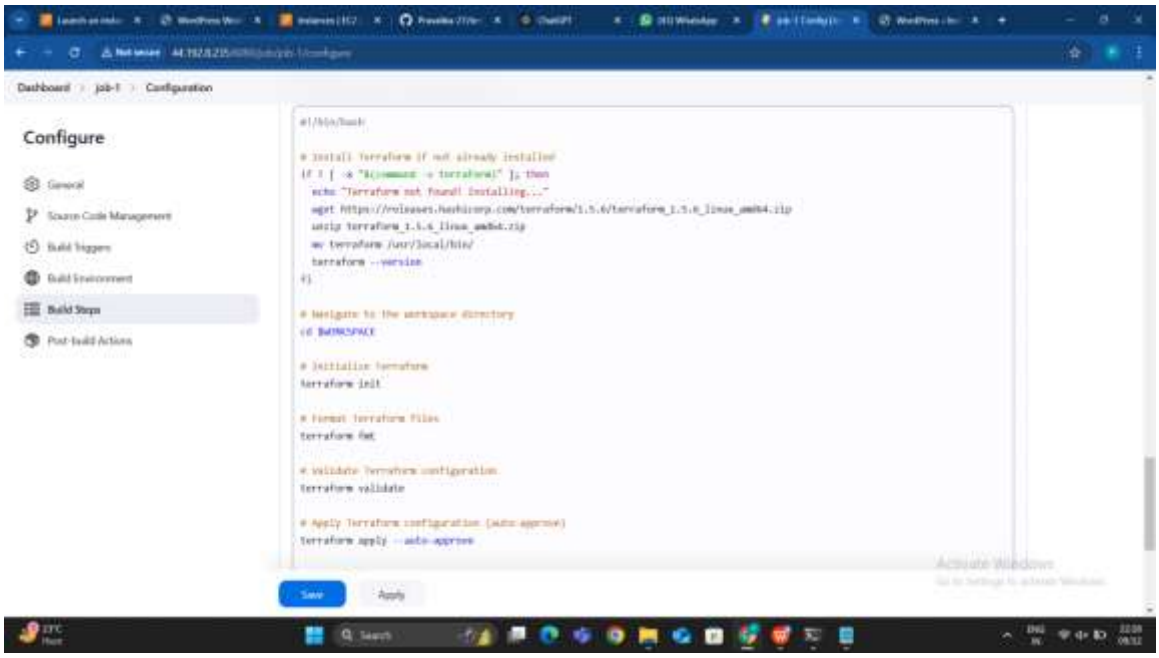
Step 3.2: Add AWS Credentials

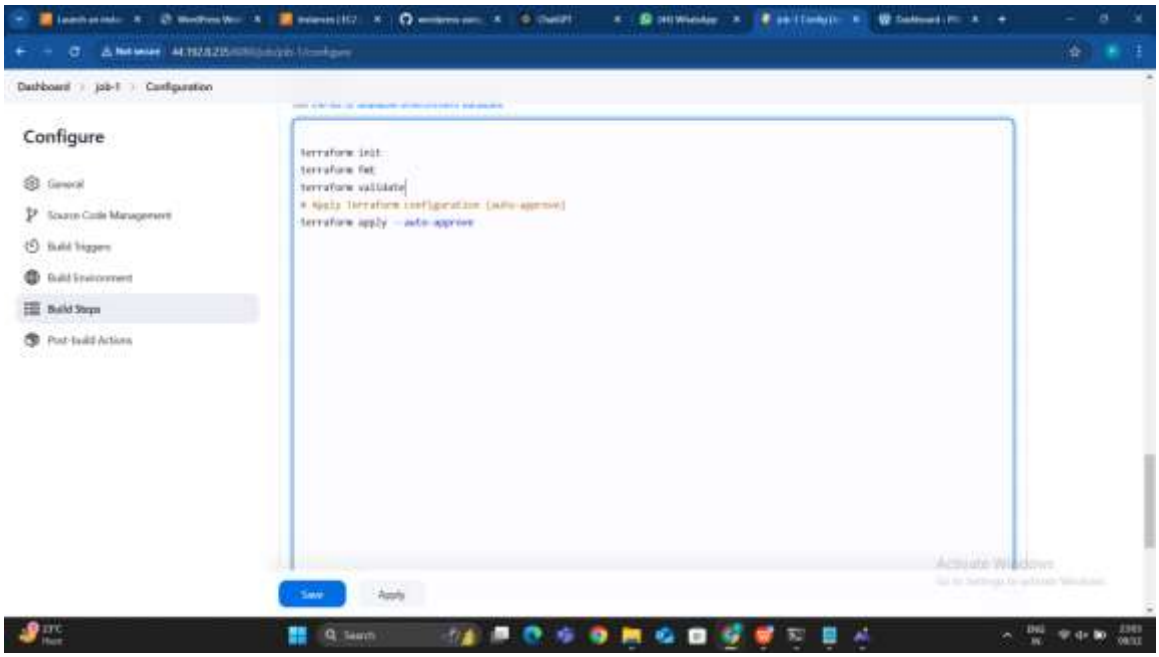
1. Go to the **Build Environment** section.
2. Check **Use secret text(s) or file(s)**.
3. Add your AWS credentials (created in Step 1) by selecting the **AWS credentials ID**.



Step 3.3: Add Execute Shell

In the **Build** section, add an **Execute Shell** build step. Paste the following script:





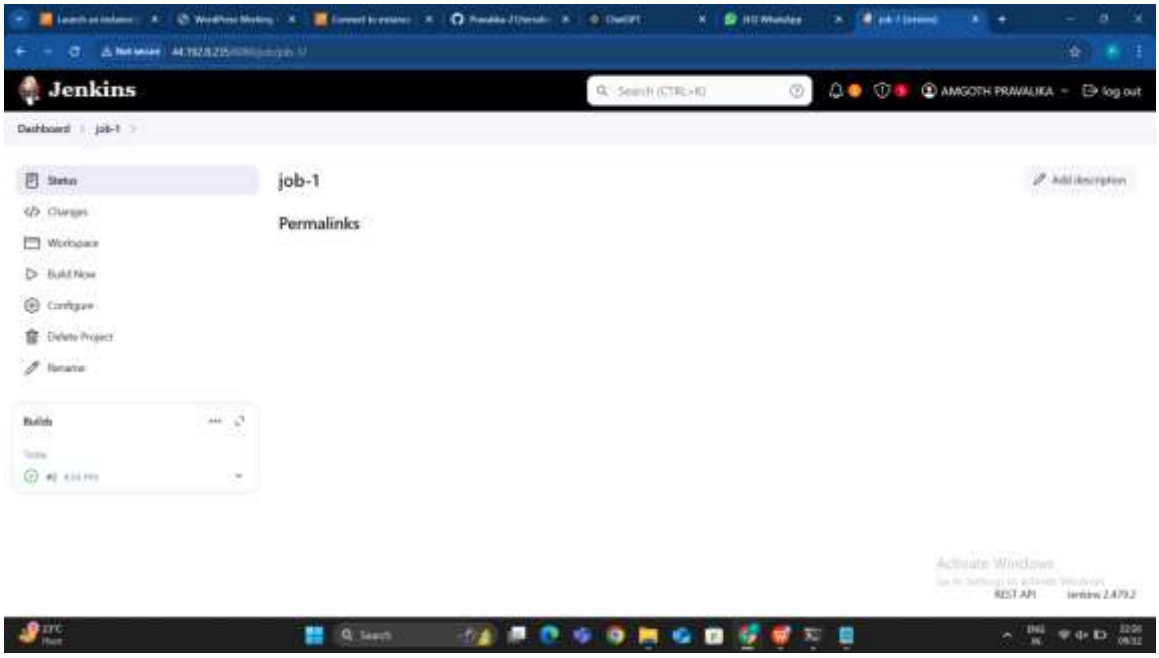
Step 4: Add Permissions to Your EC2 Instance

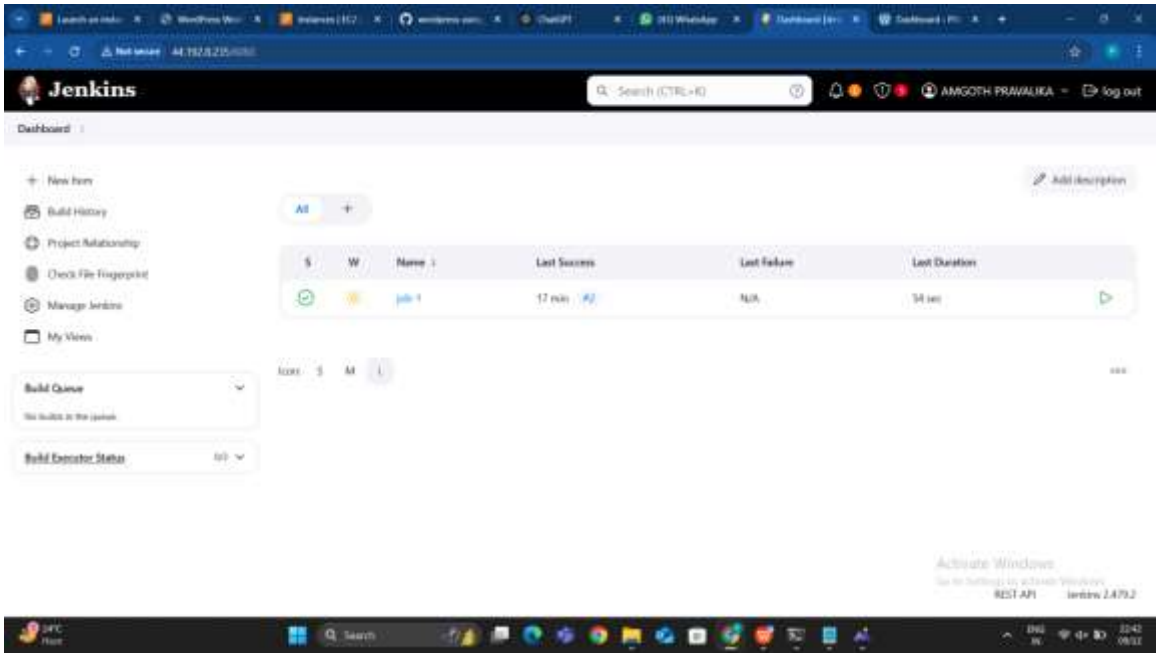
Ensure the EC2 instance created by ec2.tf has:

1. **Security Group** allowing HTTP (port 80) traffic for WordPress.
2. **IAM Role** (if necessary) to perform AWS operations.

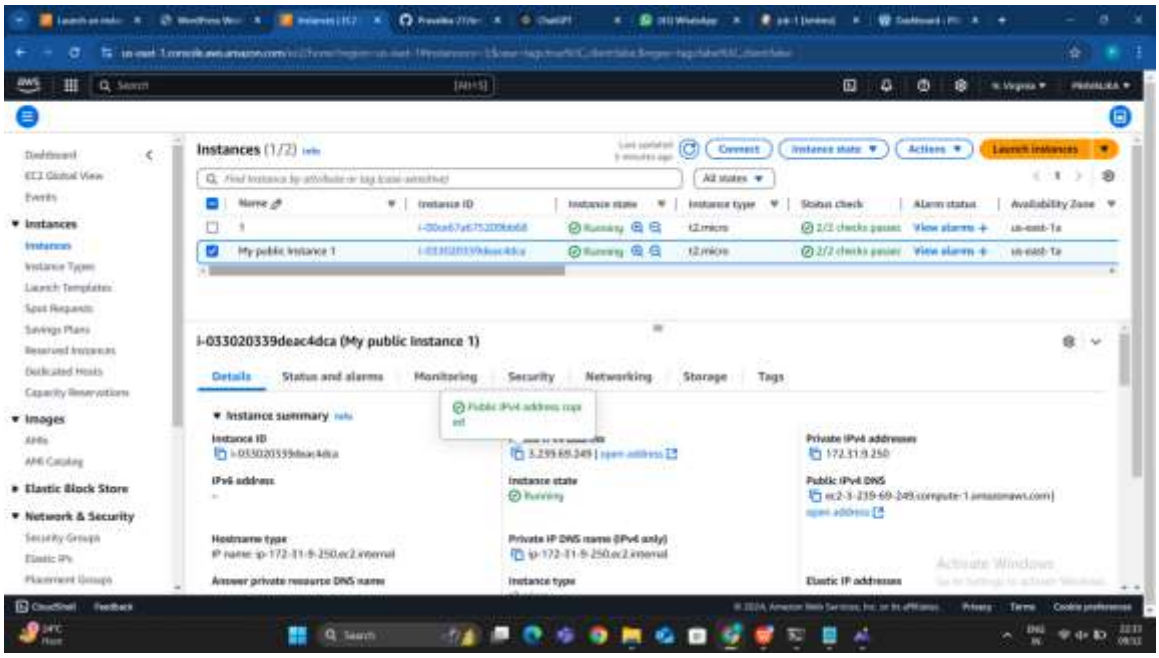
Step 5: Test the Job

1. Save the Jenkins job configuration.
2. Click **Build Now** on the Jenkins job dashboard.
3. Monitor the **Console Output** to ensure Terraform runs successfully.



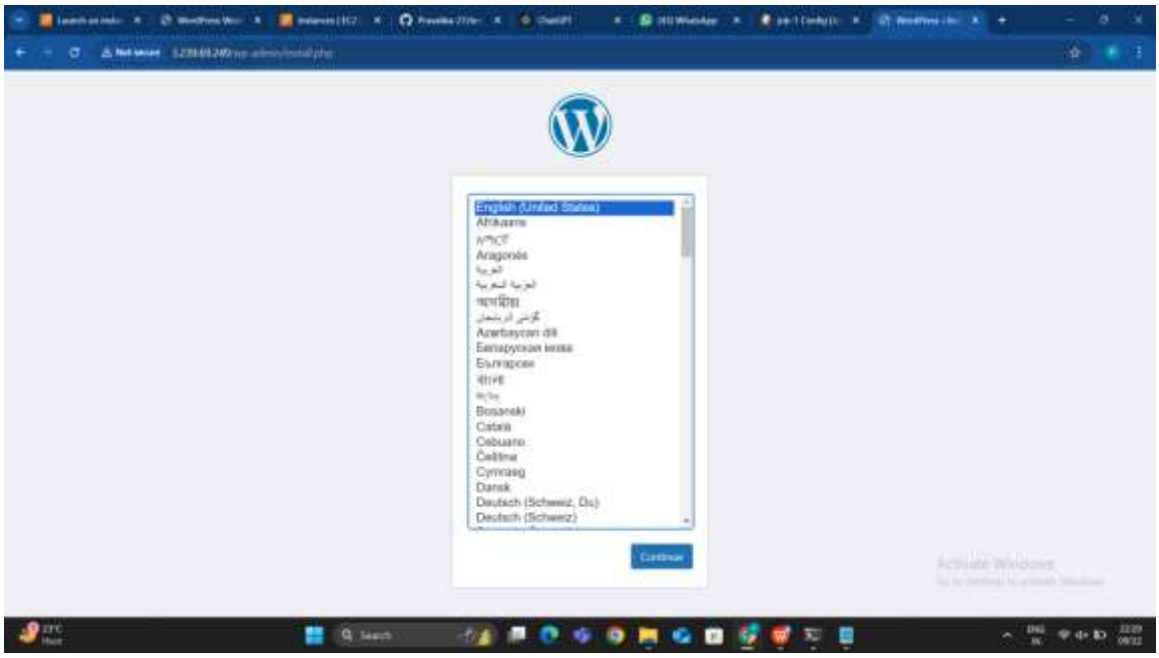


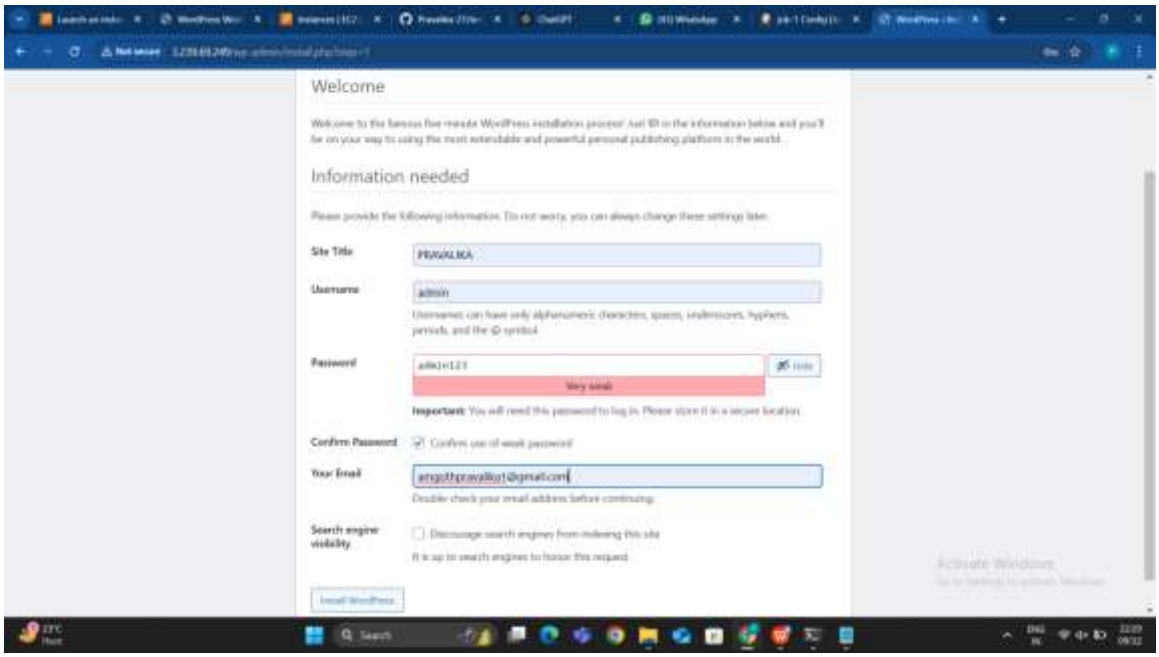
After building the job check for the instance here new instance is created

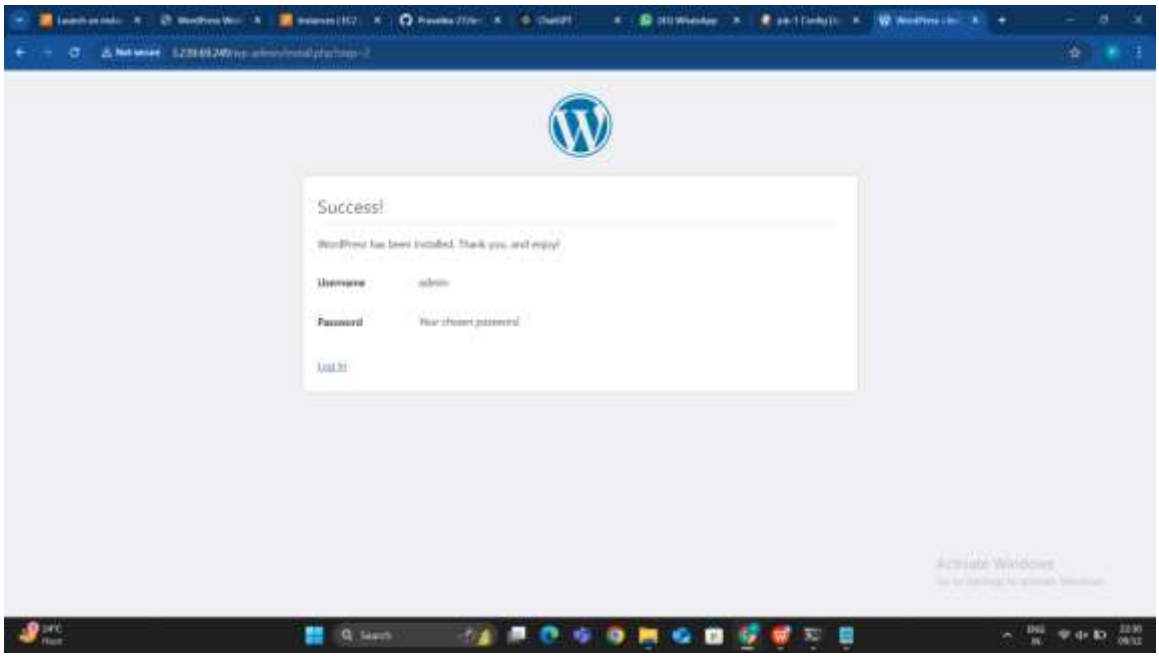


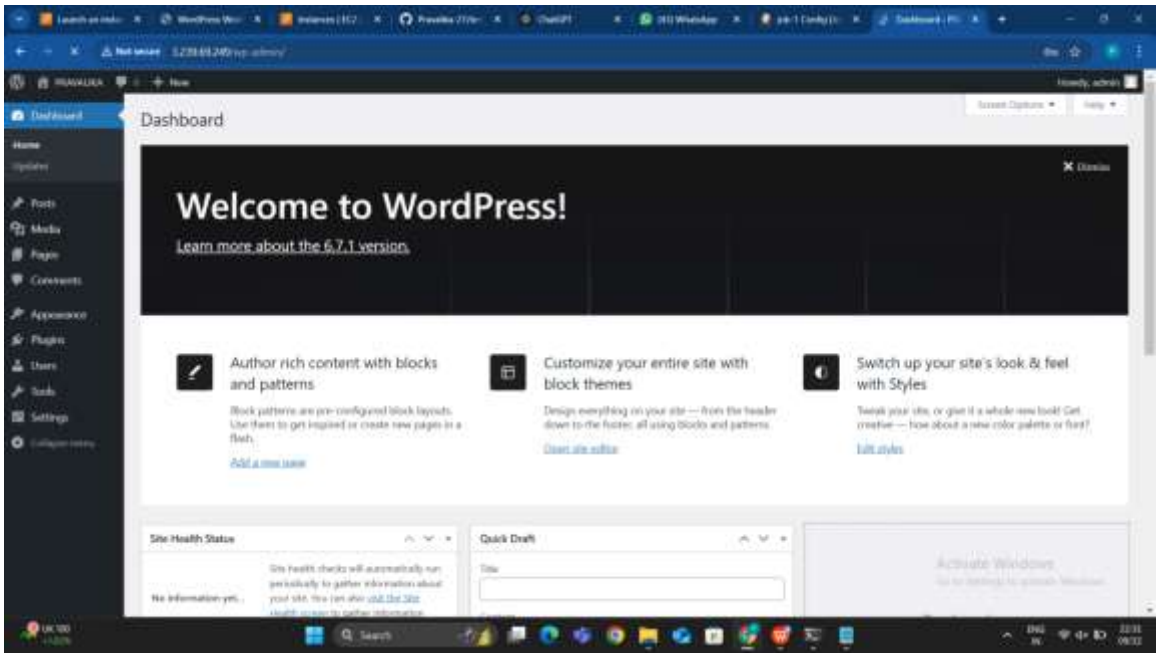
Step 6: Verify the Deployment

1. After the job completes, note the **public IP address** of the deployed EC2 instance (this should be part of the Terraform outputs).
2. Open a browser and navigate to http://<Public-IP>.
3. You should see the WordPress installation page.


English (United States)
Afrikaans
Aragonés
Azərbaycan dili
Беларуская мова
Български
Bosanski
Català
Cebuano
Čeština
Cymraeg
Dansk
Deutsch (Schweiz, Du)
Deutsch (Schweiz)
Continue


Welcome
Information needed
Site Title
PRAVALIKA
Username
admin
Password
Very weak
Confirm Password
Your Email
Search engine visibility
Install WordPress

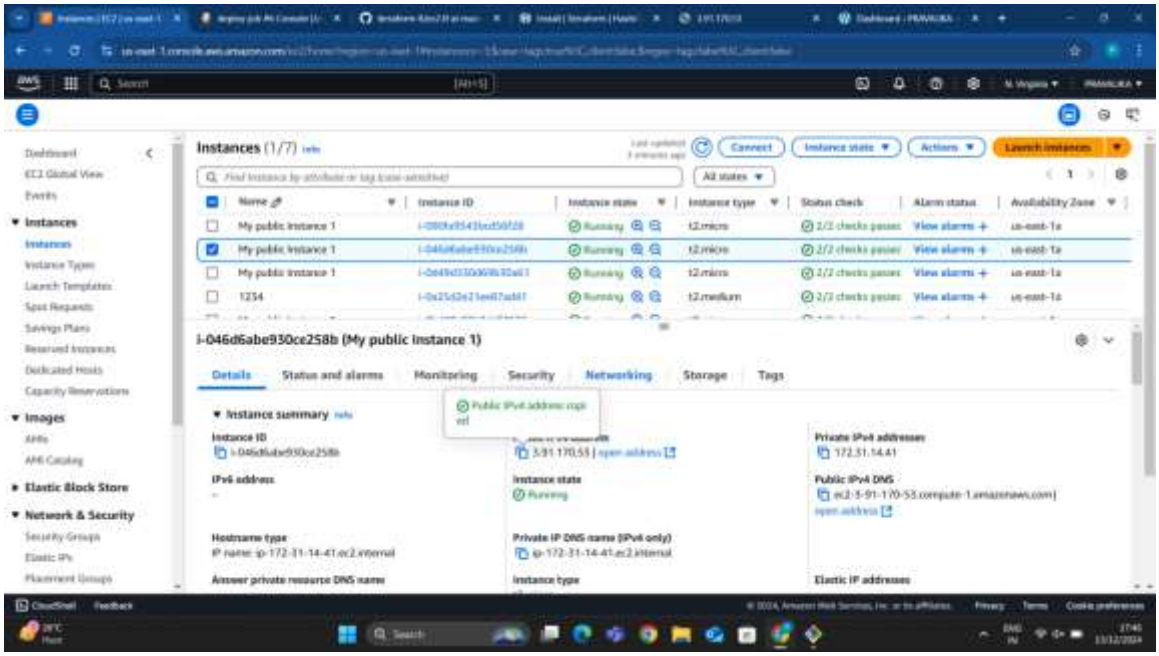


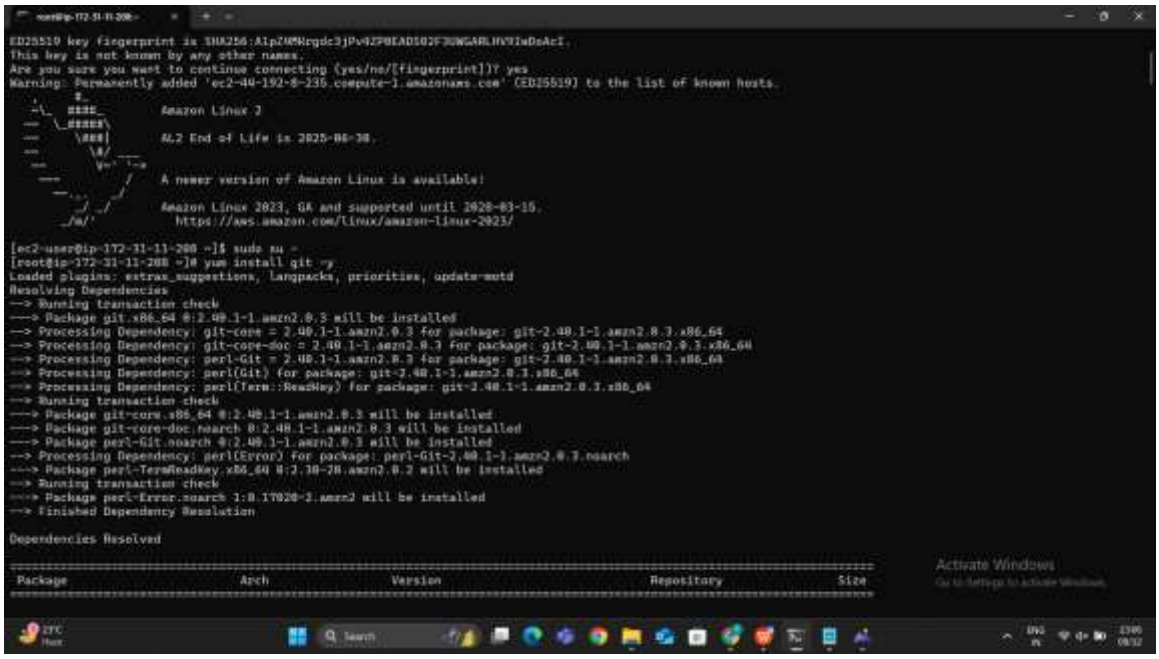


METHOD-9:

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.





Step 1: Install Necessary Plugins

Before creating jobs, ensure the required plugins are installed:

- **Git Plugin**
- **Pipeline Plugin**
- **Aws credentials**

Install and Configure AWS Credentials Plugin

The AWS Credentials Plugin helps manage AWS credentials securely within Jenkins.

Install the AWS Credentials Plugin:

1. Navigate to the **Available** plugins list in the **Plugin Manager**.
2. Search for "AWS Credentials Plugin".
3. Select it and install without restart.

Configure AWS Credentials:

1. After installation, navigate to **Manage Jenkins** > **Manage Credentials**.
2. Choose the appropriate **scope** (Global or folder-specific).
3. Click on **Add Credentials**.

Add AWS Credentials:

From the **Kind** dropdown, select **AWS Credentials**.

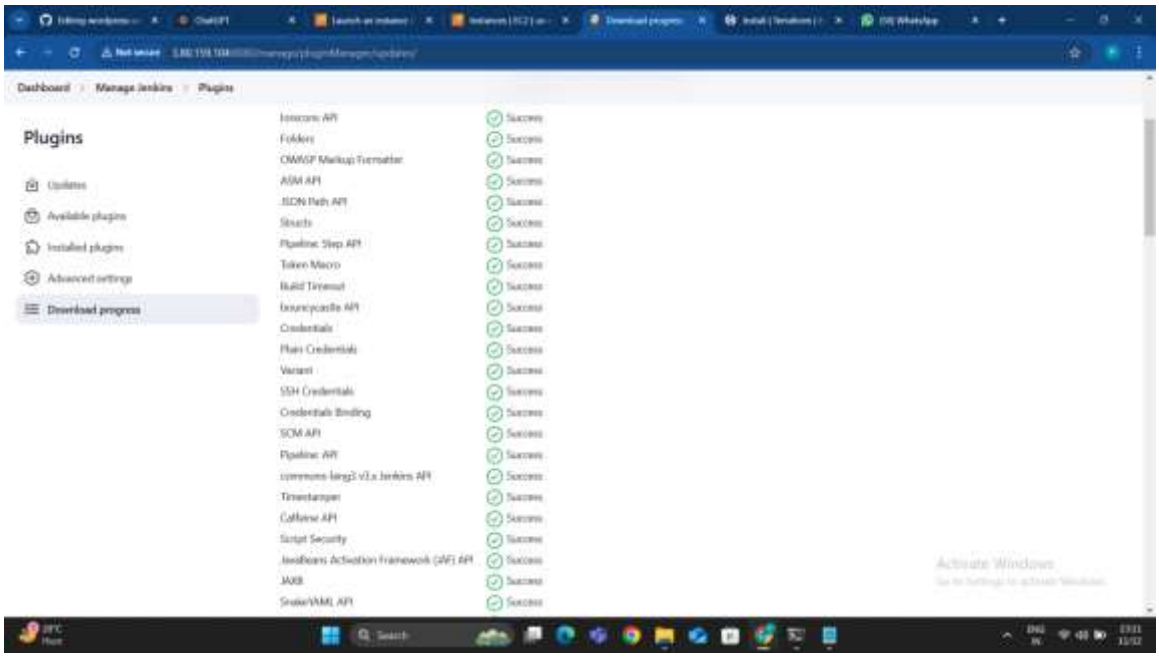
1. Provide the following:
    1. **Access Key ID**: The AWS access key.
    2. **Secret Access Key**: The AWS secret access key.
    3. **ID** (Optional): An identifier for these credentials.
    4. **Description**: A description for easy reference.
2. Click **OK** to save.

Verify Configuration:

1. Ensure the AWS credentials appear in the selected scope and are available for jobs or pipelines.

Install plugins

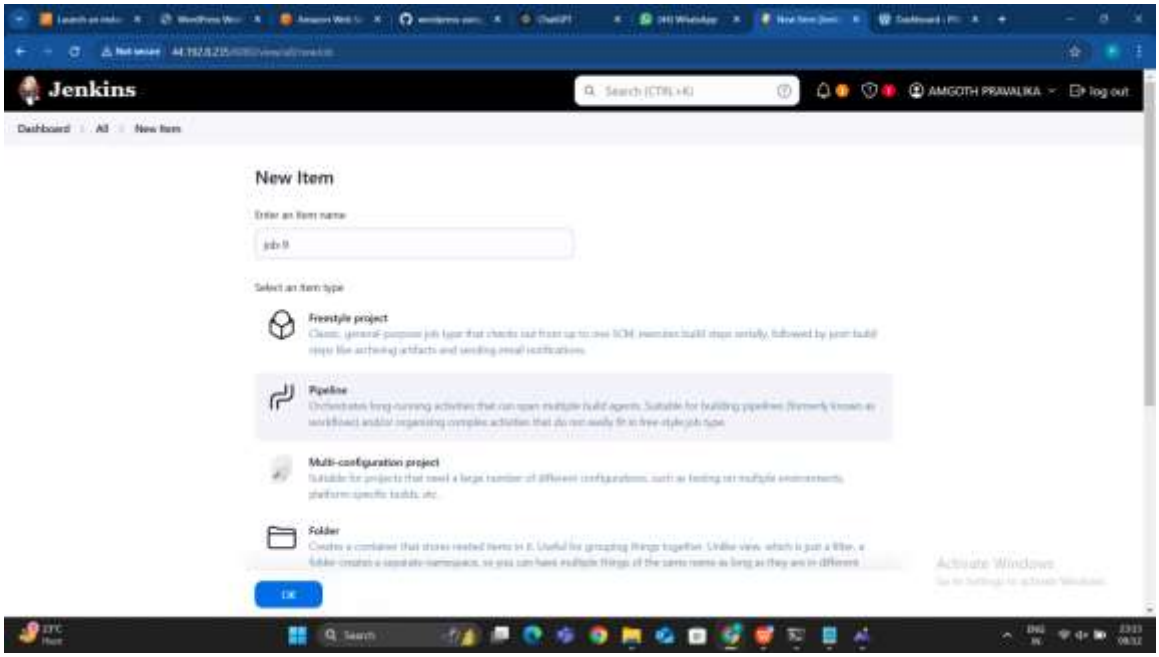
1. aws credentialns
2. Build pipeline



Step 2: Create Job-1 (Free Tier Job)

1. Navigate to Jenkins Dashboard:

- Click **"New Item"** > **"Freestyle Project"**.
- Enter a name: job-1 and click **OK**.



2. Configure Job-1:

Source Code Management:

- Select **Git**.
- Add the **Git URL** of your repository containing the source code or Terraform files.

2. Configure Source Code Management

Access the Job Configuration:

1. After creating the job, you will be taken to the configuration page.
2. If you are editing an existing job, click on **Configure** from the job menu.

Select Source Code Management:

1. Scroll to the **Source Code Management** section.
2. Select **Git**.

Provide the Git Repository URL:

1. Enter the **Git URL** of your repository. This URL can be HTTPS or SSH-based, depending on your setup.
    1. **Example HTTPS URL**: https://github.com/username/repository.git
    2. **Example SSH URL**: git@github.com:username/repository.git

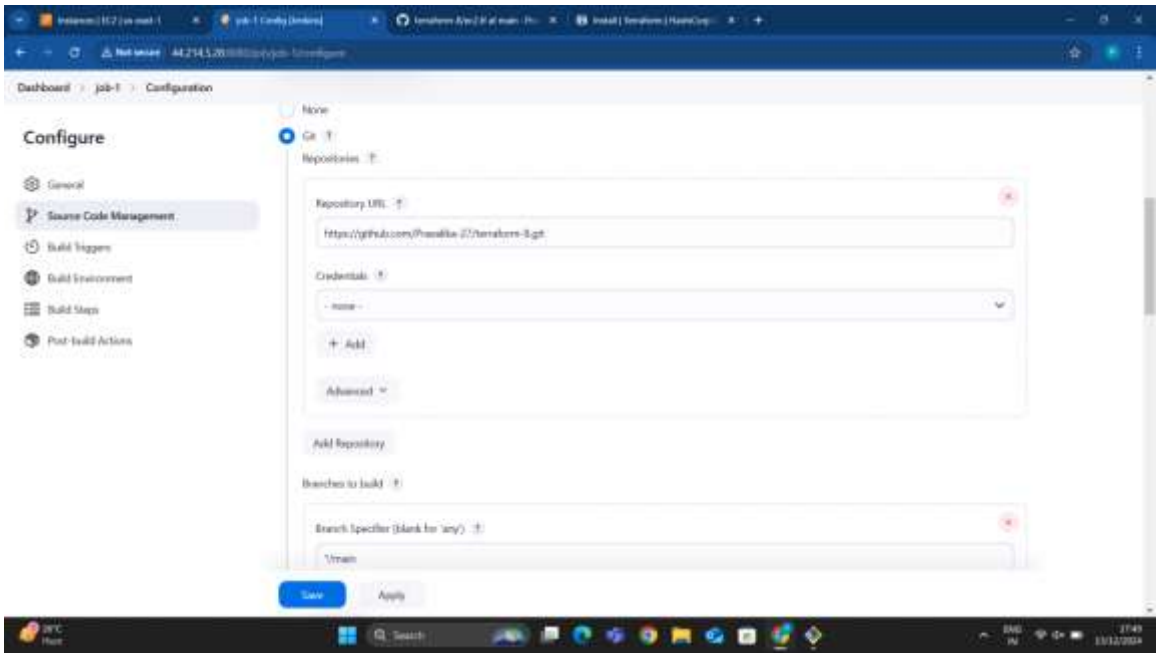
Add Git Credentials (if required):

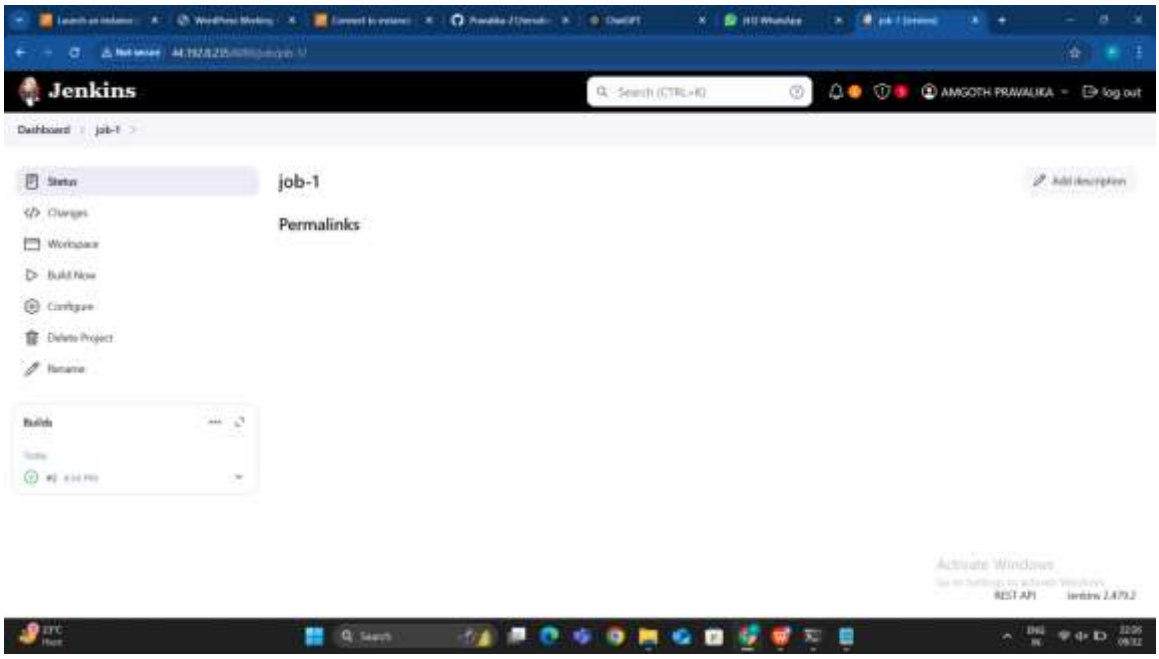
1. If your repository is private, click **Add** next to **Credentials**.
2. Select the appropriate type of credentials:
    1. **Username with password**: Enter your Git username and personal access token/password.
    2. **SSH Username with Private Key**: Paste your private SSH key.
3. Click **Add** to save the credentials.
4. Select the saved credentials from the dropdown menu.

Specify a Branch (Optional):

1. Under **Branches to build**, enter the branch you want to use.
2. Example:
    1. To build the default branch: */main or */master.
    2. To build a specific branch: */feature-branch.

Advanced Options (Optional):

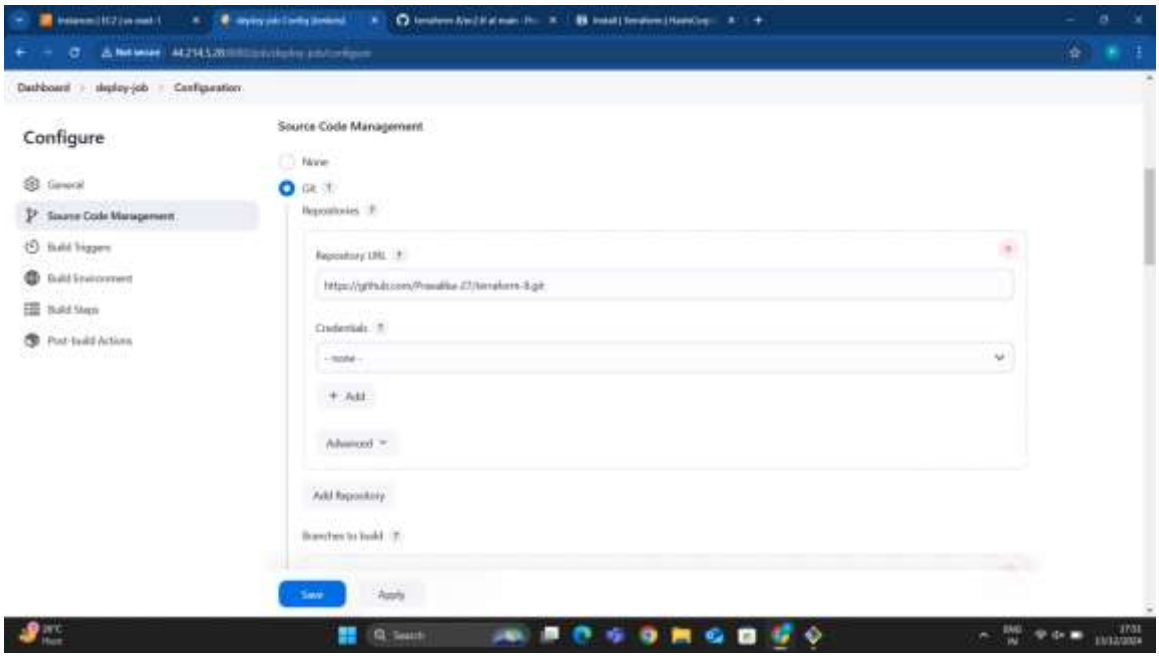




Step 3: Create Deploy Job (job-2)

1. Navigate to Jenkins Dashboard:

- Click **"New Item"** > **"Freestyle Project"**.
- Enter a name: deploy-job and click **OK**.



2. Configure job-2:

General:

- Select **"Delete workspace before build starts"** (optional cleanup step).

Source Code Management:

- Select **Git**.
- Add the **Git URL** for your Terraform or application deployment repository.

Build Triggers:

- Select **"Build after other projects are built"**.
- Specify **job-1** as the dependent project.
- Enable **Trigger only if build is stable**.

Build Environment:

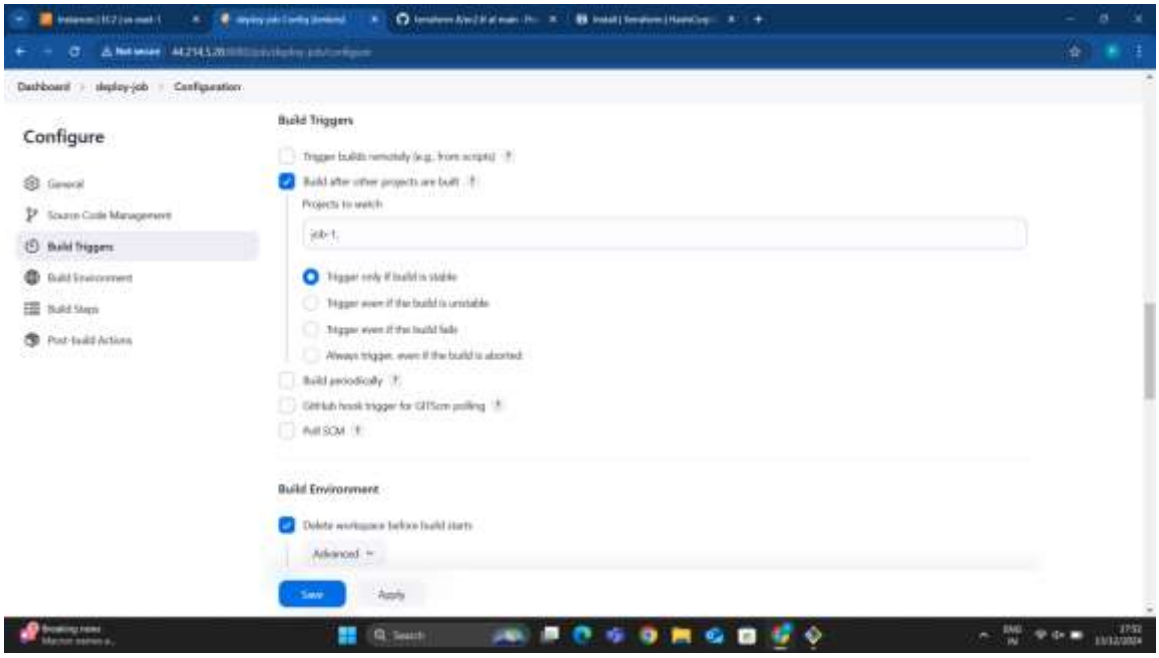
- Select **Use secret text(s) or files**.
- Add your **AWS credentials**.

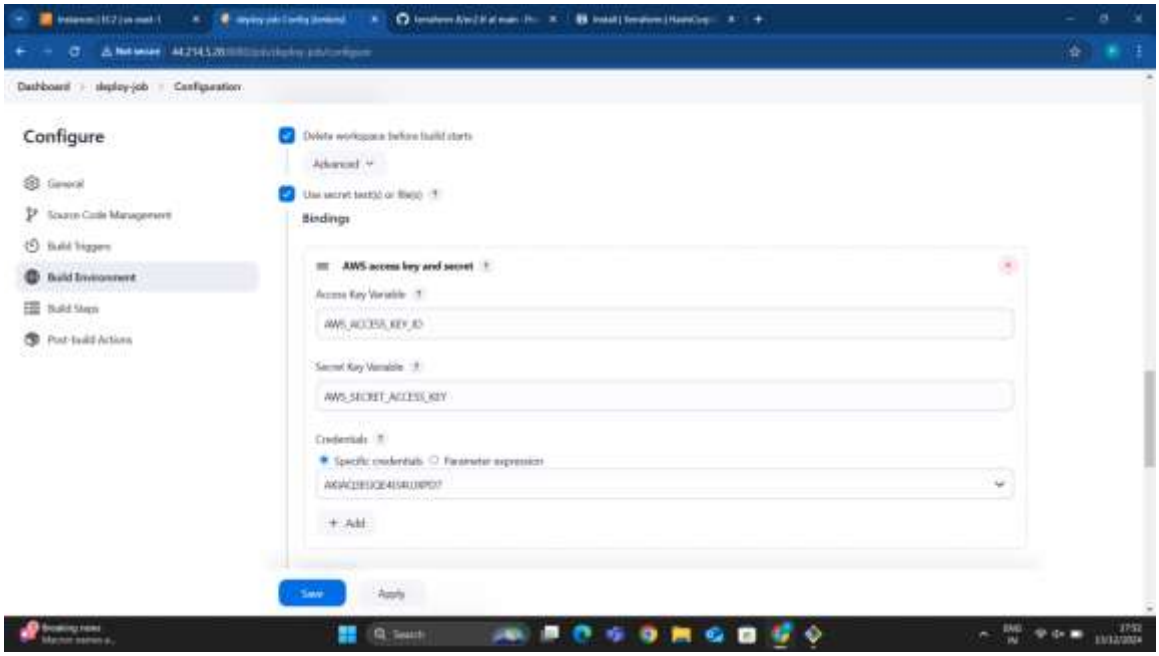
Build Steps:

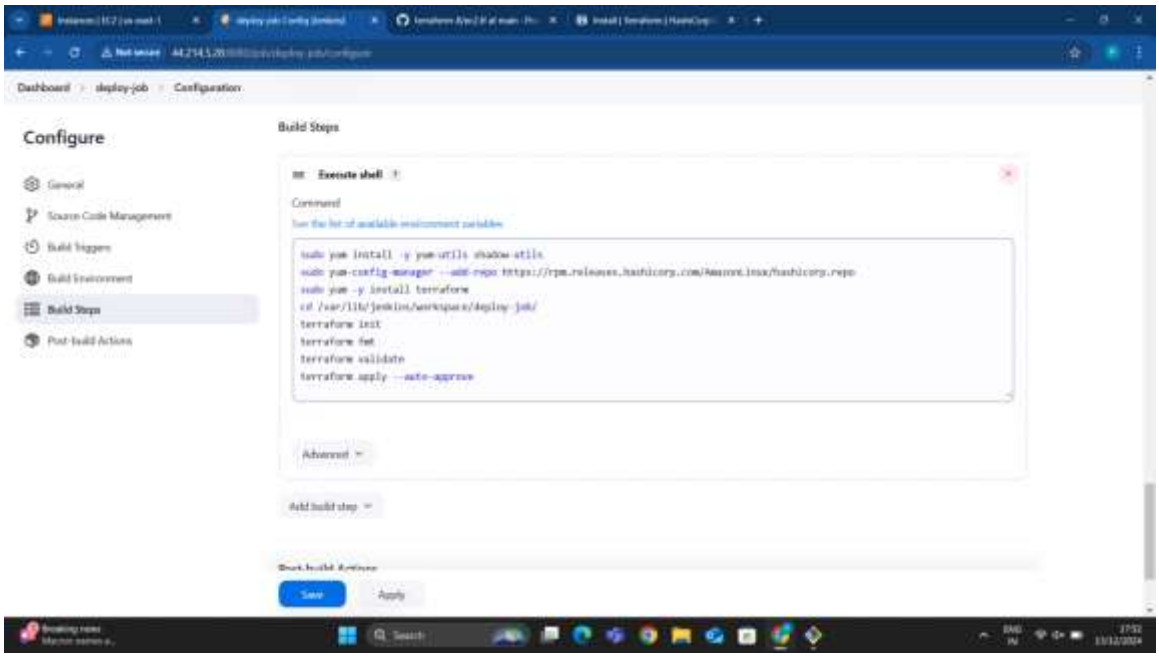
- Add **"Terraform Installation"**:

- Configure Terraform version if required.

- Add **"Execute Shell"**:





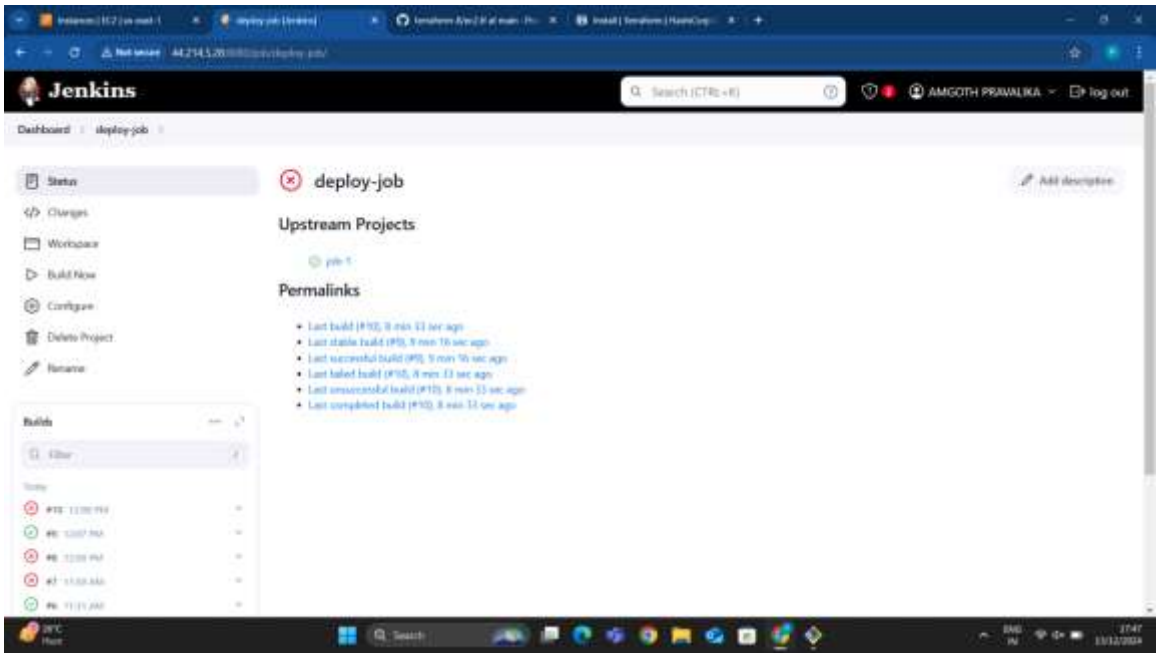


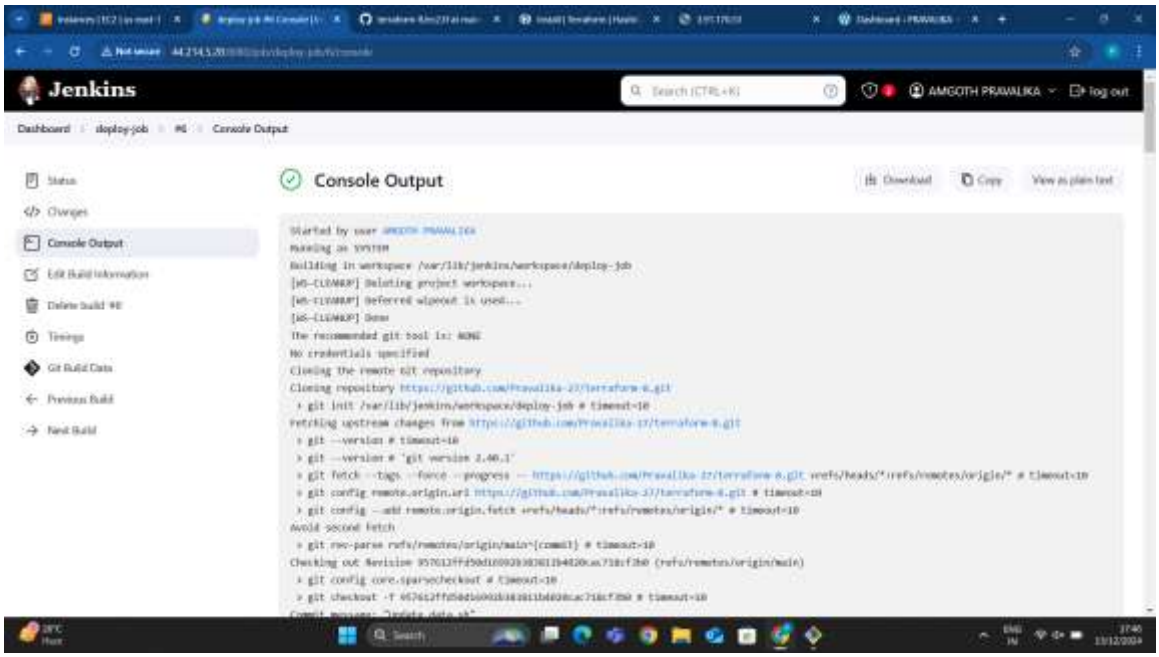
· **Post-Build Actions**:

- · Add **"Build Other Projects"** and specify: job-1 (if circular pipeline setup is required).

· **Save and Build**:

- · Save the job and test by running a build.

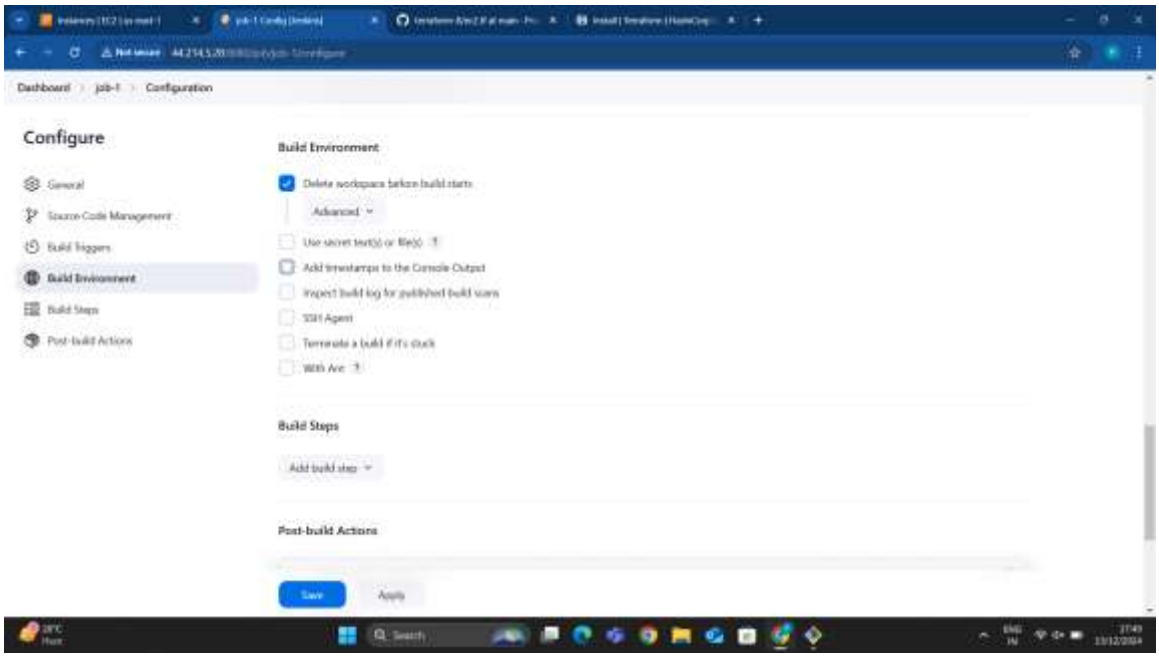


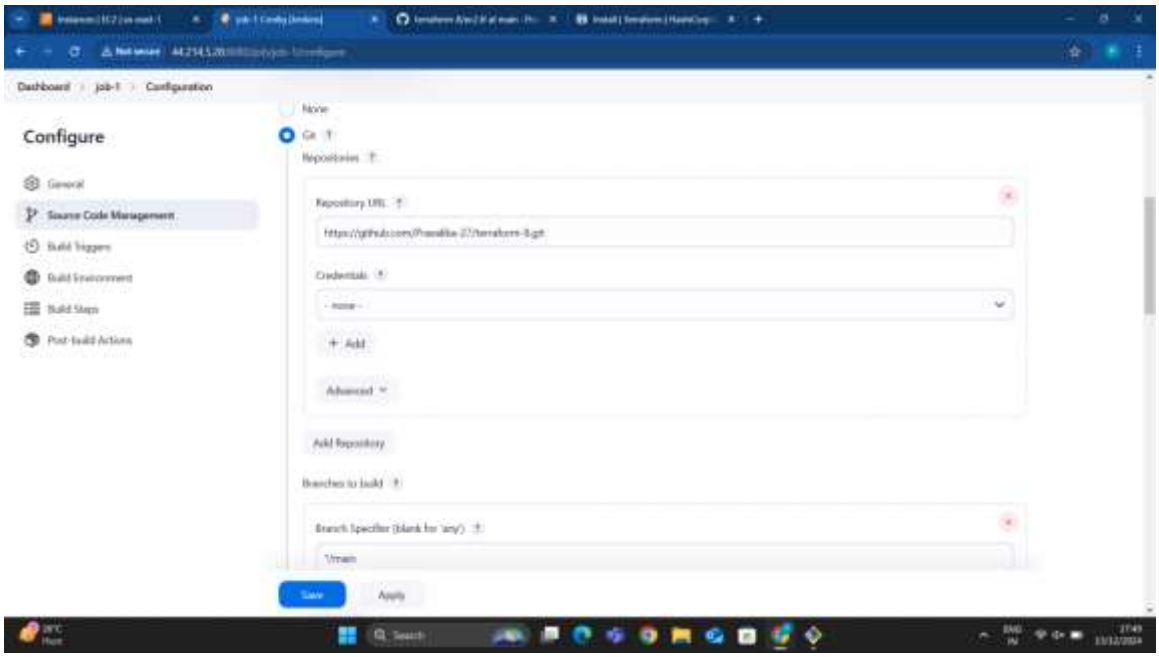


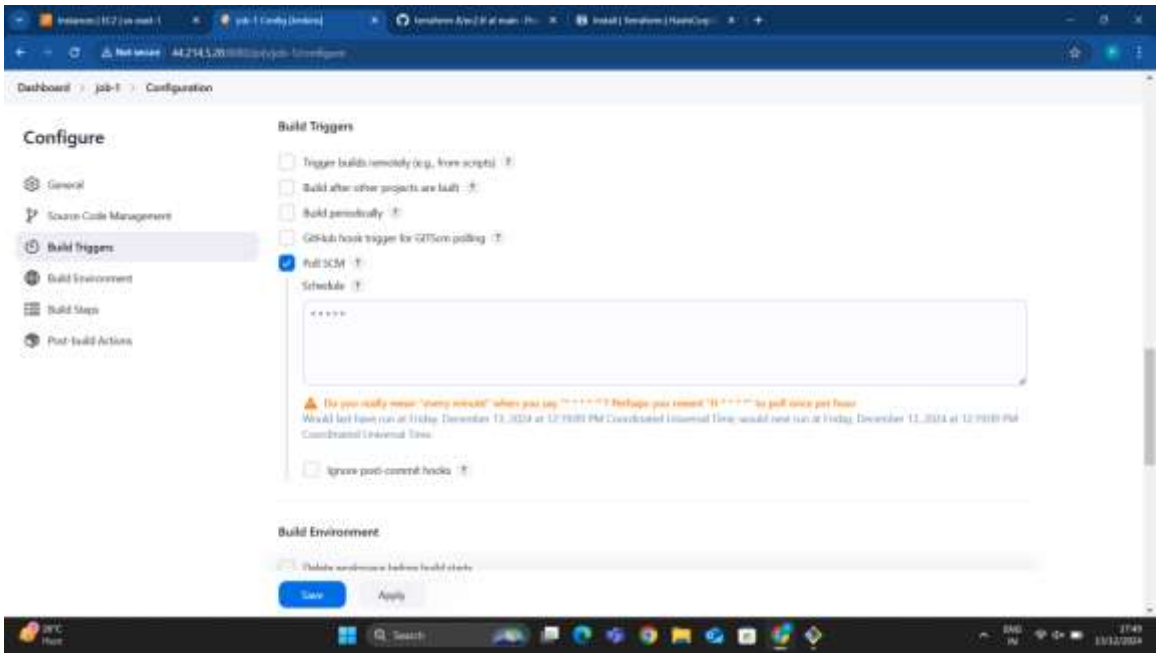
Step 4: Modify Job-1 (Polling and Trigger)

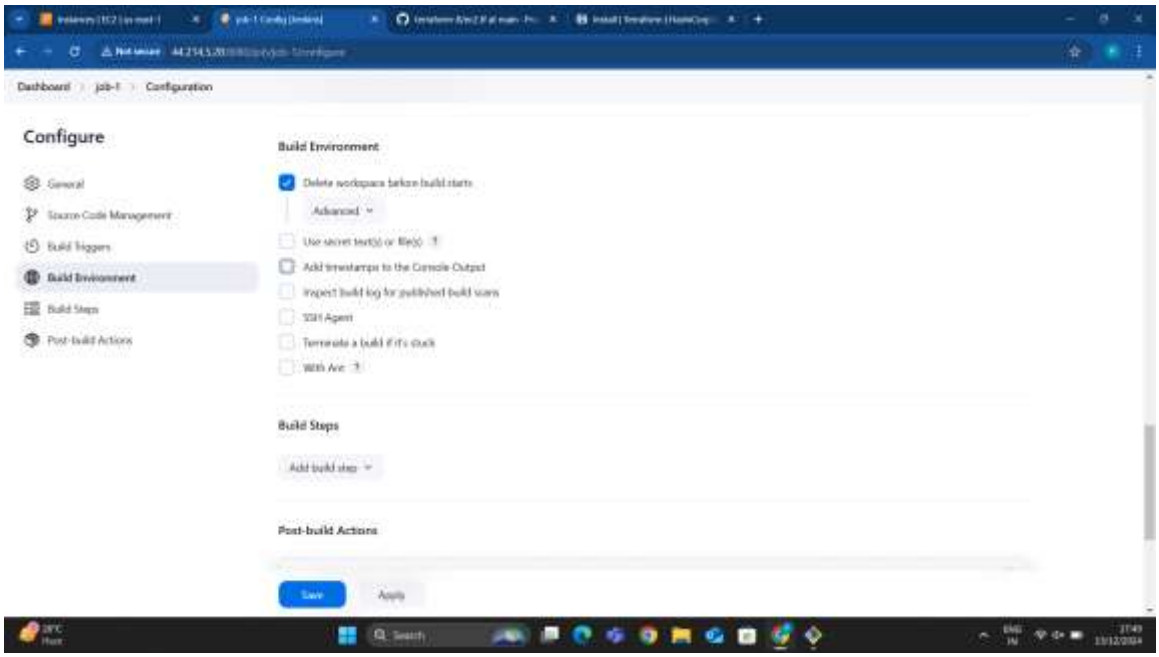
1. Add Poll SCM in job-1:

- Open **job-1** > **Configure**.
- Enable **"Poll SCM"**:
  - Add a polling schedule, e.g., H/5 * * * * for every 5 minutes.


Dashboard
job-1
Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Build Environment
Delete workspace before build starts
Advanced
Use secret text(s) or file(s)
Add timestamps to the Console Output
Inject build log for published build scans
SSH Agent
Terminate a build if it's stuck
With Ansi
Build Steps
Add build step
Post-build Actions
Save
Apply


Dashboard
job-1
Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
None
Git
Repositories
Repository URL
Credentials
- none -
Add
Advanced
Add Repository
Branches to build
Branch Specifier (blank for 'any')
Save
Apply





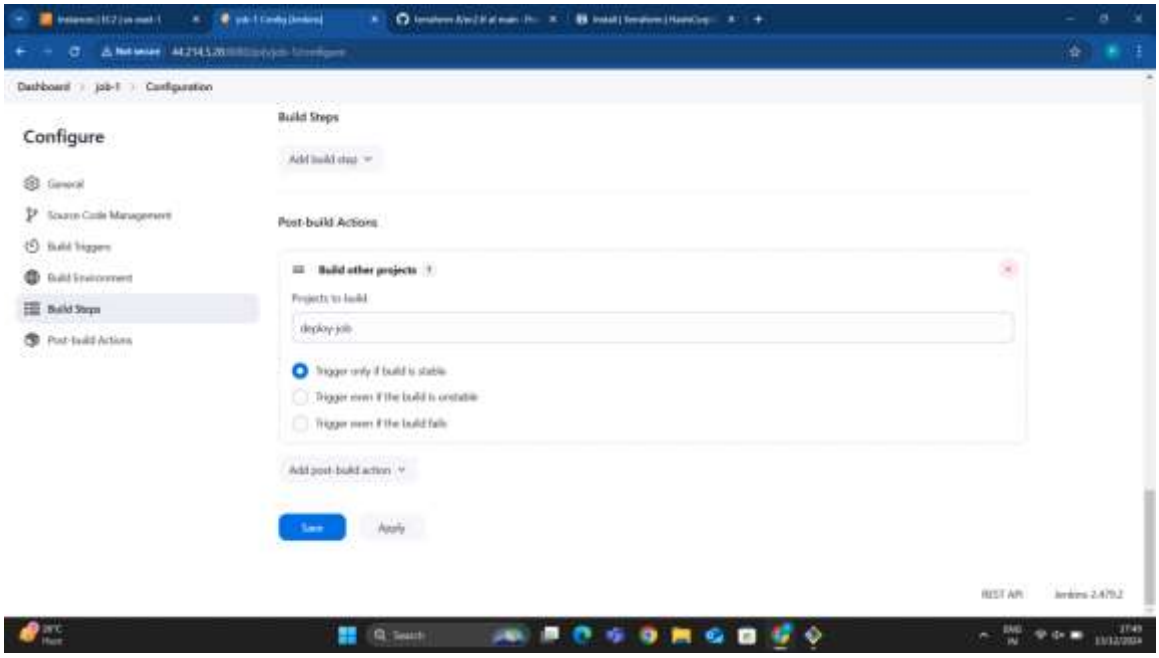
2. Add Post-Build Actions:

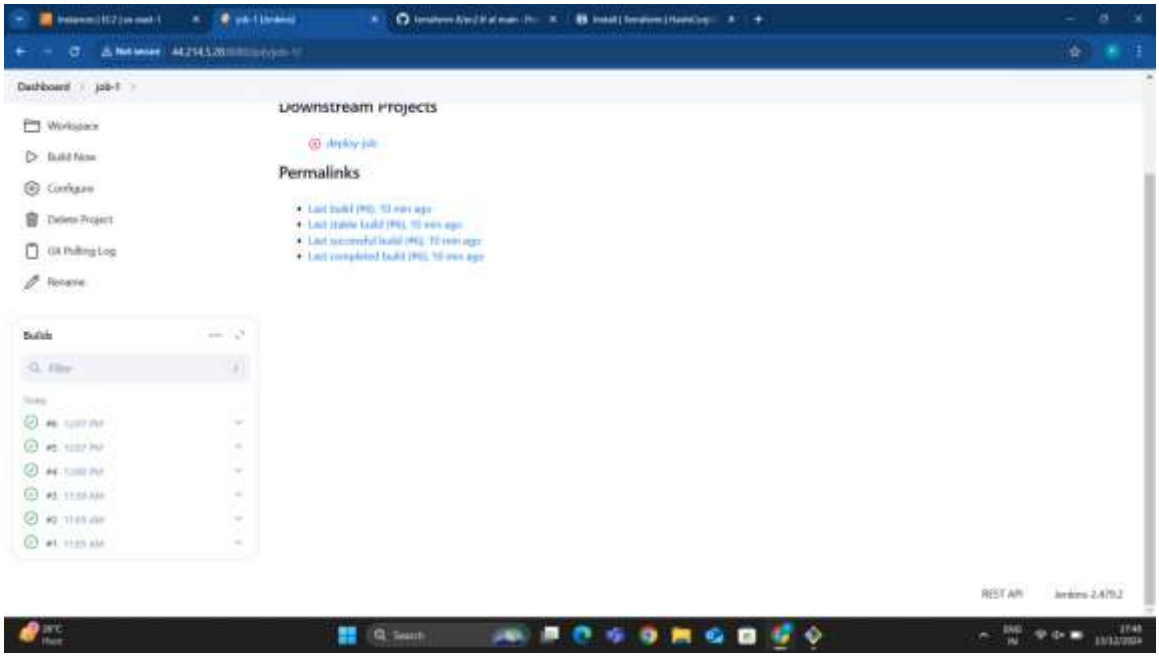
- Add **"Build Other Projects"**.
- Specify the `deploy-job`.

Add "Build Other Projects"

1. From the dropdown menu, select **Build Other Projects**.

2. In the **Projects to build** field, enter the name of the job to be triggered after Job-1 completes. For this scenario:
    1. Enter deploy-job.





5. Save the Configuration

1. After adding the post-build action, click **Save** at the bottom of the page.

6. Verify the Post-Build Action

1. Run **Job-1** by clicking **Build Now**.
2. Once Job-1 completes successfully, check if deploy-job is automatically triggered.
3. You can view the status of deploy-job in the Jenkins dashboard or build history.

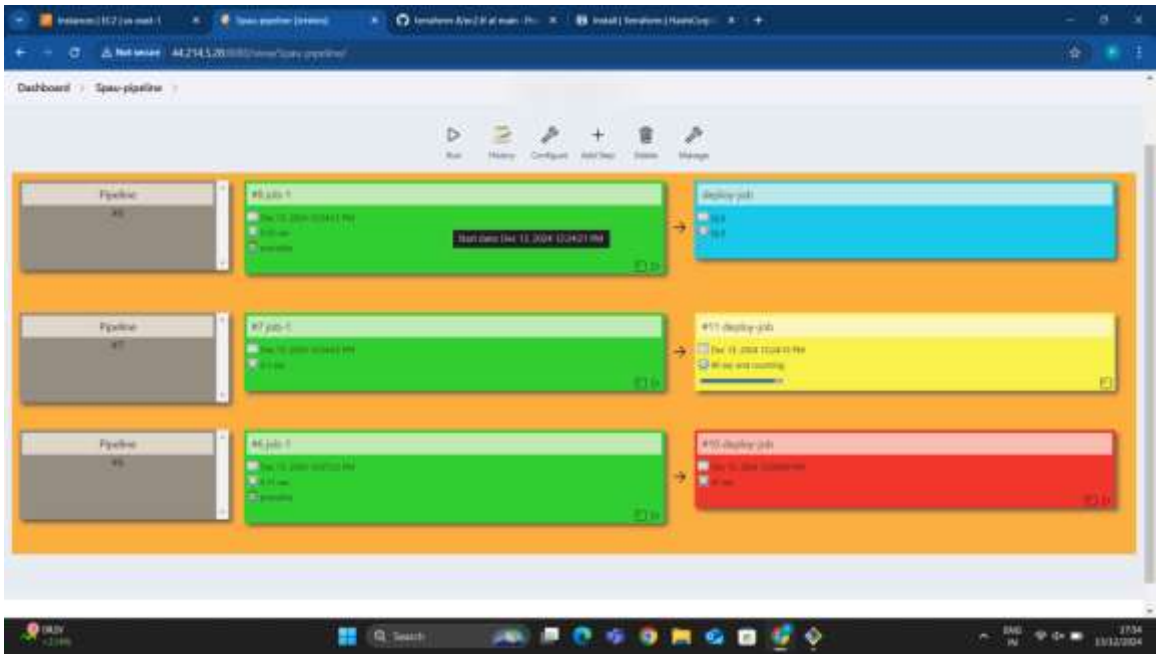
Build and View the Pipeline

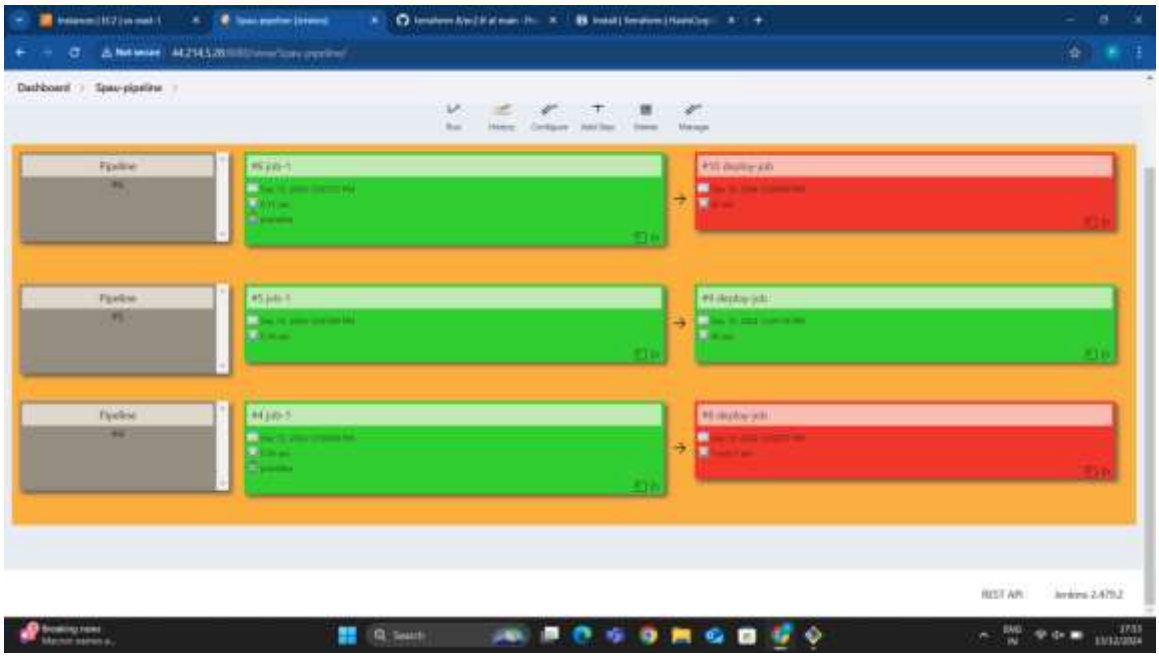
Trigger the Build:

1. Navigate to **Job-1** and click **Build Now**.
2. Ensure the post-build action triggers the `deploy-job`.

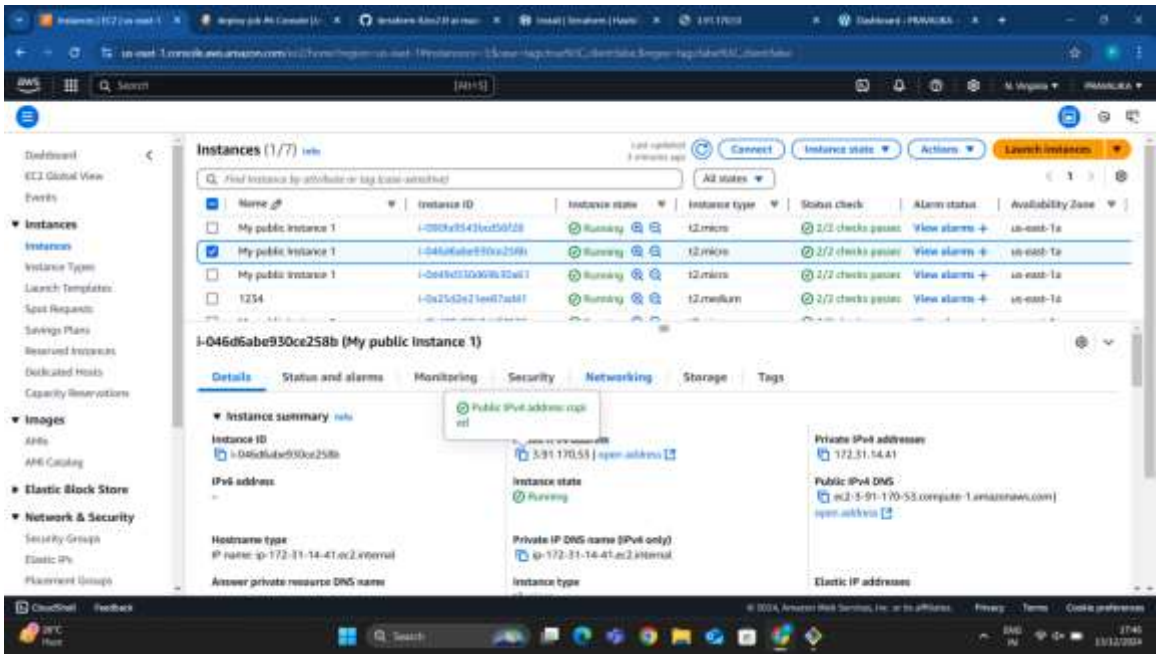
Monitor the Pipeline:

1. Go to the **Pipeline View** you created.
2. Watch the status of `Job-1` and `deploy-job` as they execute.
3. If both jobs succeed, the pipeline will show a completed status.





After successful creation of the jobs new instances are created



Access the Application Using the IP Address

Identify the Application's IP Address:

1. The IP address is usually tied to an EC2 instance or server where your application is deployed.
2. If you used Terraform, retrieve the public IP of the deployed server. Example:

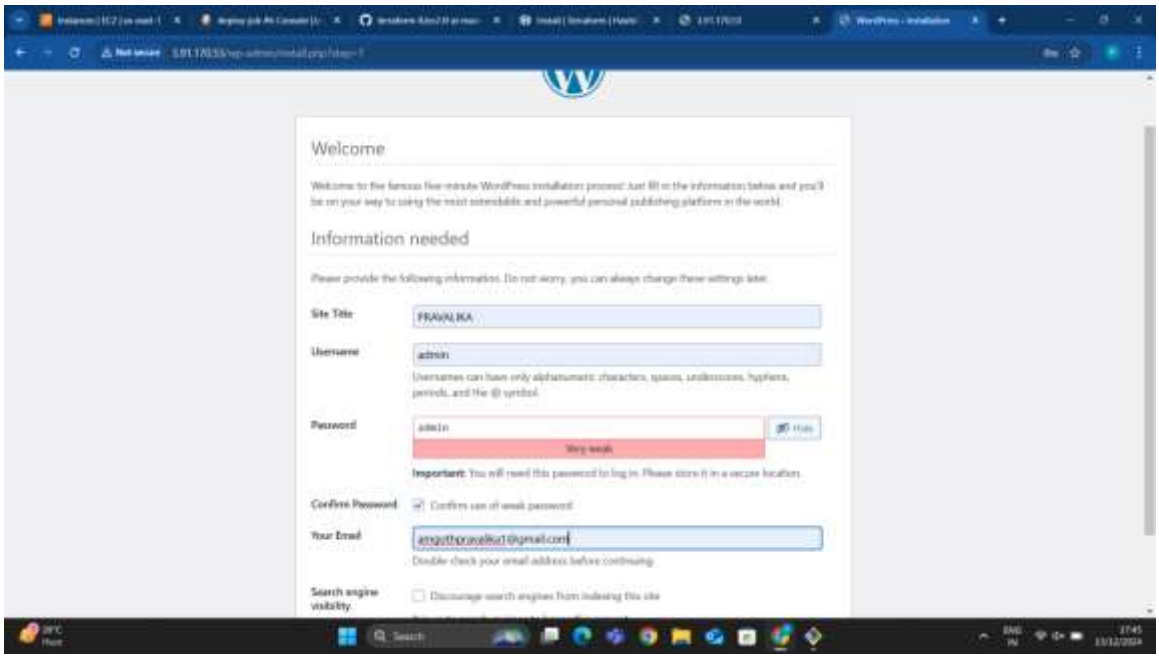
1. Run `terraform output` in your deployment directory.
2. Look for the output variable that provides the public IP address.

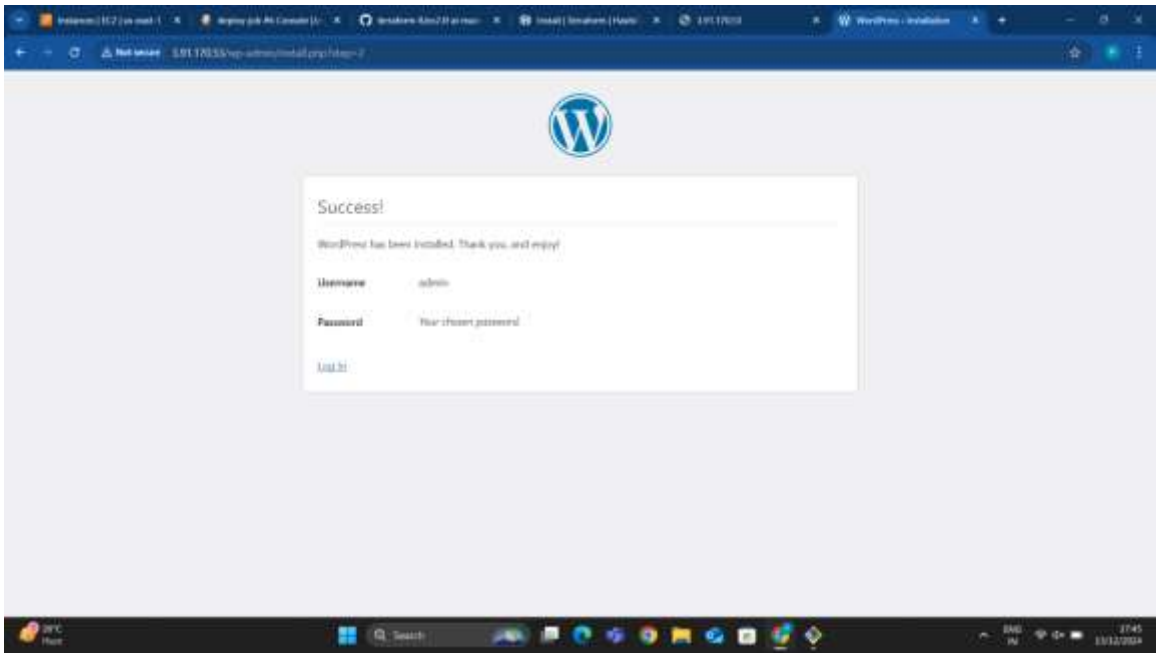
Access the Application:

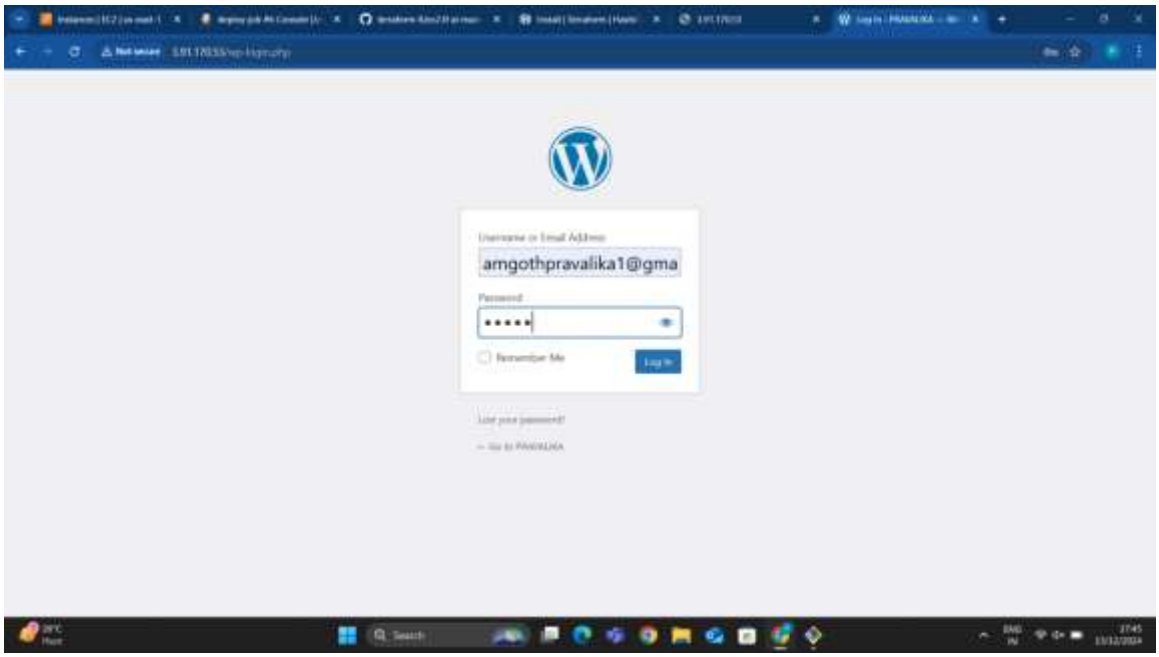
1. Open a browser and enter the public IP address (or public DNS) of the deployed server.
2. If the application runs on a specific port (e.g., port 8080), include it in the URL:
    1. Example: `http://<public-ip>:8080`

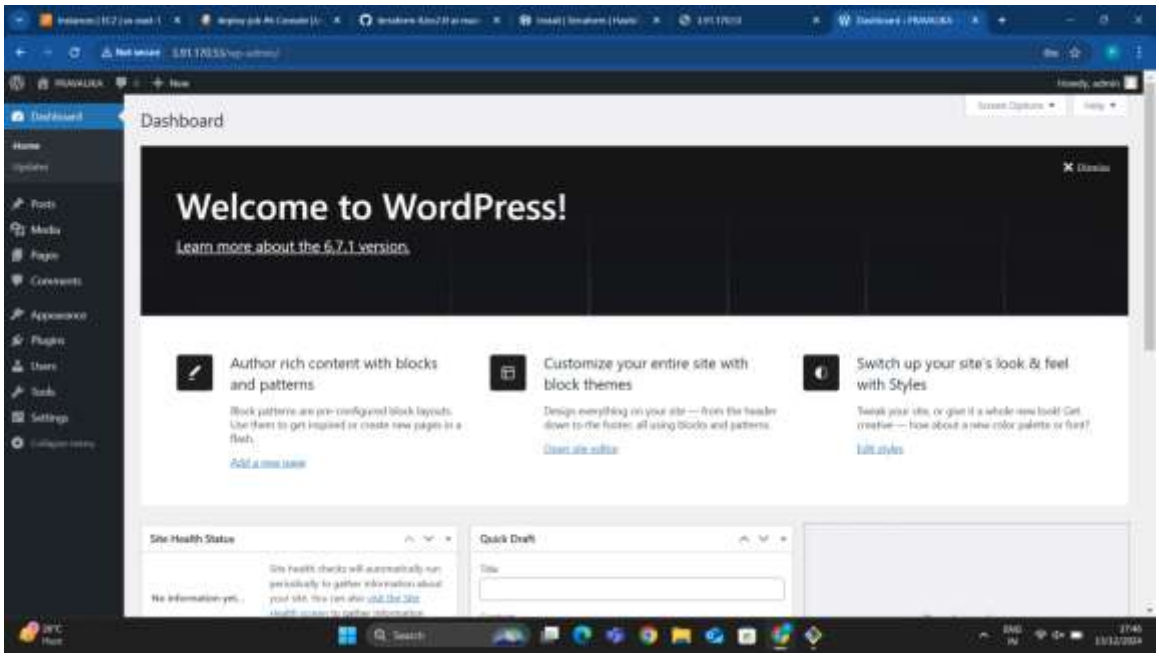
Verify Application Access:

1. Confirm that the application is accessible and functioning as expected.




Success!
Username
Password
Log In

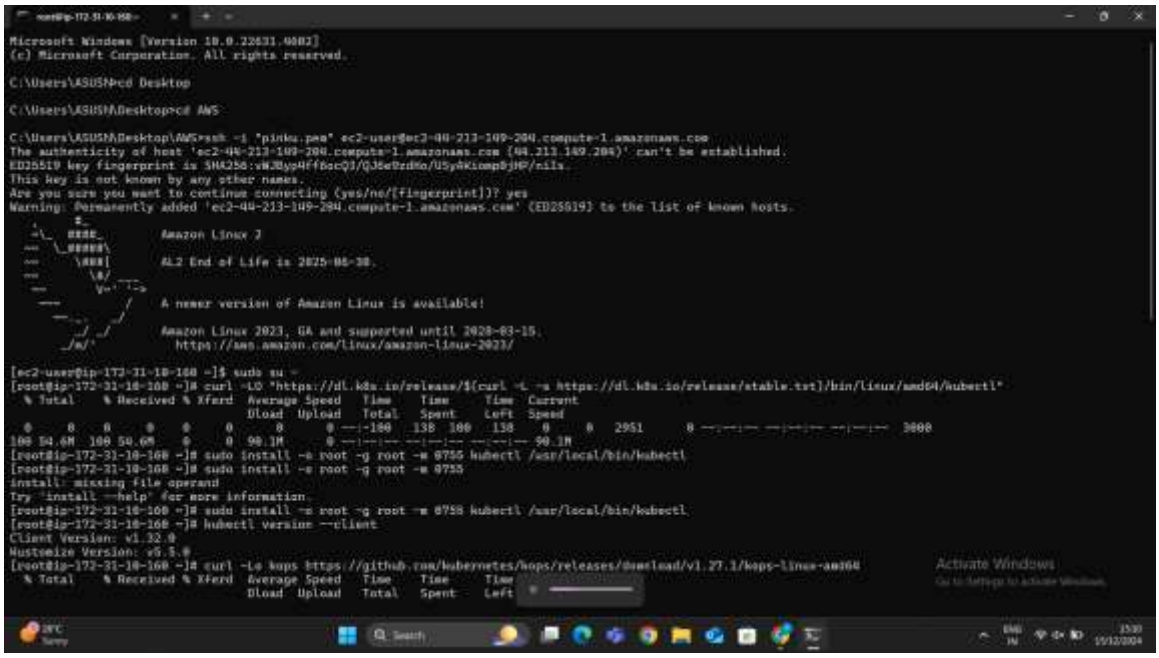

amgothpravalika1@gma
Log In



METHOD-10:Kubernetes (Declarative manifest method) with the help of docker hub images

Step 1: Create an EC2 Instance

1. **Log in to your AWS Management Console.**
2. Navigate to the **EC2 service**.
3. **Launch a New Instance:**
    1. Choose an **Amazon Linux 2 AMI** or **Ubuntu 20.04** for compatibility.
    2. Select an instance type (e.g., `t2.micro` for low-cost testing).
    3. Configure key pair for secure access or use an existing one.
    4. Assign the instance to a public subnet in your default VPC.
    5. Allow necessary inbound rules in the security group:
        1. SSH (Port 22) for instance access.
        2. HTTP (Port 80) and NodePort range (e.g., 30000–32767) for Kubernetes applications.



Set Up AWS S3 Bucket for KOPS State Store

Log in to your AWS account.

Open the **S3 service** on the AWS Management Console.

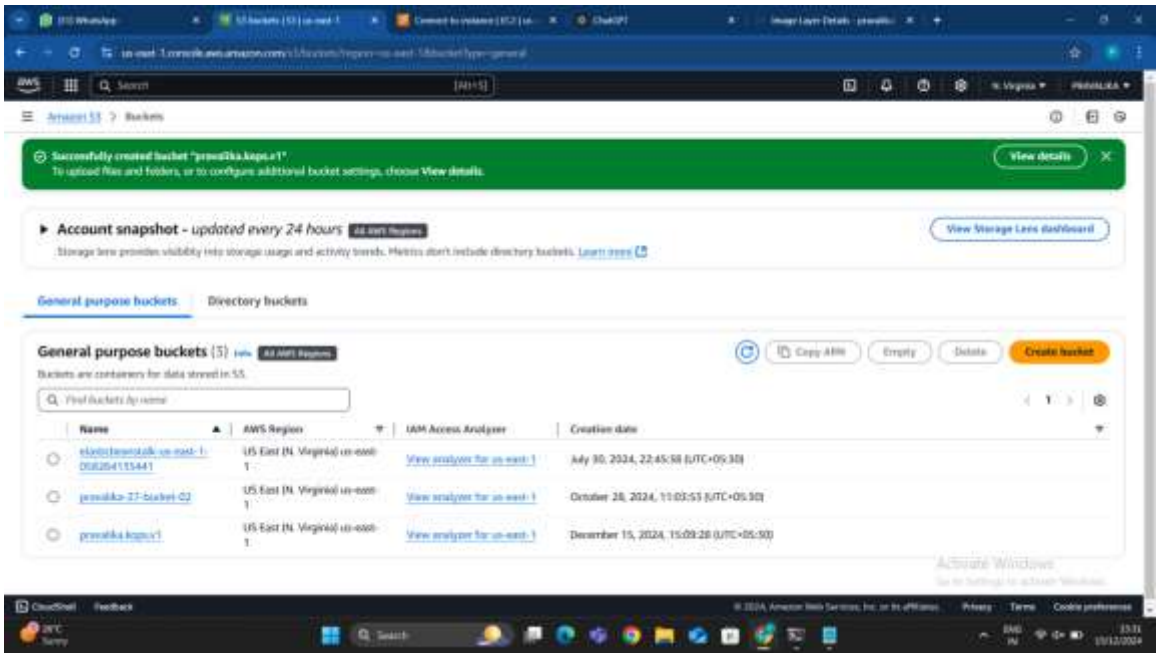
Create an S3 Bucket:

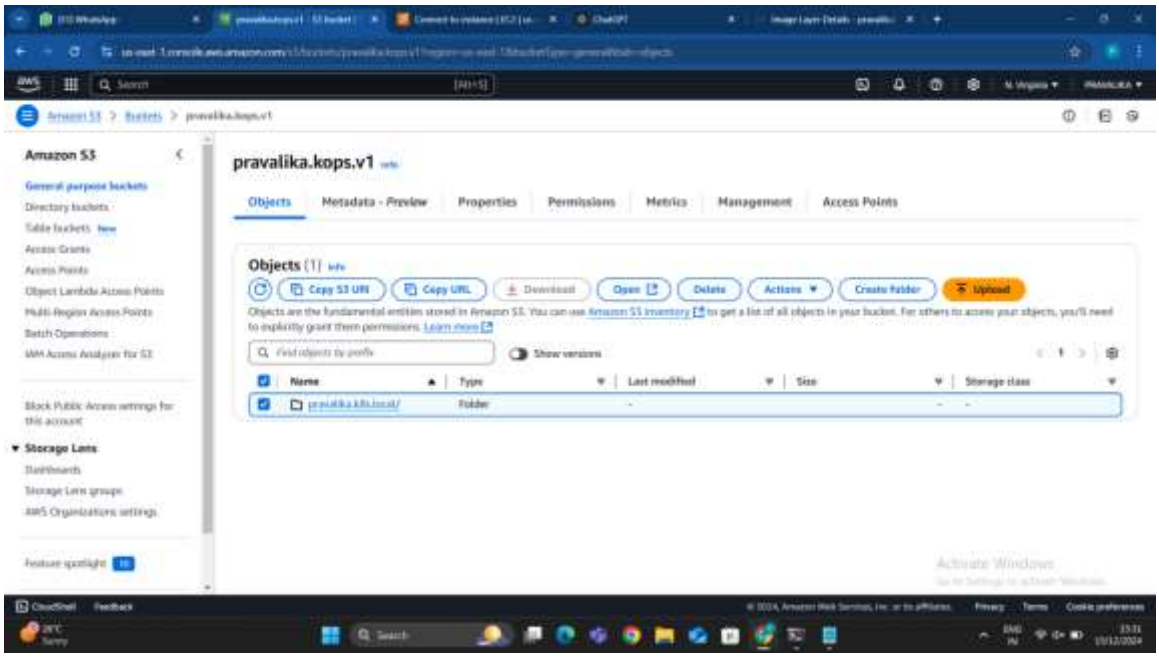
1. Click on **Create Bucket**.
2. Enter a unique **Bucket Name** (e.g., `bhargavi.kops.v1`).
3. Select the region (e.g., `us-east-1`).
4. Leave other settings as default and click **Create**.

Once the bucket is created, note the bucket name for later use

Export the S3 Bucket Name as the State Store Environment Variable:

export KOPS_STATE_STORE=s3://pravalika.kops.v1





Step 2: SSH Key Generation

- • This command generates a pair of keys: a public key (id_rsa.pub) and a private key (id_rsa).

· **Specify the Key Location**:

- · When prompted, specify a path where you want to save the key (e.g., ~/.ssh/id_rsa).

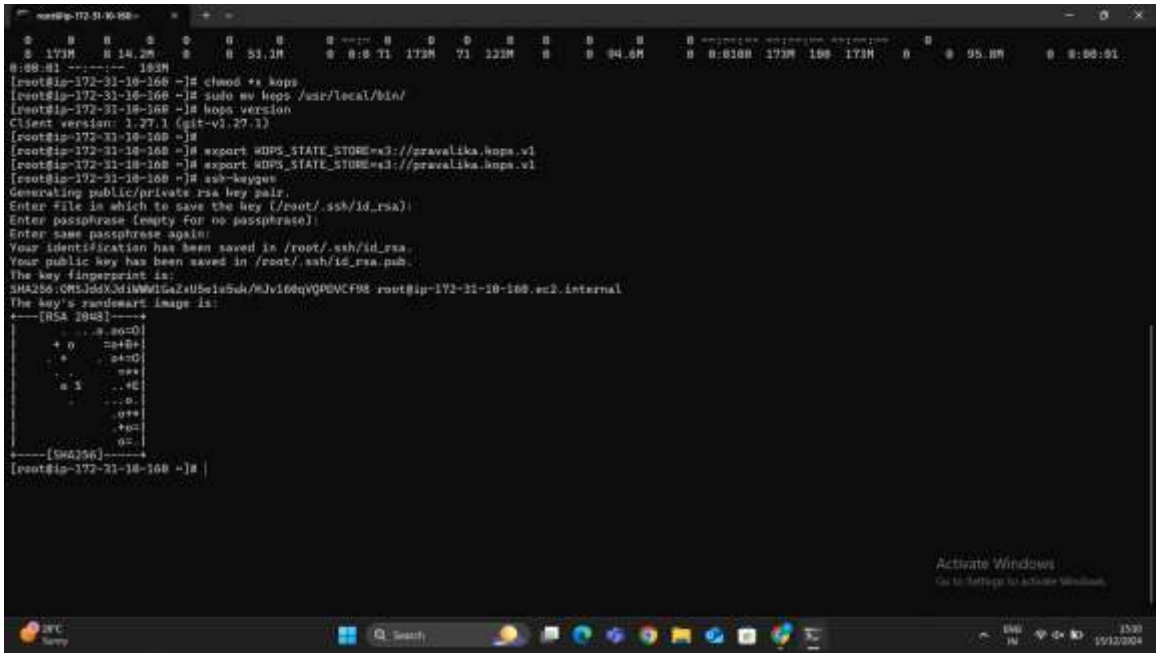
- Press **Enter** to use the default location.

· **Set a Passphrase (Optional)**:

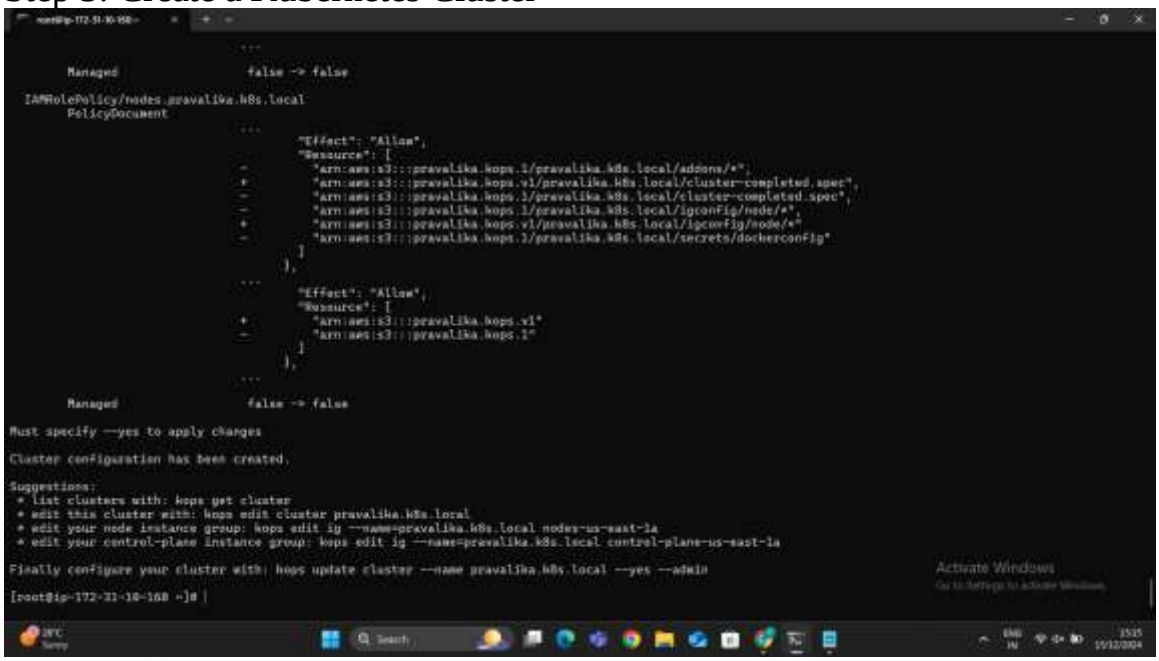
- · You can add an extra layer of security by setting a passphrase.
- If you prefer no passphrase, press **Enter** to skip.

· **Verify the Keys**:

- · Check that the key pair has been generated:ls ~/.ssh/

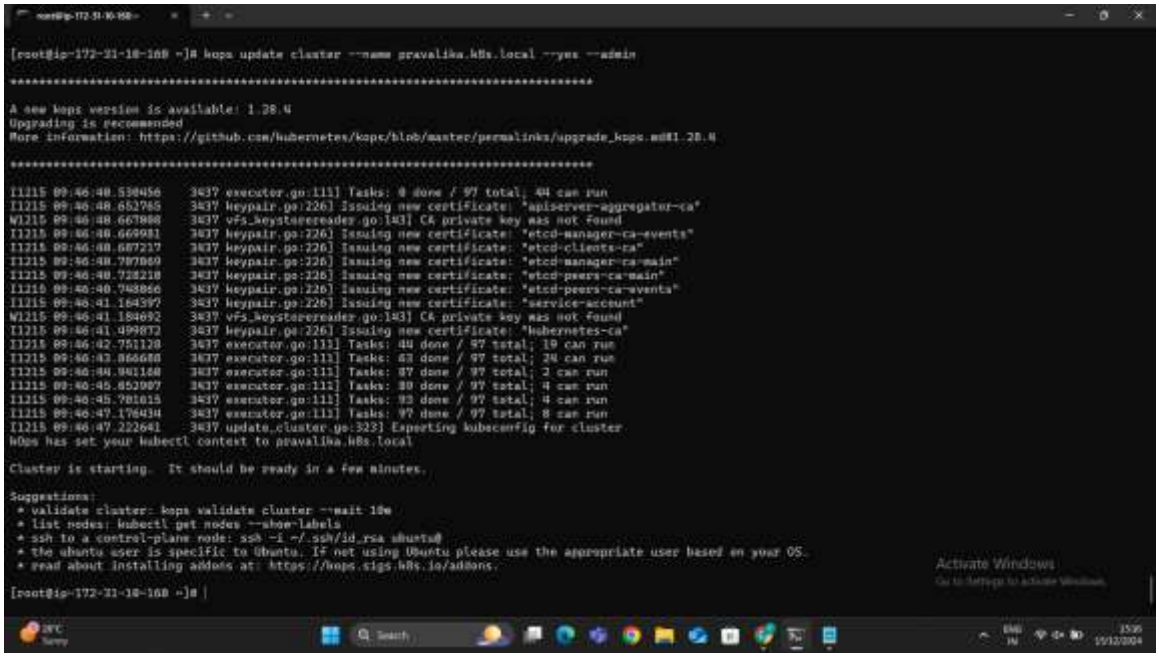


Step 3: Create a Kubernetes Cluster



Step 3: Install KOPS and Kubectl

KOPS (Kubernetes Operations) and Kubectl are essential tools for managing Kubernetes clusters. This step involves setting up these tools to interact with the Kubernetes infrastructure and manage the cluster lifecycle.



What is KOPS?

KOPS is a command-line tool that simplifies creating, configuring, upgrading, and maintaining Kubernetes clusters. It automates tasks such as provisioning infrastructure, deploying Kubernetes components, and managing cluster updates. KOPS is widely used for production-ready cluster setups on cloud providers like AWS.

What is Kubectl?

Kubectl is the command-line interface (CLI) for Kubernetes. It allows users to interact with Kubernetes clusters by running commands to deploy applications, inspect resources, troubleshoot issues, and manage cluster components.

Why Install KOPS and Kubectl?

1. **Cluster Management**:
    1. KOPS simplifies the deployment and configuration of Kubernetes clusters.
    2. Kubectl provides the ability to control and monitor the cluster once it's running.
2. **Automation**:

    1. These tools automate complex tasks, reducing manual errors and improving efficiency.
3. **Integration**:

*Both tools integrate seamlessly with cloud platforms like AWS, making Kubernetes setup straightforward.**Key Benefits of Using KOPS and Kubectl***

- **Efficiency**: Both tools reduce the effort required to set up and manage clusters.
- **Standardization**: They follow industry best practices for Kubernetes deployments.
- **Flexibility**: Allow you to customize and scale clusters easily.
- **Troubleshooting**: Provide robust tools to inspect, debug, and manage workloads.

Install Kubectl

Download the Latest Version of Kubectl:

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

Install Kubectl:

Move the binary to /usr/local/bin/ and set the required permissions

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

Verify the Installation:

kubectl version --client

install KOPS

Download the Latest Version of KOPS:

curl -Lo kops https://github.com/kubernetes/kops/releases/download/v1.27.1/kops-linux-amd64

Make the Binary Executable:

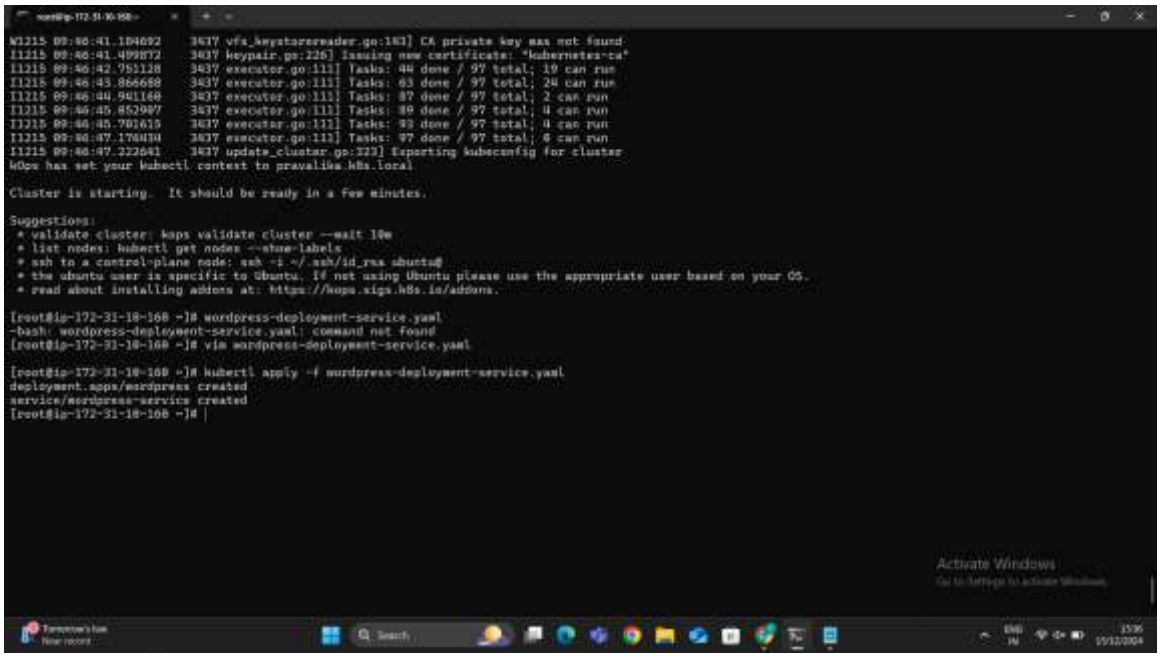
chmod +x kops

Move KOPS to /usr/local/bin/:

sudo mv kops /usr/local/bin/

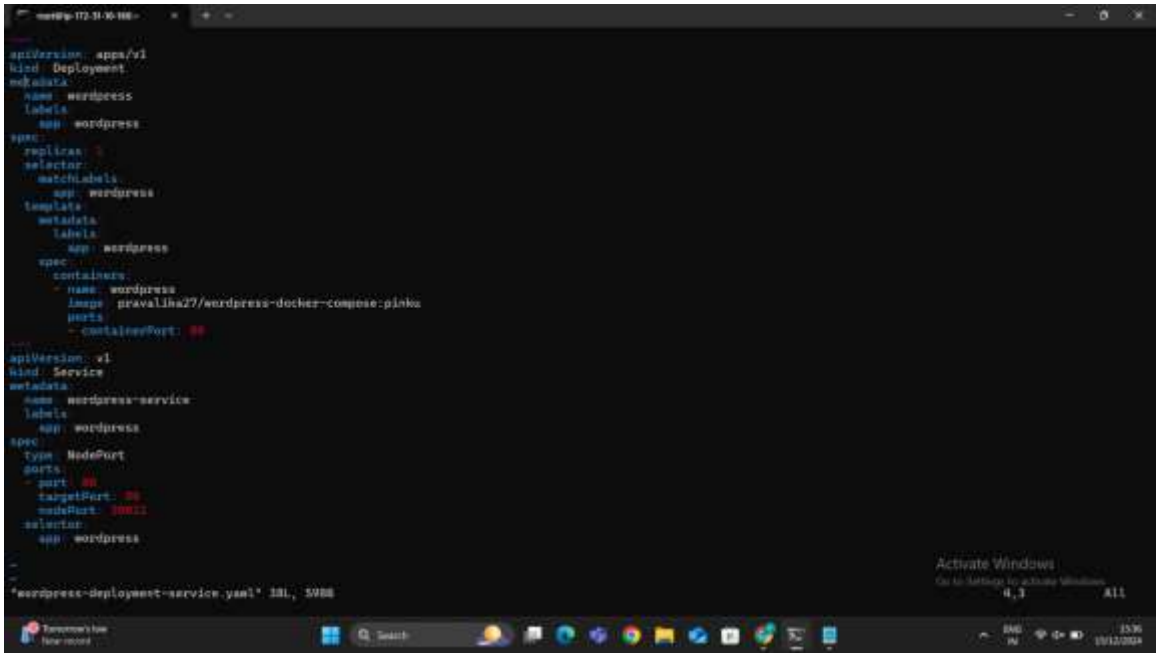
Verify the Installation:

kops version



Step 5: Write the Kubernetes Manifest File

Create a Deployment and Service YAML File: Use a text editor to create the file wordpress-deployment.yaml with the following content:



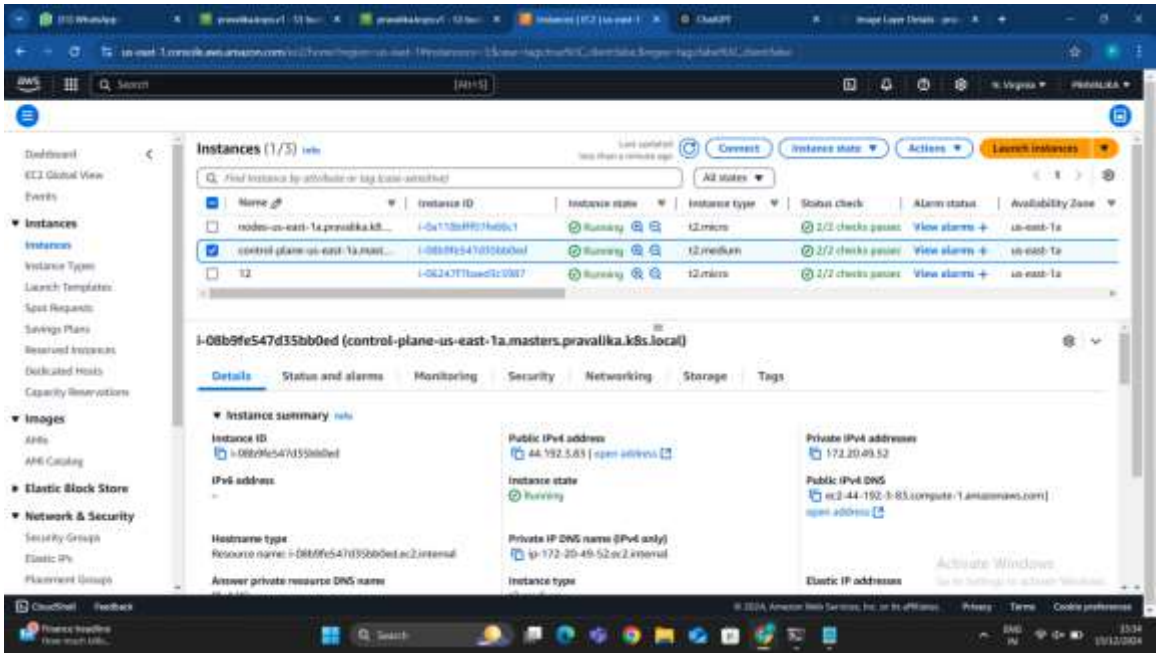
Apply the Manifest File:

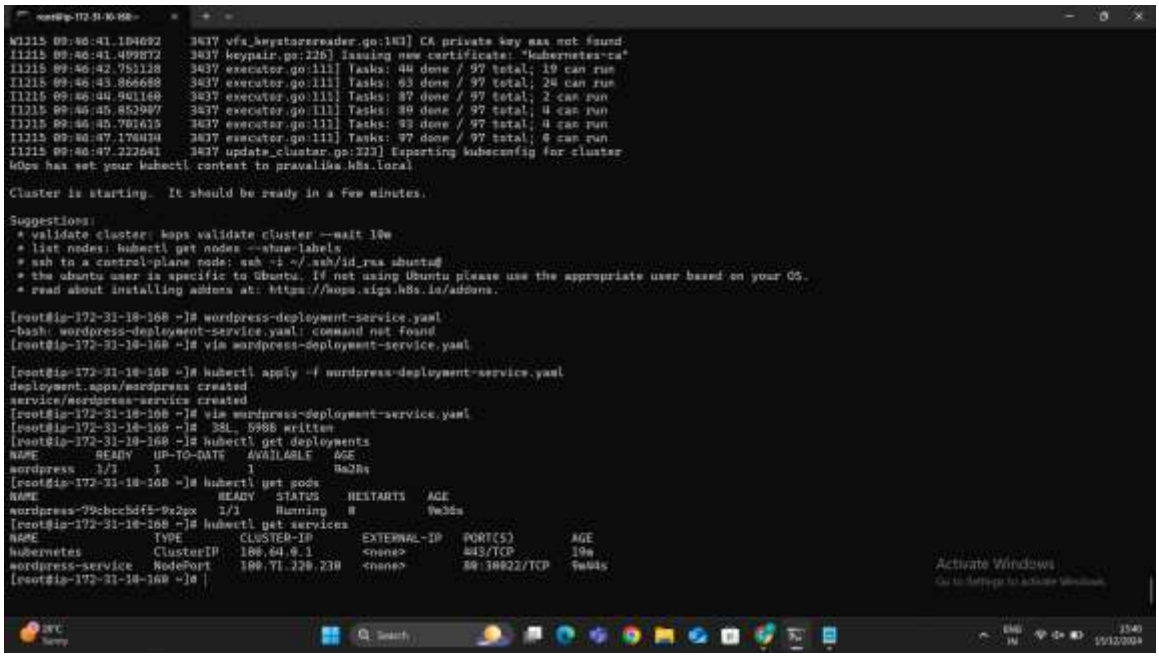
kubectl apply -f wordpress-deployment.yaml

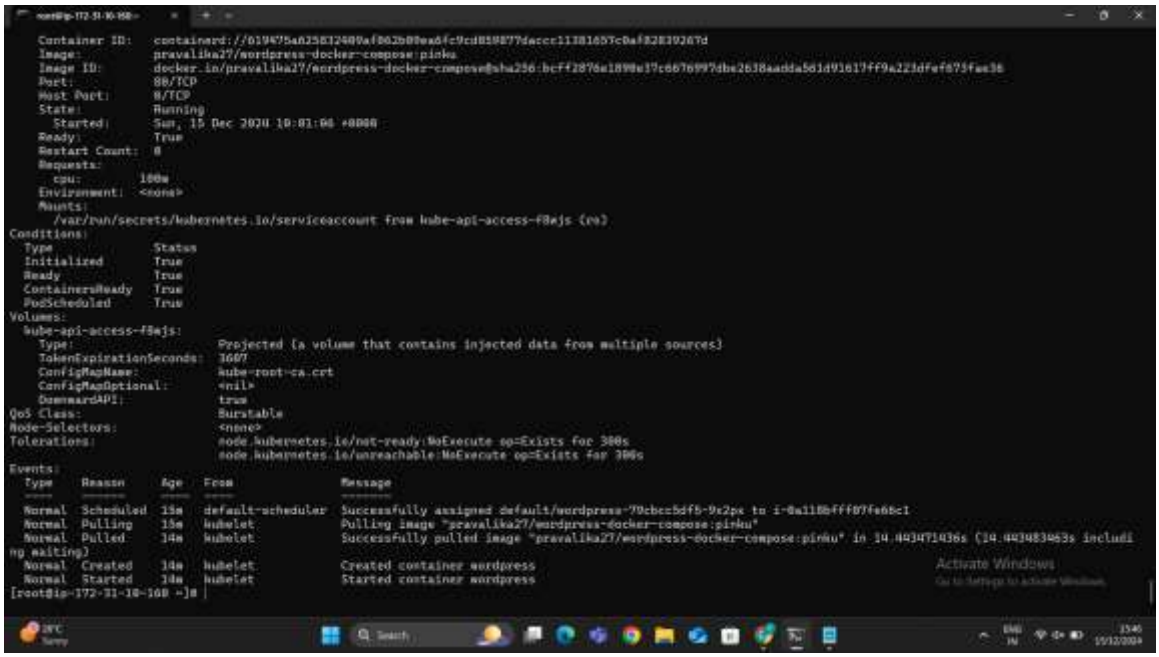
Verify Deployment:

kubectl get pods

kubectl get services







Step 6: Access the Application

Get the Public IP Address of the Instance:

1. You can retrieve the IP from the AWS Console or using the EC2 instance's metadata.

Access the WordPress Application:

1. Combine the instance's public IP and the NodePort (e.g., `30022`)

Verify the WordPress UI:

1. Open the URL in your browser to confirm that the WordPress application is running.

Step 7: Clean Up Resources

Delete the Cluster: When you're done, delete the Kubernetes cluster to avoid unnecessary charges:

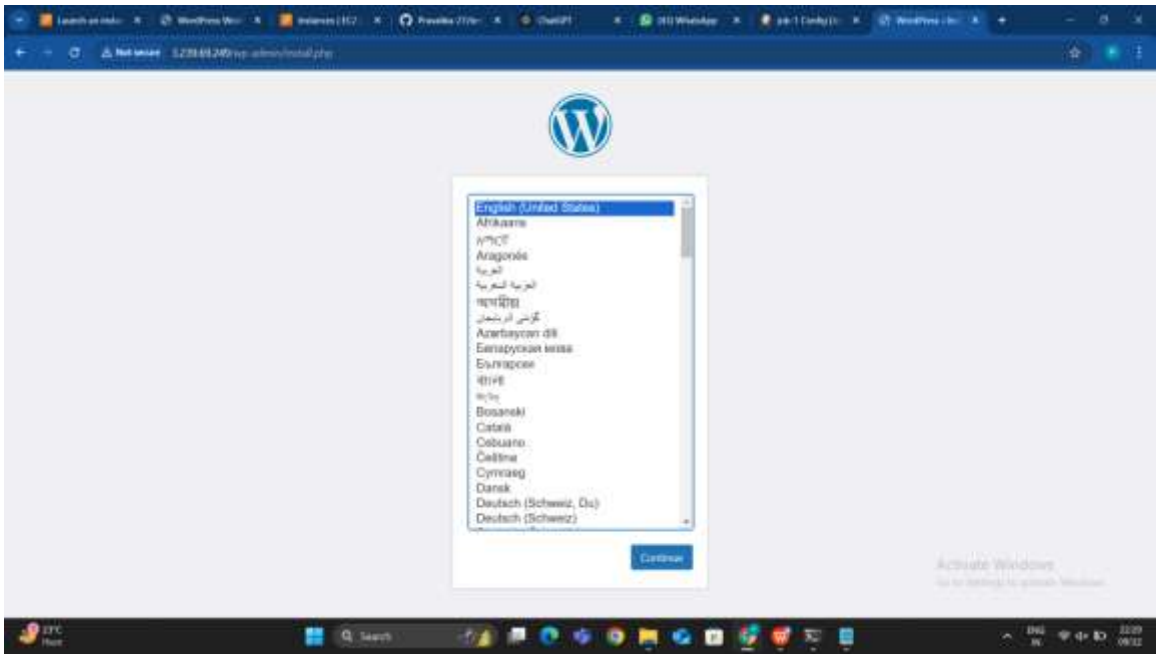
kops delete cluster pravalika.k8s.local --yes

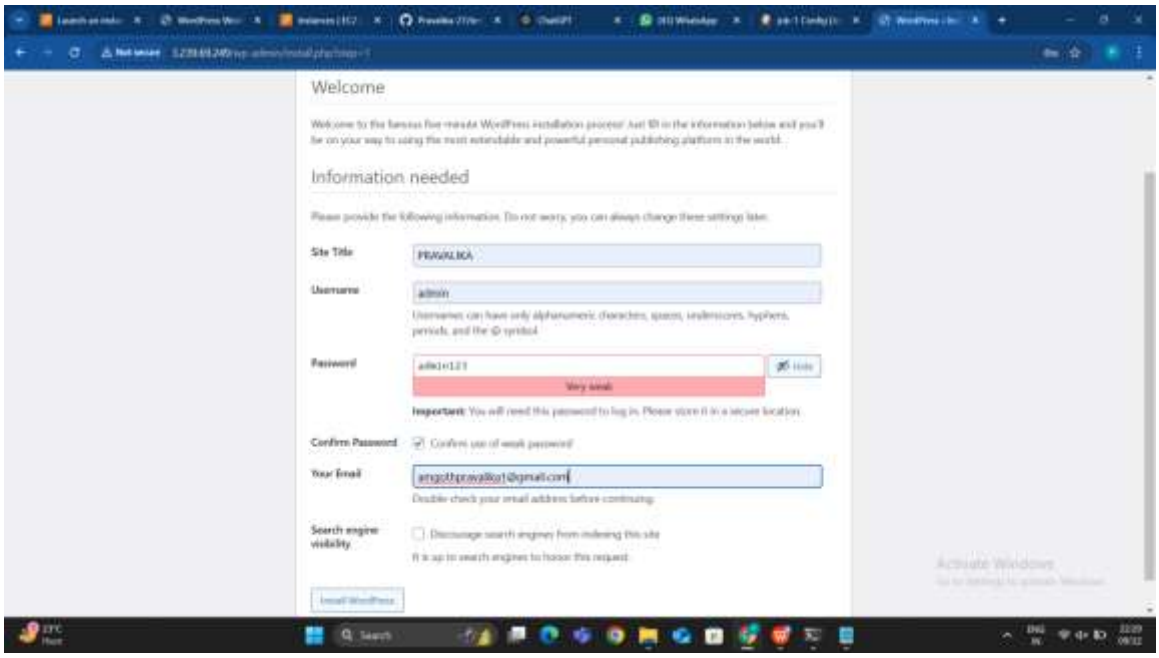
· **Delete the S3 Bucket:**

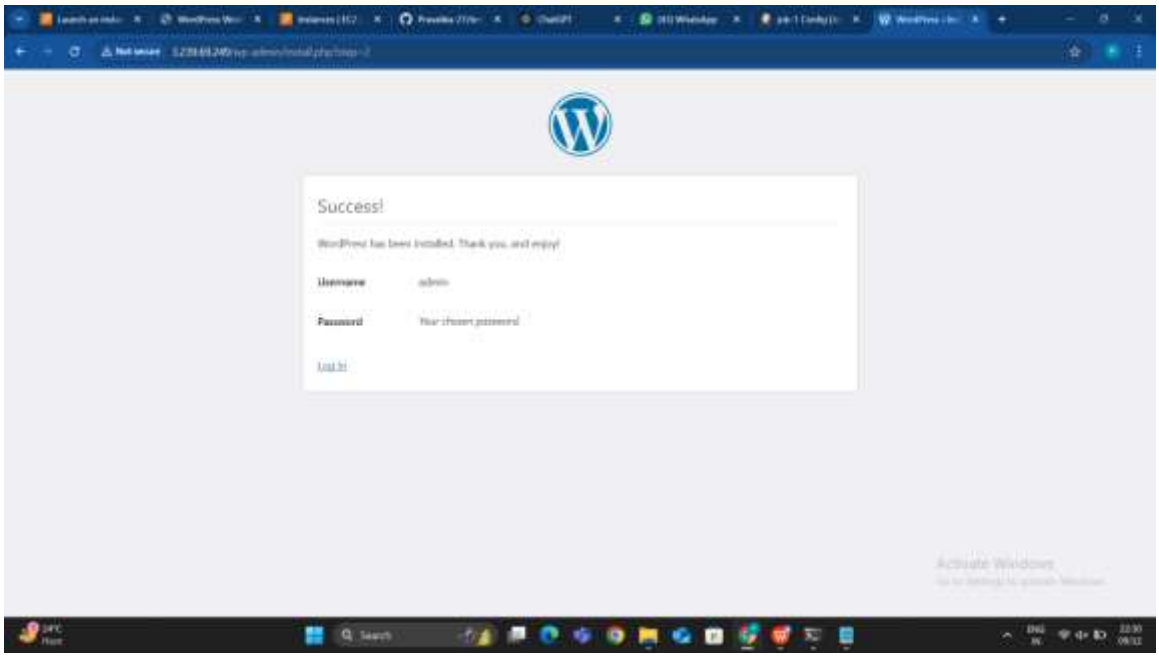
- · Go to the AWS S3 Console and delete the bucket bhargavi.kops.v1.

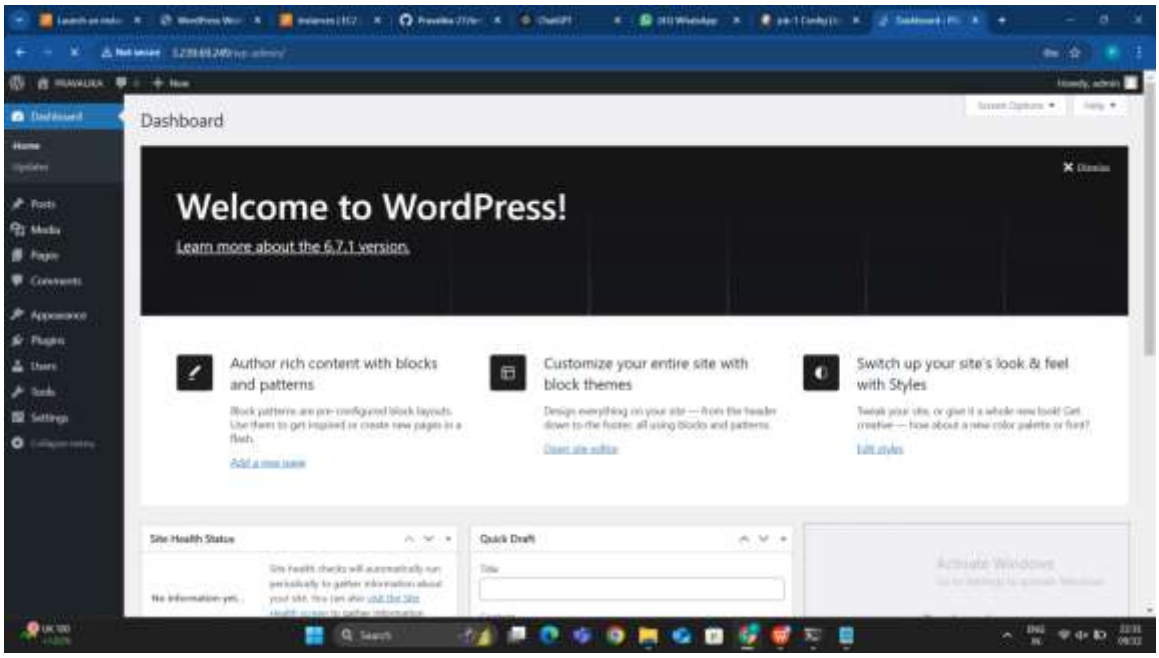
· **Terminate the EC2 Instance:**

- · In the AWS EC2 Console, stop and terminate the instance.




Welcome
Information needed
Site Title
PRAVALIKA
Username
admin
Password
Very weak
Important: You will need this password to log in. Please store it in a secure location.
Confirm Password
Your Email
Search engine visibility
Activate Windows


Success!
Username
Password
Activate Windows


Dashboard
Welcome to WordPress!
Learn more about the 6.7.1 version.
Author rich content with blocks and patterns
Customize your entire site with block themes
Switch up your site's look & feel with Styles
Site Health Status
Quick Draft