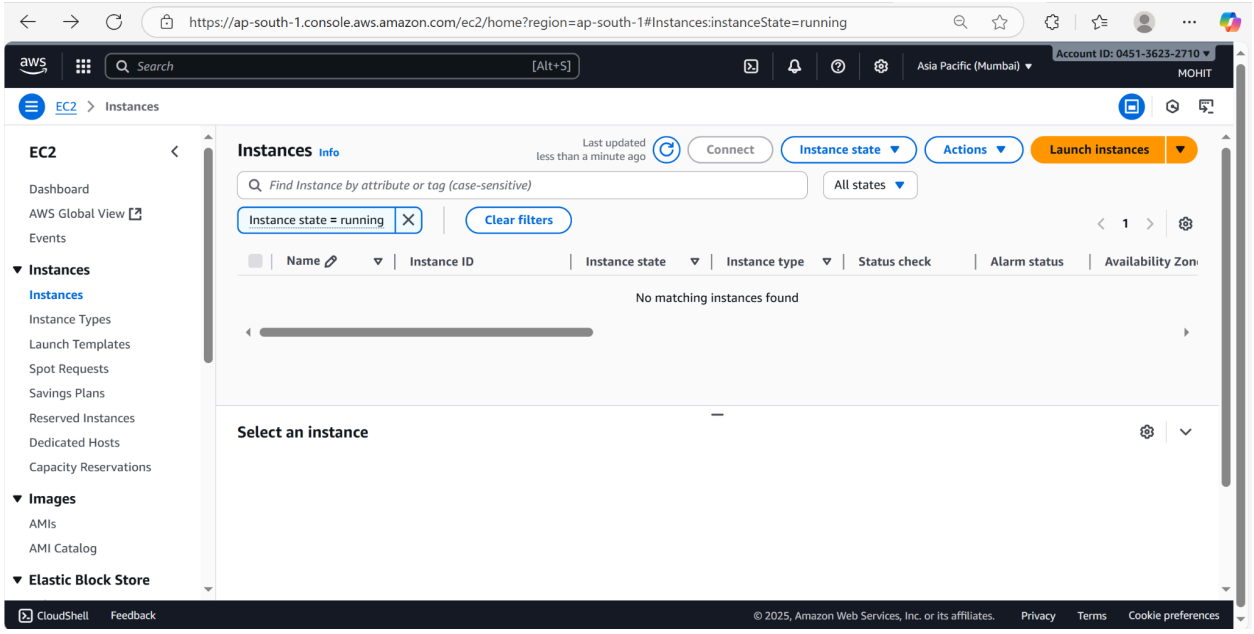
CI/CD Deployment Assignment

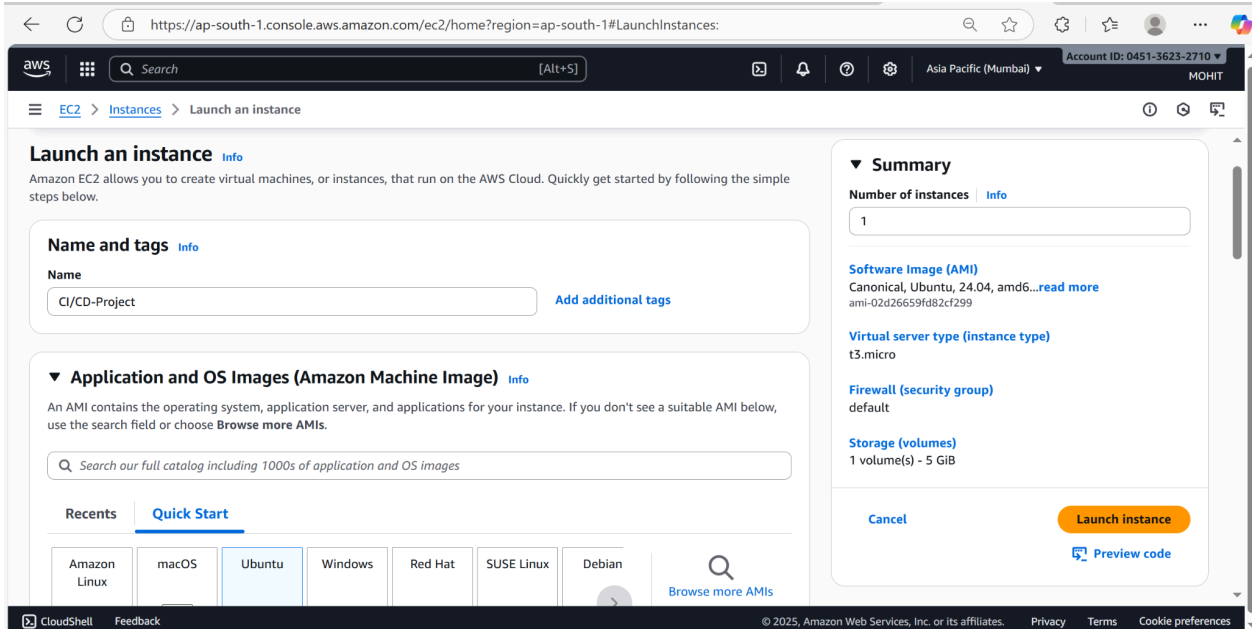
You are required to deploy a **Flask backend** and an **Express frontend** on an **Amazon EC2 instance**. Additionally, implement a **CI/CD pipeline** using Jenkins to automate the deployment process.

Part 1: Deploy Flask and Express on a Single EC2 Instance

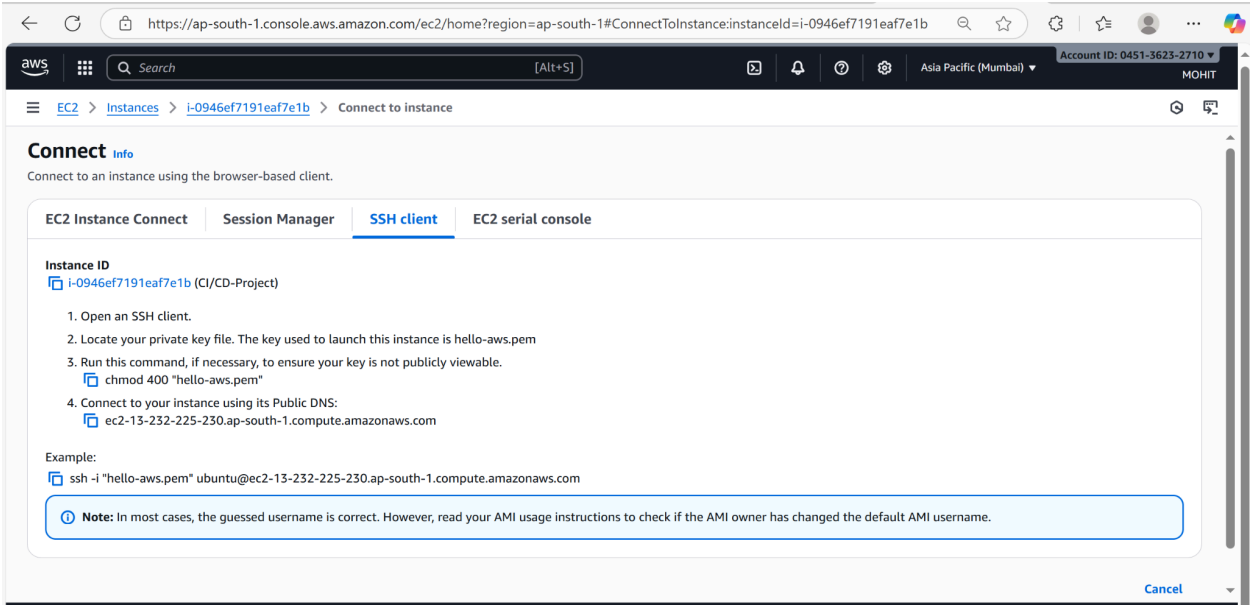
1. **Objective**:
    - Deploy both the Flask backend and the Express frontend on a **single Amazon EC2 instance**.
2. **Steps**:
    - **Provisioning the EC2 Instance**:
        - Launch an EC2 instance on AWS (you can use a free-tier eligible instance).
        - SSH into the instance and install the following dependencies:
            - Python for Flask.
            - Node.js for Express.
            - Git for pulling the application code.
    - **Application Setup**:
        - Clone the Flask and Express repositories onto the EC2 instance.
        - Install the required dependencies for both applications using `pip` and `npm`.
        - Configure both applications to run on different ports (e.g., Flask on port 5000 and Express on port 3000).
        - Start the applications using process managers like `pm2` or `systemd` to ensure they remain active.
3. **Deliverables**:
    - A running EC2 instance with Flask and Express accessible via the instance's public IP.
    - A description or diagram of the deployed architecture.

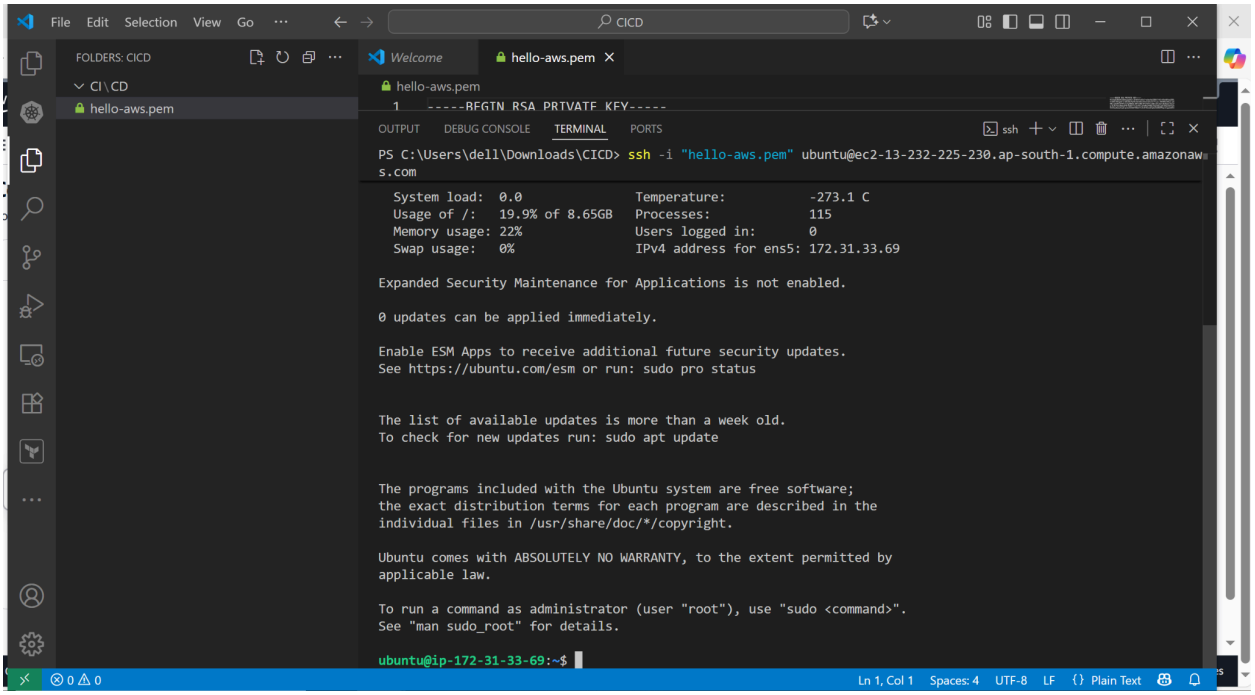
Create an instance by clicking on the launch instances



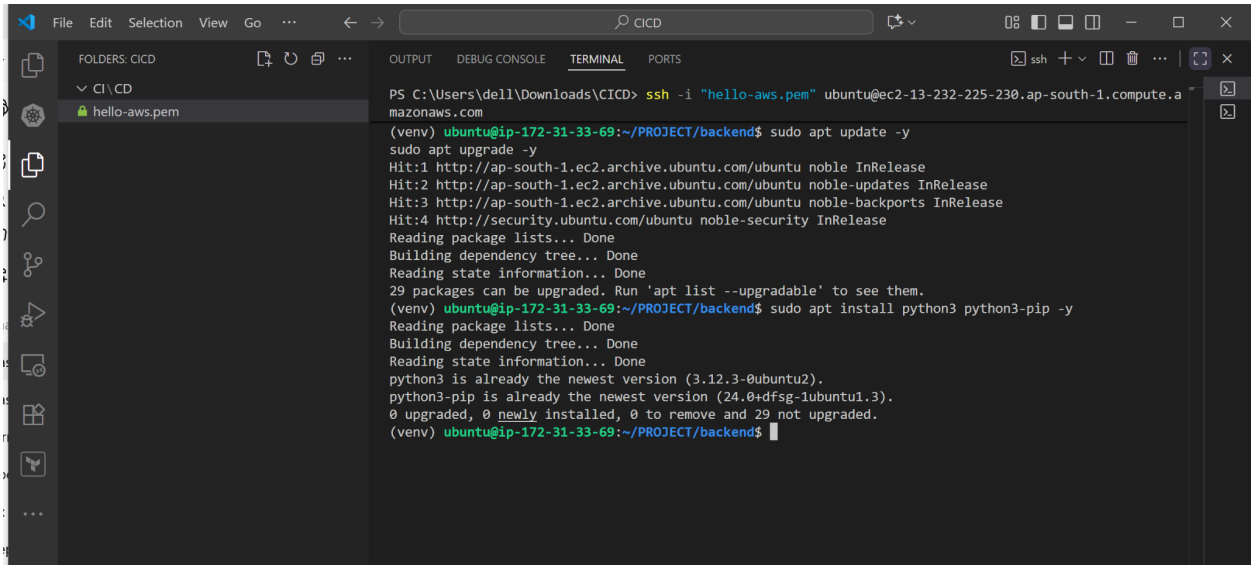


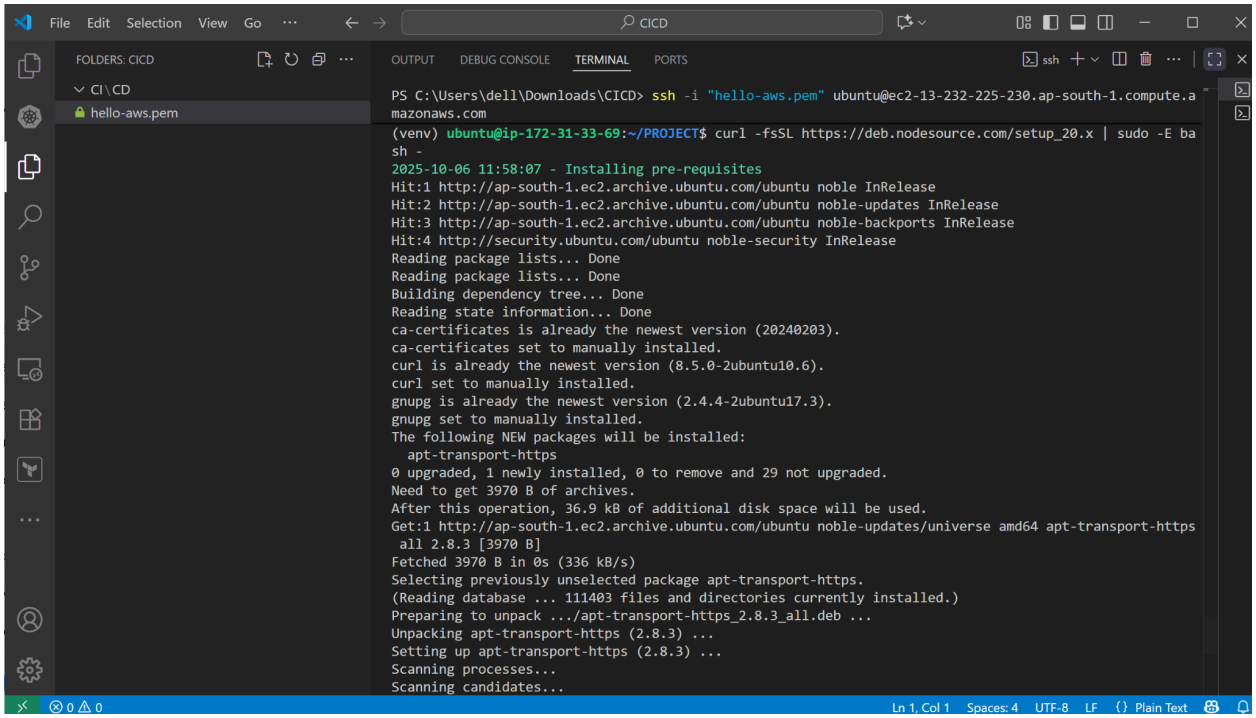
Now click on connect and run following code on terminal

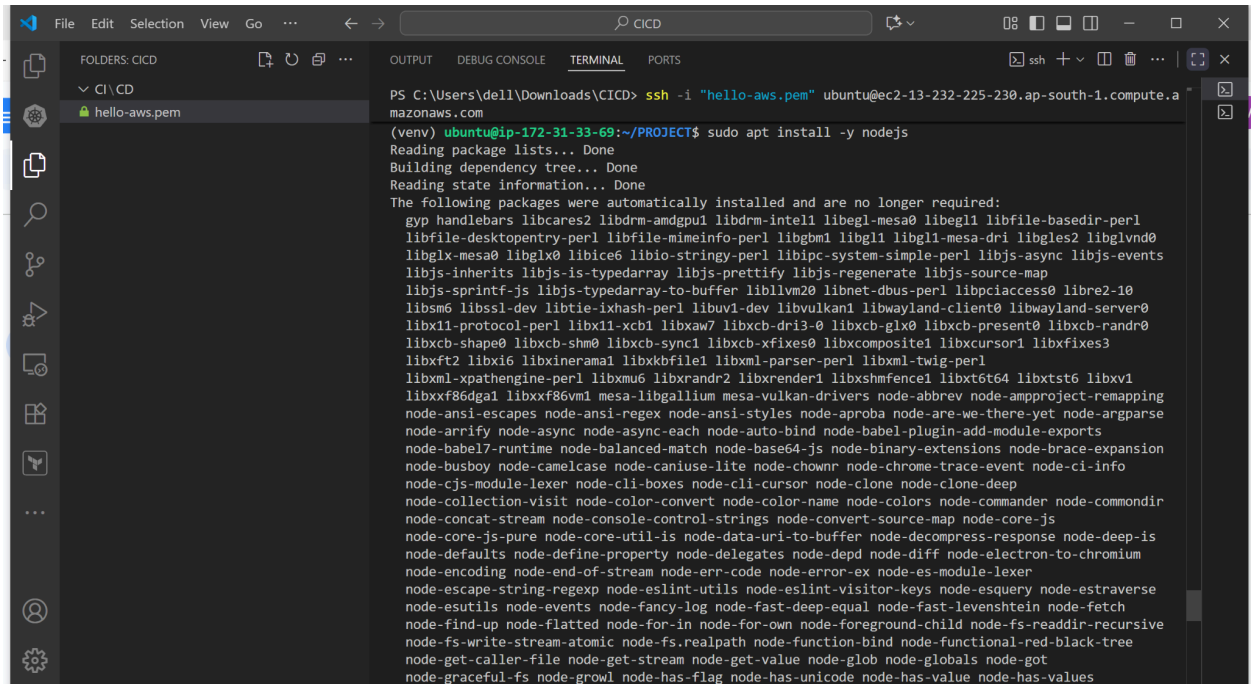




Install Dependencies

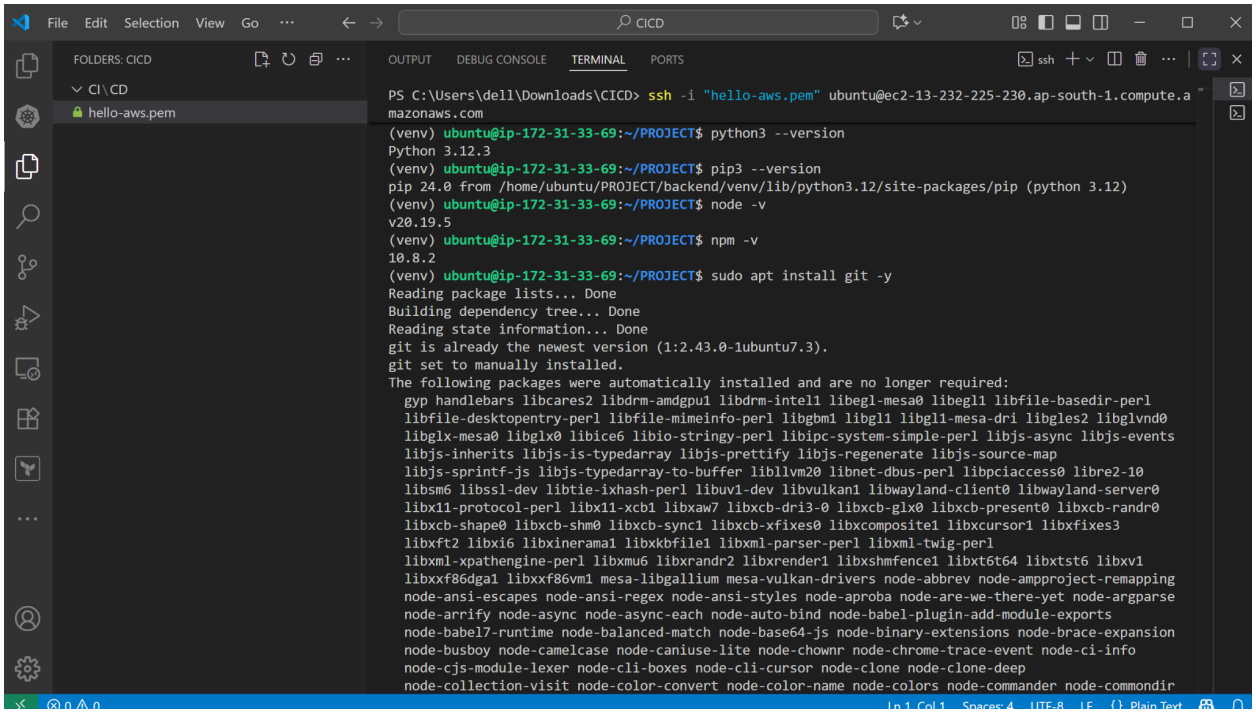




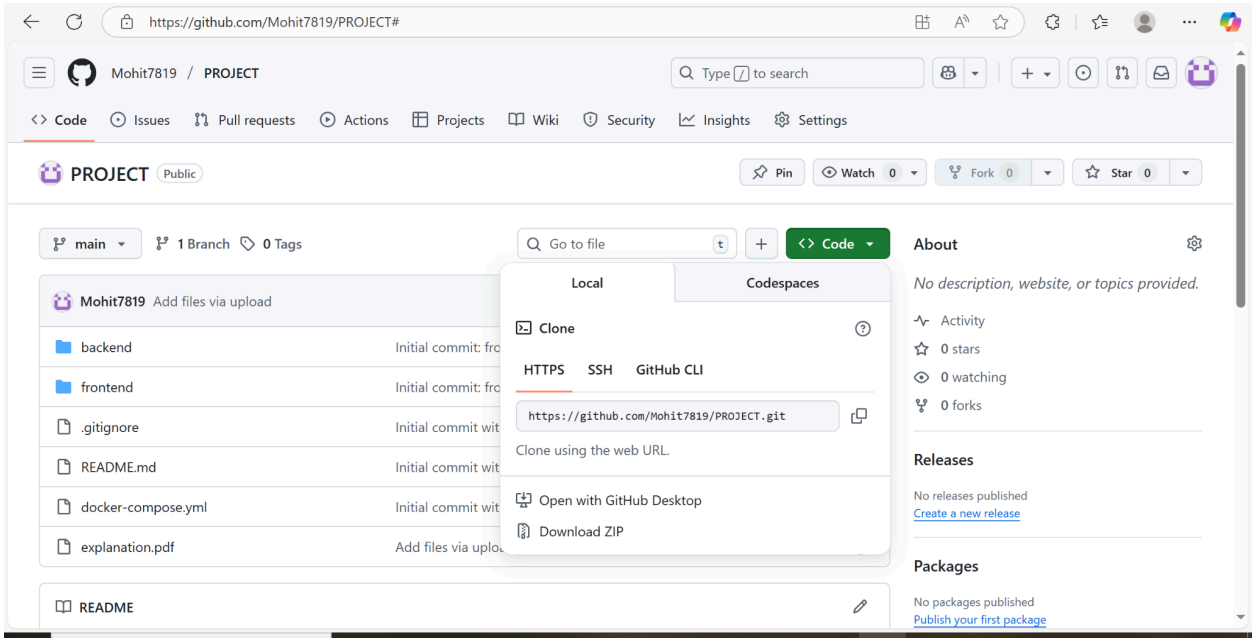



```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT$ python3 --version
Python 3.12.3
(venv) ubuntu@ip-172-31-33-69:~/PROJECT$ pip3 --version
pip 24.0 from /home/ubuntu/PROJECT/backend/venv/lib/python3.12/site-packages/pip (python 3.12)
(venv) ubuntu@ip-172-31-33-69:~/PROJECT$ node -v
v20.19.5
(venv) ubuntu@ip-172-31-33-69:~/PROJECT$ npm -v
10.8.2
(venv) ubuntu@ip-172-31-33-69:~/PROJECT$
```

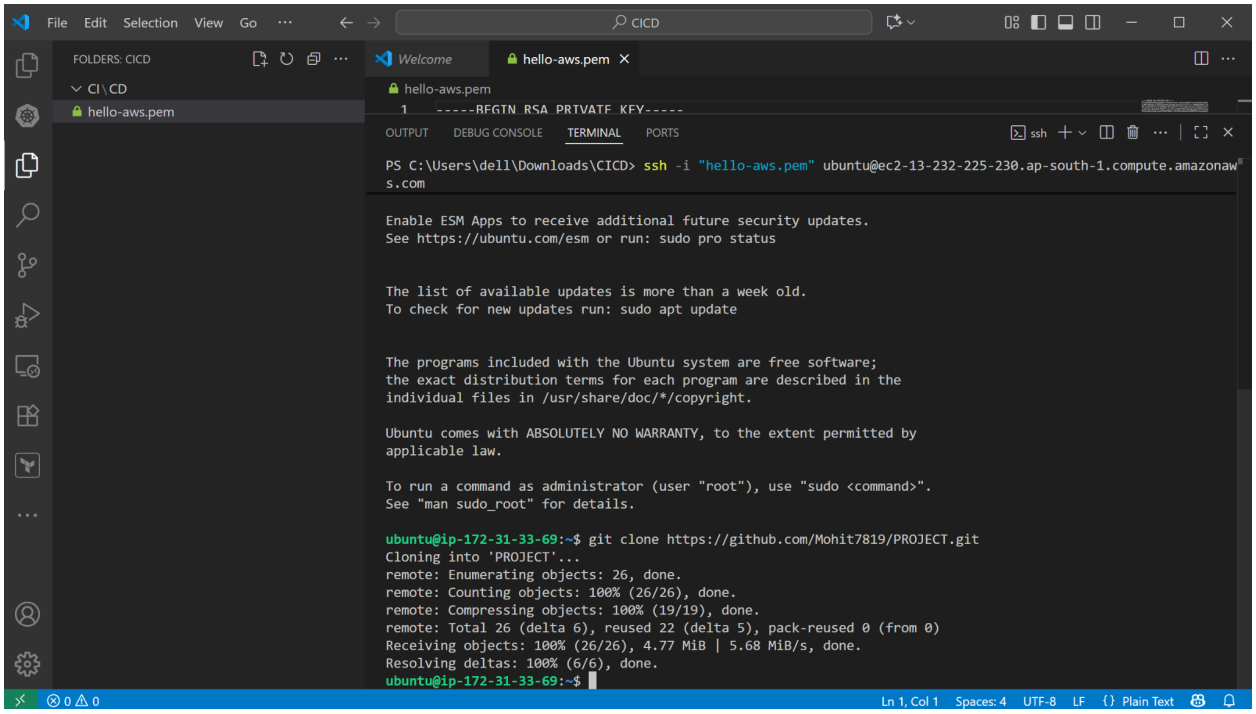
Install Git



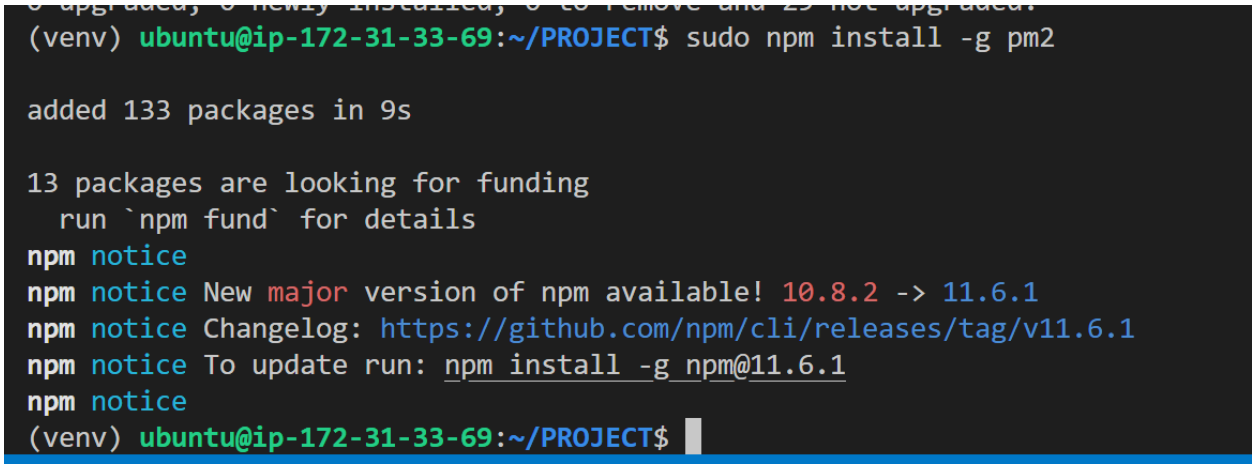
Now clone frontend and backend code from git repository

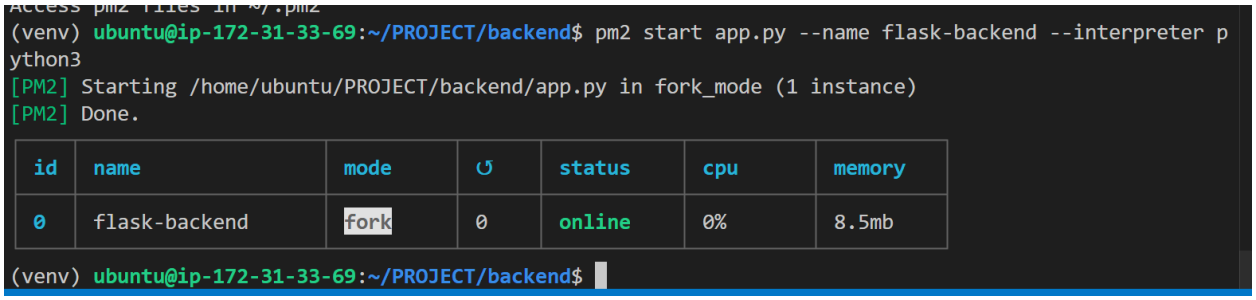


Now run git clone https://github.com/Mohit7819/PROJECT.git in terminal



Run Apps with PM2





```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$ pm2 start app.js --name express-frontend
[PM2] Starting /home/ubuntu/PROJECT/frontend/app.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	↺	status	cpu	memory
1	express-frontend	fork	0	online	0%	22.2mb
0	flask-backend	fork	0	online	0%	32.0mb

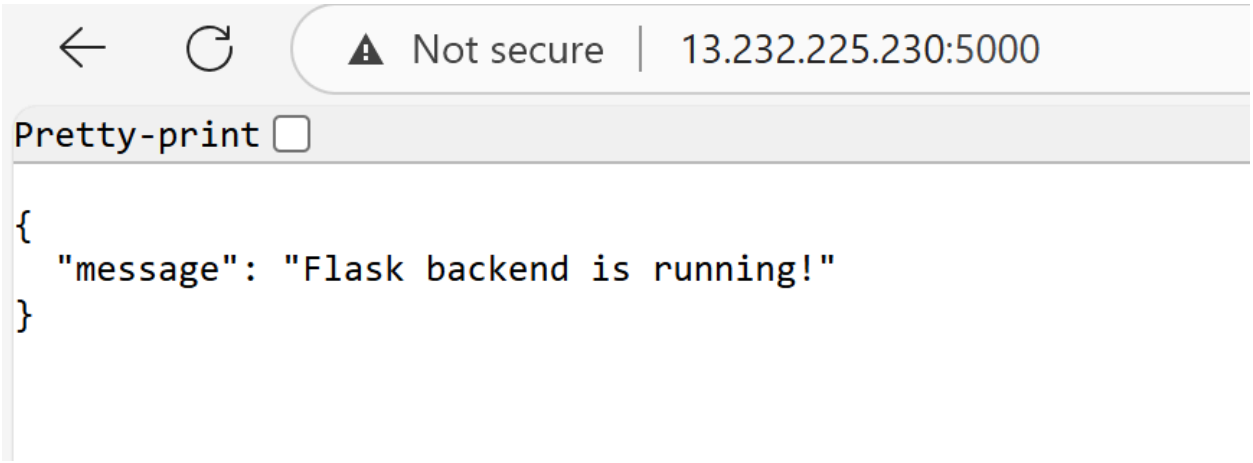
```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$
```

Save PM2 Process

```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$ pm2 save
[PM2] Saving current process list...
[PM2] Successfully saved in /home/ubuntu/.pm2/dump.pm2
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$ pm2 startup
[PM2] Init System found: systemd
[PM2] To setup the Startup Script, copy/paste the following command:
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u ubuntu --hp /home/ub
untu
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$
```

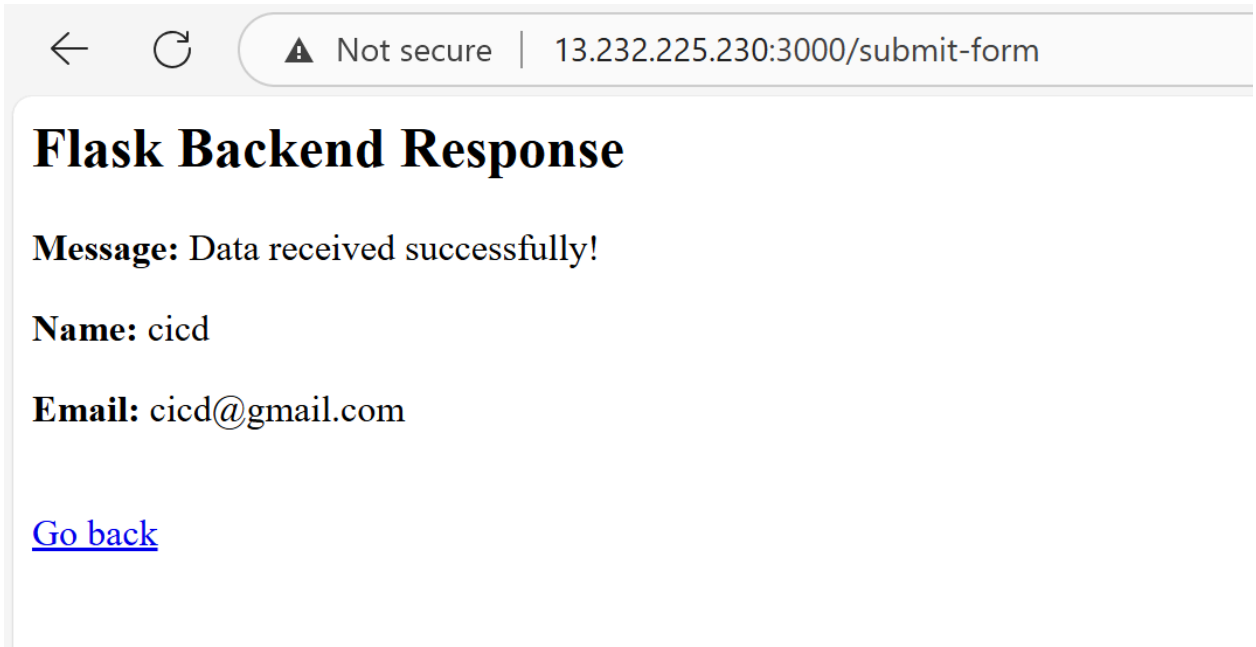
Tests apps

- Flask: `http://13.232.225.230:5000`



- Express:


Not secure | 13.232.225.230:3000
Send Data to Flask Backend
Name: cicd
Email: cicd@gmail.com
Submit


Not secure | 13.232.225.230:3000/submit-form
Flask Backend Response
Message: Data received successfully!
Name: cicd
Email: cicd@gmail.com
Go back

For stopping the app

```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$ pm2 stop express-frontend
[PM2] Applying action stopProcessId on app [express-frontend](ids: [ 1 ])
[PM2] [express-frontend](1) ✓
```

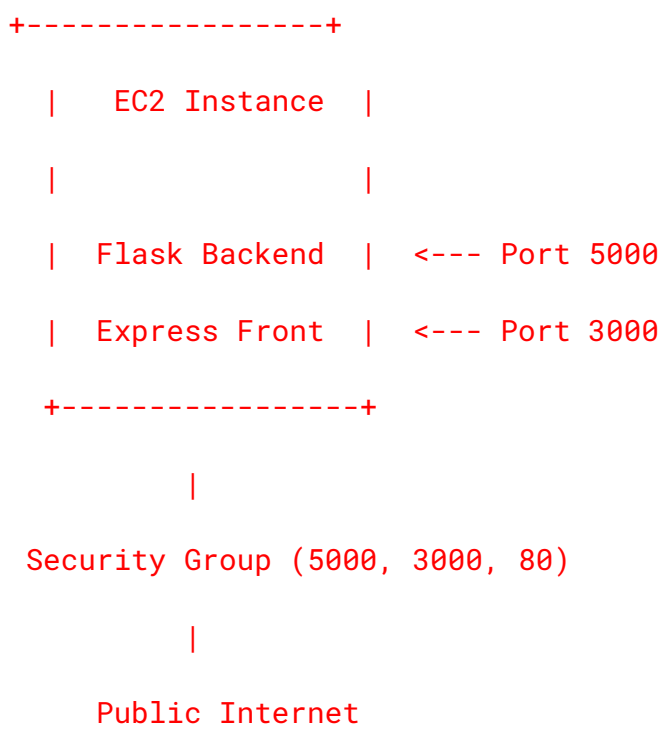
id	name	mode	↺	status	cpu	memory
1	express-frontend	fork	0	stopped	0%	0b
0	flask-backend	fork	0	online	0%	32.0mb

```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$ pm2 stop flask-backend
[PM2] Applying action stopProcessId on app [flask-backend](ids: [ 0 ])
[PM2] [flask-backend](0) ✓
```

id	name	mode	↺	status	cpu	memory
1	express-frontend	fork	0	stopped	0%	0b
0	flask-backend	fork	0	stopped	0%	0b

```
(venv) ubuntu@ip-172-31-33-69:~/PROJECT/frontend$
```

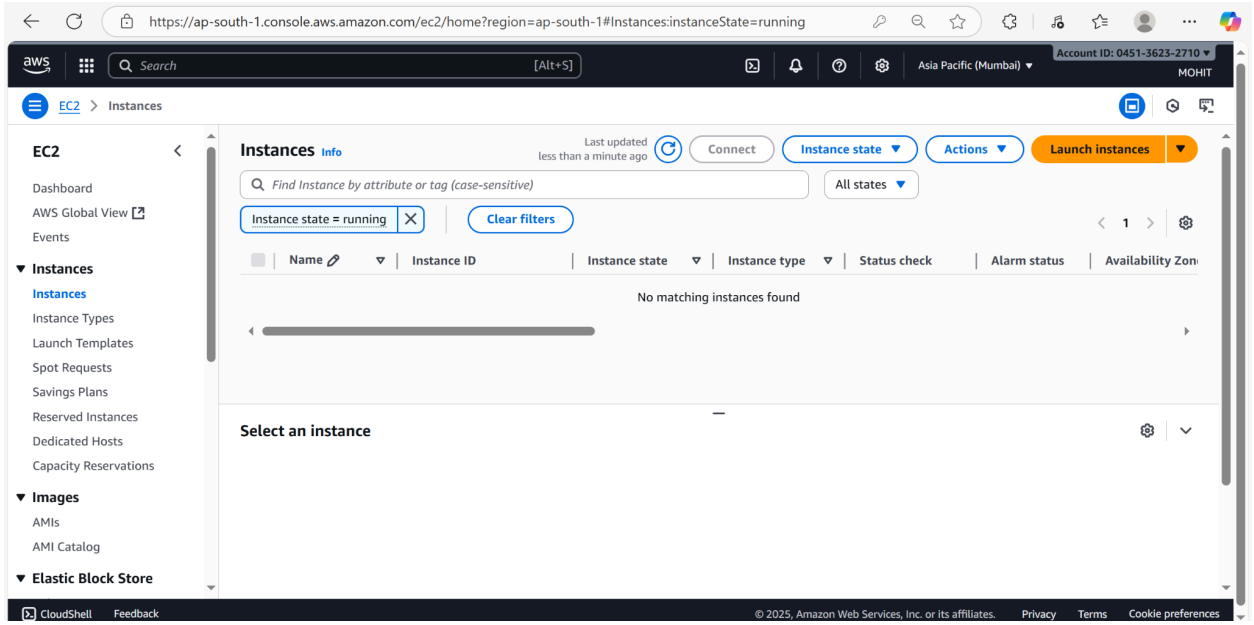
Architecture Diagram

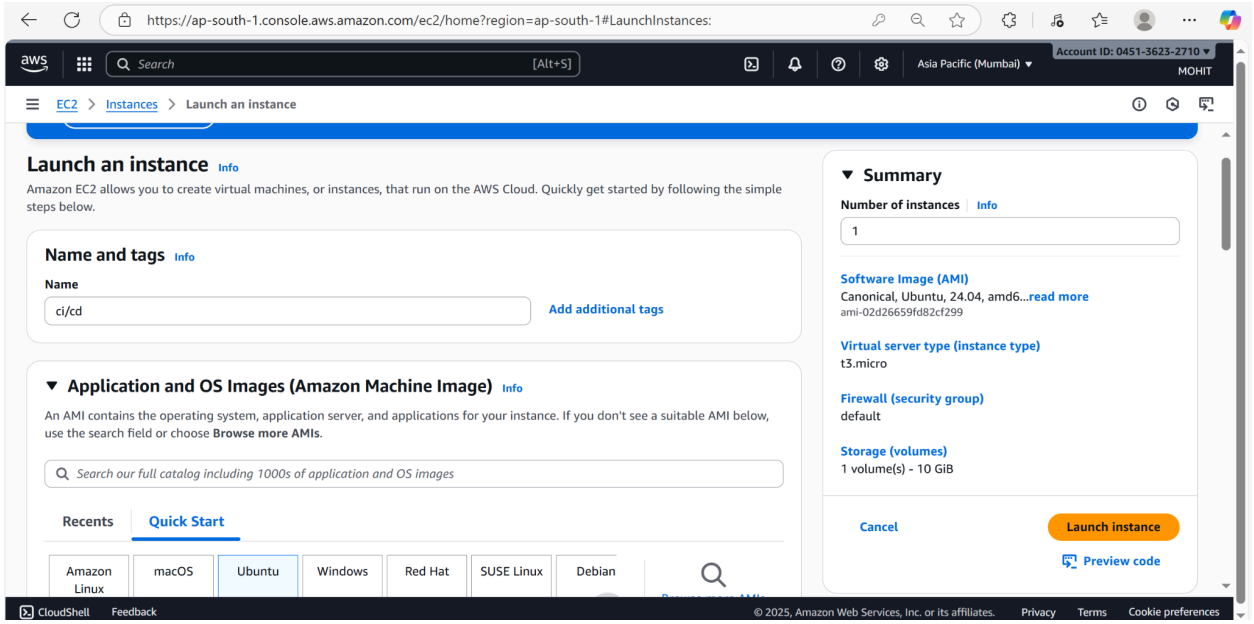


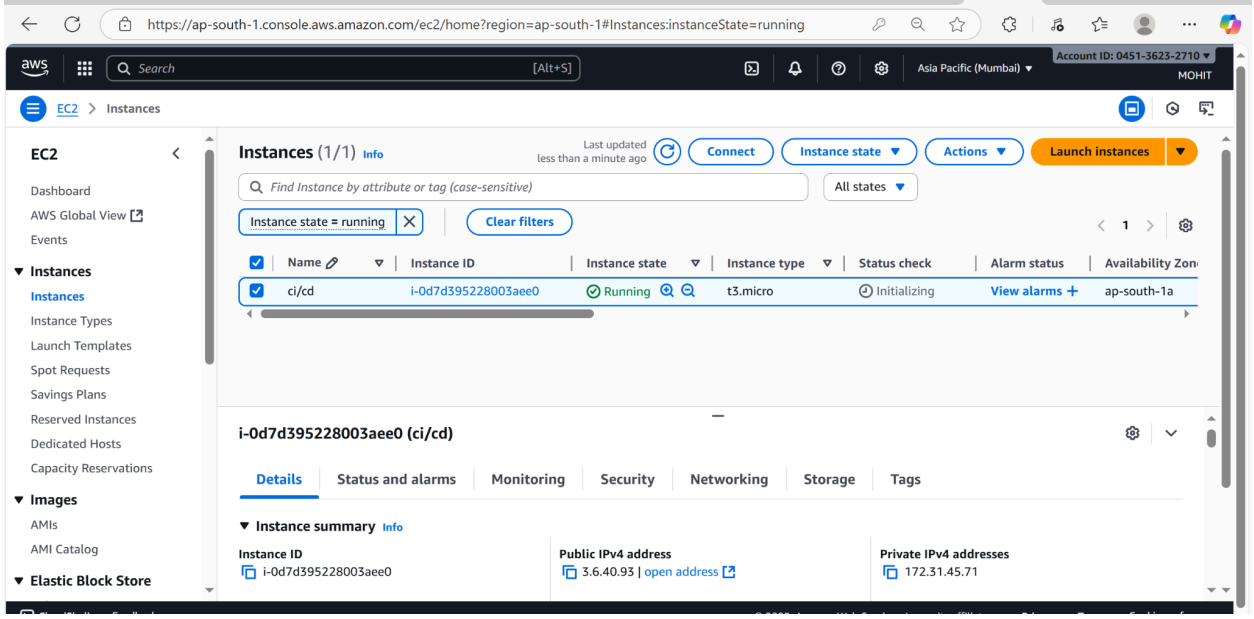
Part 2: Implement CI/CD Pipeline Using Jenkins

1. **Objective**:
    - Automate the deployment of Flask and Express applications using Jenkins.
2. **Steps**:
    - **Install Jenkins**:
        - Install Jenkins on the same EC2 instance or on a separate machine.
        - Configure Jenkins by installing essential plugins like Git, NodeJS, and Python.
    - **Set Up Jenkins Pipeline**:
        - Create two separate Jenkins pipelines for the Flask and Express applications.
        - **Pipeline Steps**:
            - Pull the latest code from the respective Git repositories.
            - Install dependencies for Flask (`pip install -r requirements.txt`) and Express (`npm install`).
            - Restart the applications using the process manager (e.g., `pm2 restart <app>`).
    - **Triggering the Pipeline**:
        - Set up a GitHub webhook to trigger the Jenkins pipeline on every push to the repositories.
        -
    - **Optional Enhancements**:
        - Add testing stages to the pipeline for both applications.
        - Configure environment variables in Jenkins for managing sensitive data like API keys.
3. **Deliverables**:
    - A fully functional CI/CD pipeline that automates the deployment process for Flask and Express.
    - A Jenkins pipeline script (`Jenkinsfile`) for each application.
    - Evidence of the pipeline working (e.g., screenshots of successful builds and deployments).

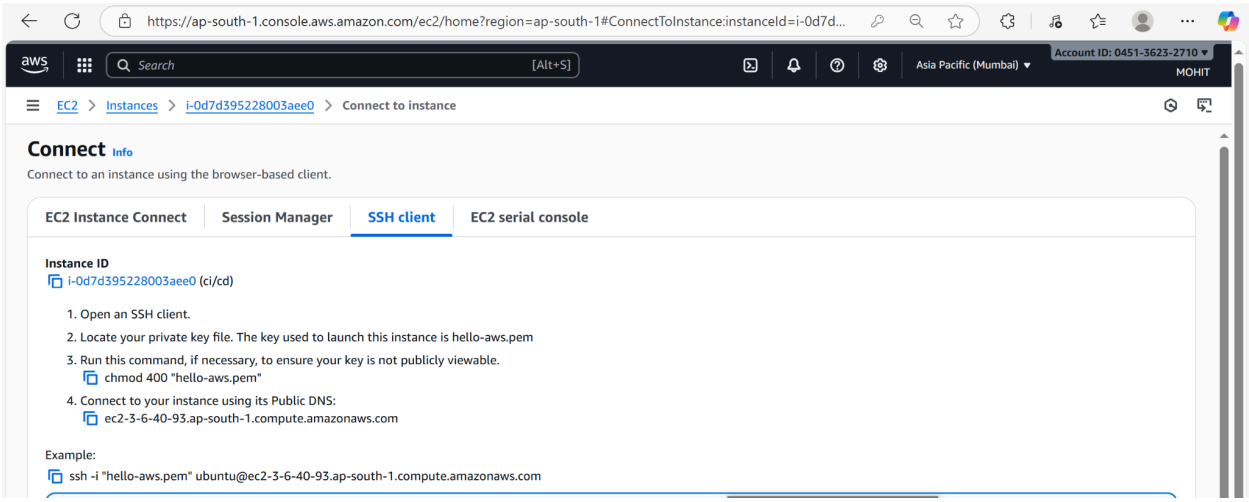
Creating instances by clicking on launch instances.

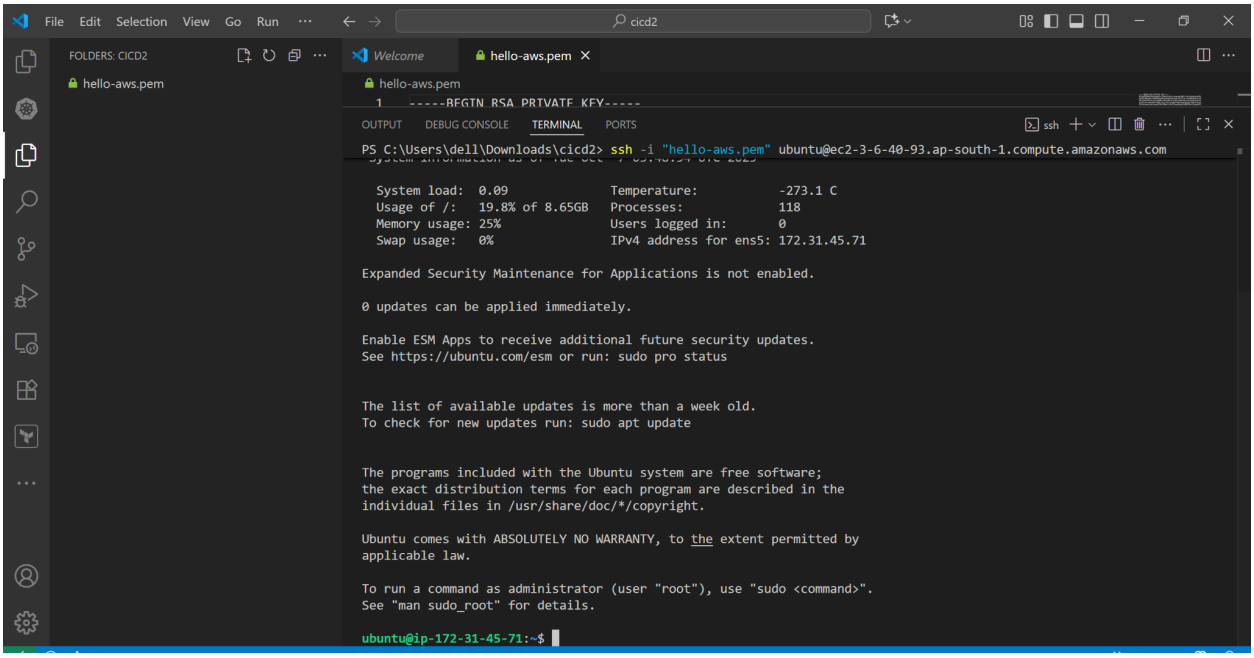



https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#LaunchInstances:
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 0451-3623-2710
MOHIT
EC2 > Instances > Launch an instance
Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
ci/cd
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose Browse more AMIs.
Search our full catalog including 1000s of application and OS images
Recents
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Linux
Debian
Summary
Number of instances Info
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-02d26659fd82cf299
Virtual server type (instance type)
t3.micro
Firewall (security group)
default
Storage (volumes)
1 volume(s) - 10 GiB
Cancel
Launch instance
Preview code
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Instances:instanceState=running
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 0451-3623-2710
MOHIT
EC2 > Instances
EC2
Dashboard
AWS 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
Instances (1/1) Info
Last updated less than a minute ago
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 Zone
ci/cd
i-0d7d395228003aee0
Running
t3.micro
Initializing
View alarms
ap-south-1a
i-0d7d395228003aee0 (ci/cd)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0d7d395228003aee0
Public IPv4 address
3.6.40.93 | open address
Private IPv4 addresses
172.31.45.71

Now connect instances to terminal by clicking on the connect and paste the given command in your terminal.





System update + create deployer user by following commands:

sudo apt update -y && sudo apt upgrade -y

create deployer

sudo adduser --disabled-password --gecos "" deployer

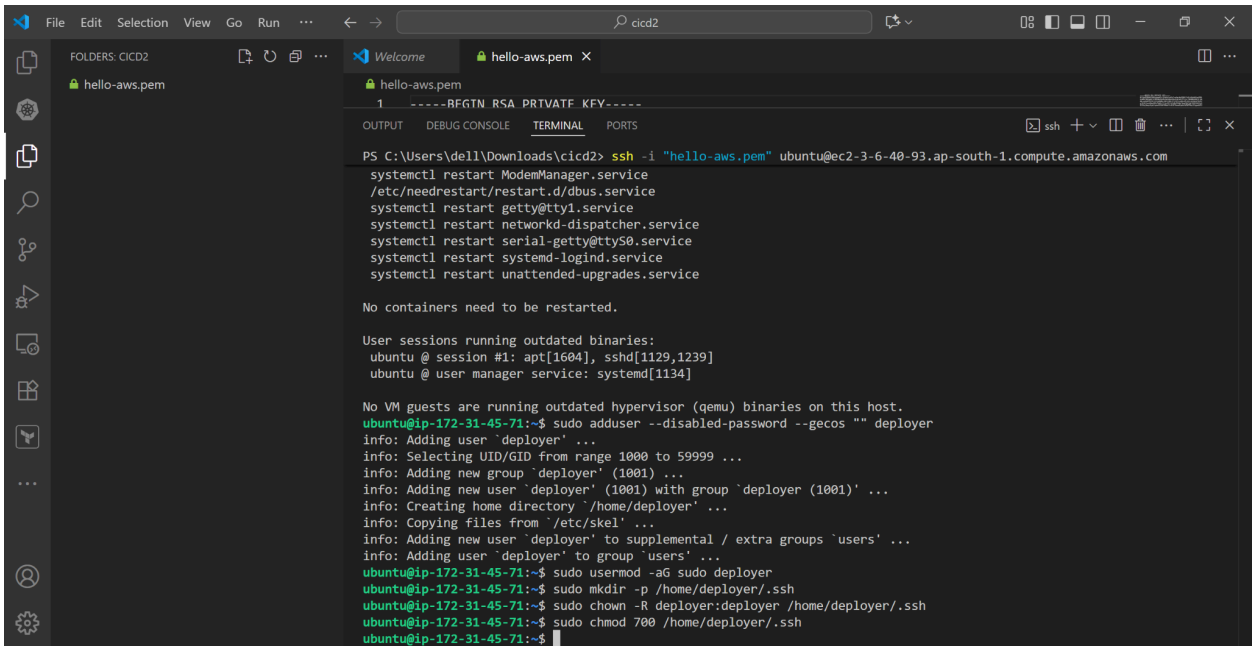
sudo usermod -aG sudo deployer

create .ssh for deployer (used later)

sudo mkdir -p /home/deployer/.ssh

sudo chown -R deployer:deployer /home/deployer/.ssh

sudo chmod 700 /home/deployer/.ssh



Install required packages (Java, Git, Node, Python, rsync, etc.) and pm2

basic tools

sudo apt install -y git python3 python3-venv python3-pip build-essential rsync curl

install openjdk (Jenkins requirement)

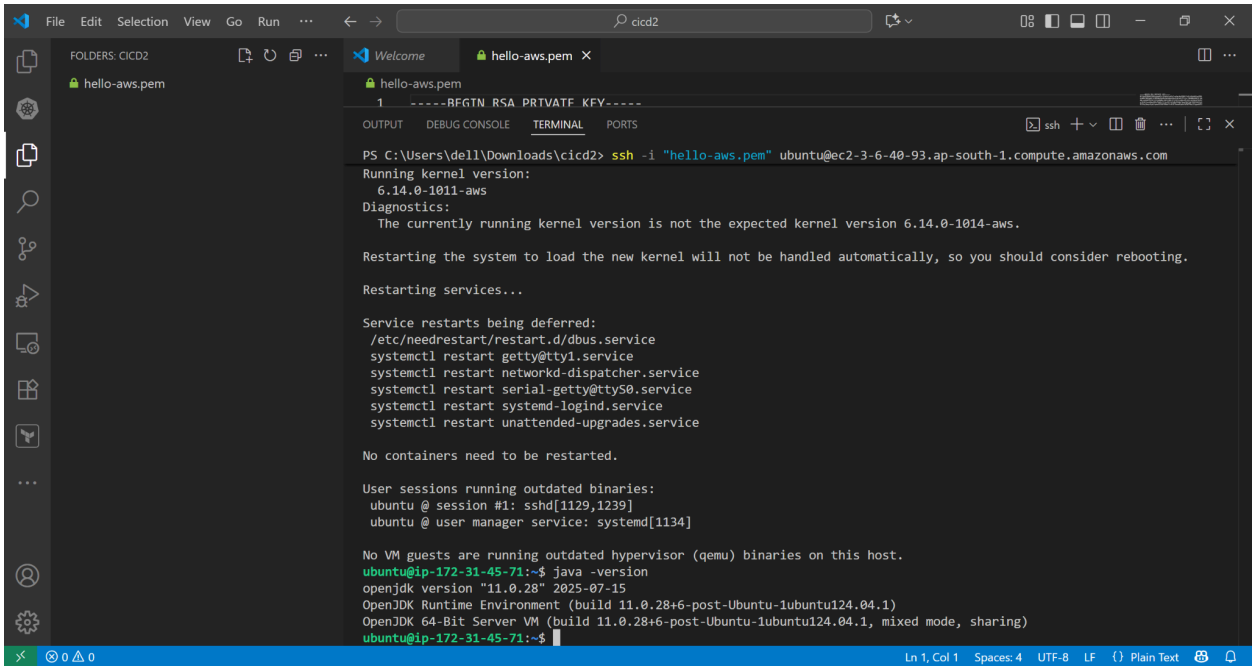
sudo apt install -y openjdk-11-jdk

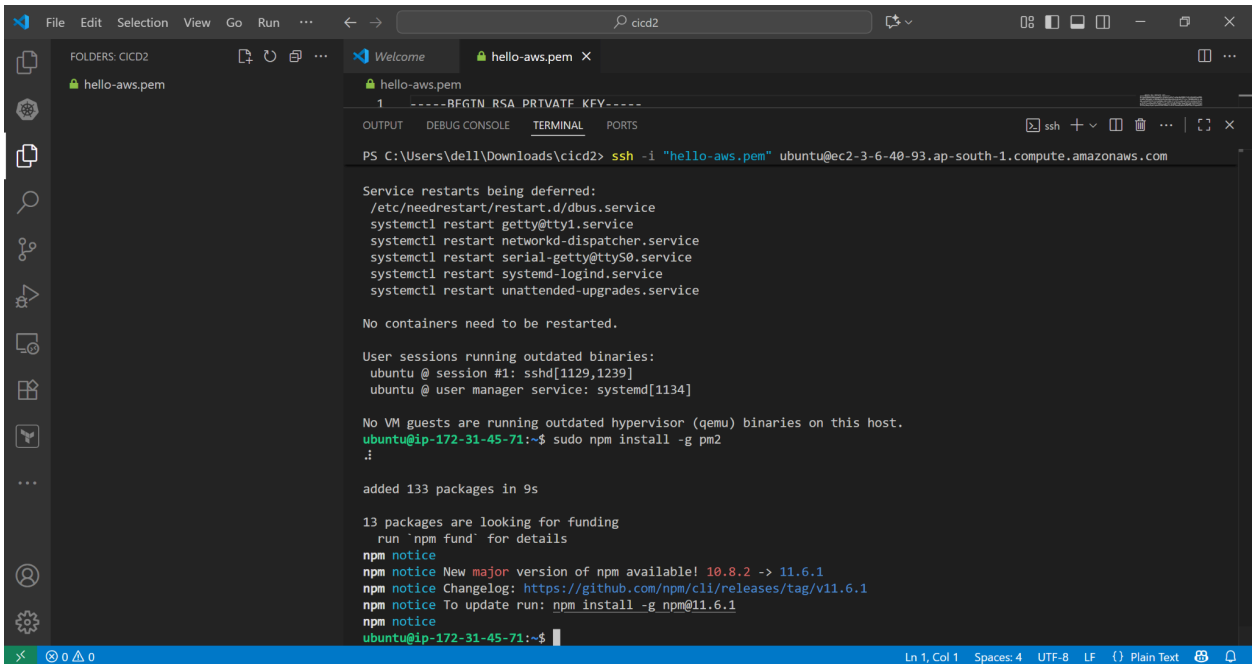
java -version

Node.js (LTS)

curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -

```
sudo apt-get install -y nodejs

node -v && npm -v


# global pm2 (for Node and python service management)

sudo npm install -g pm2

pm2 -v
```



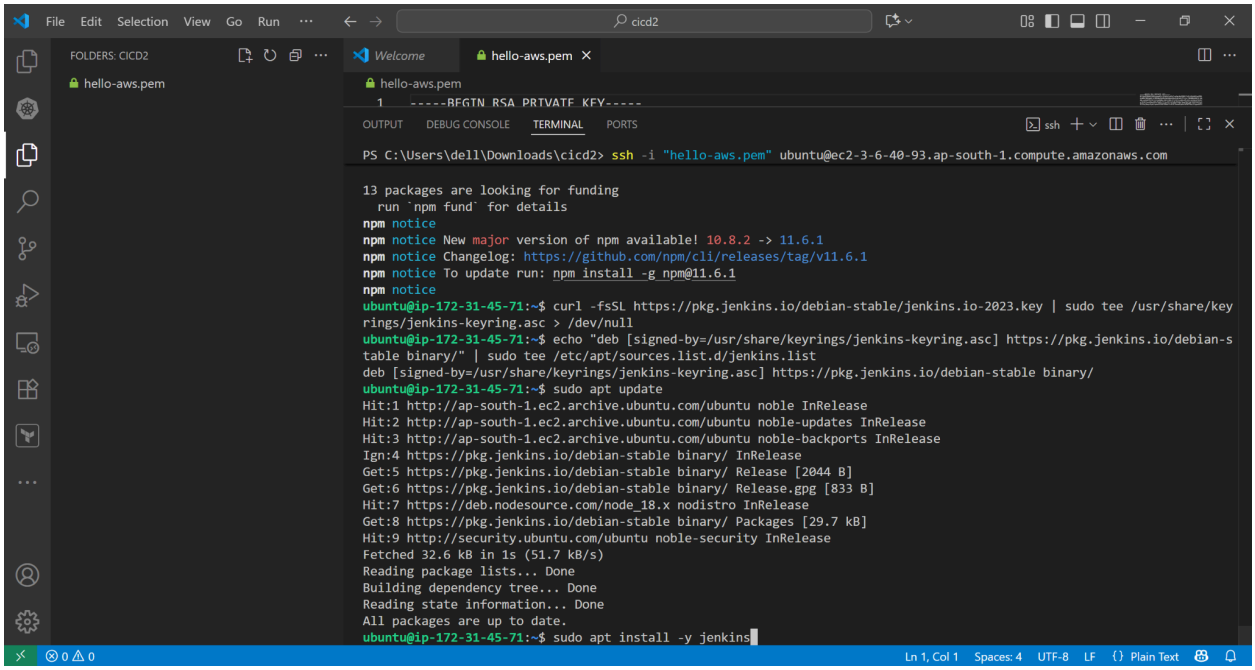


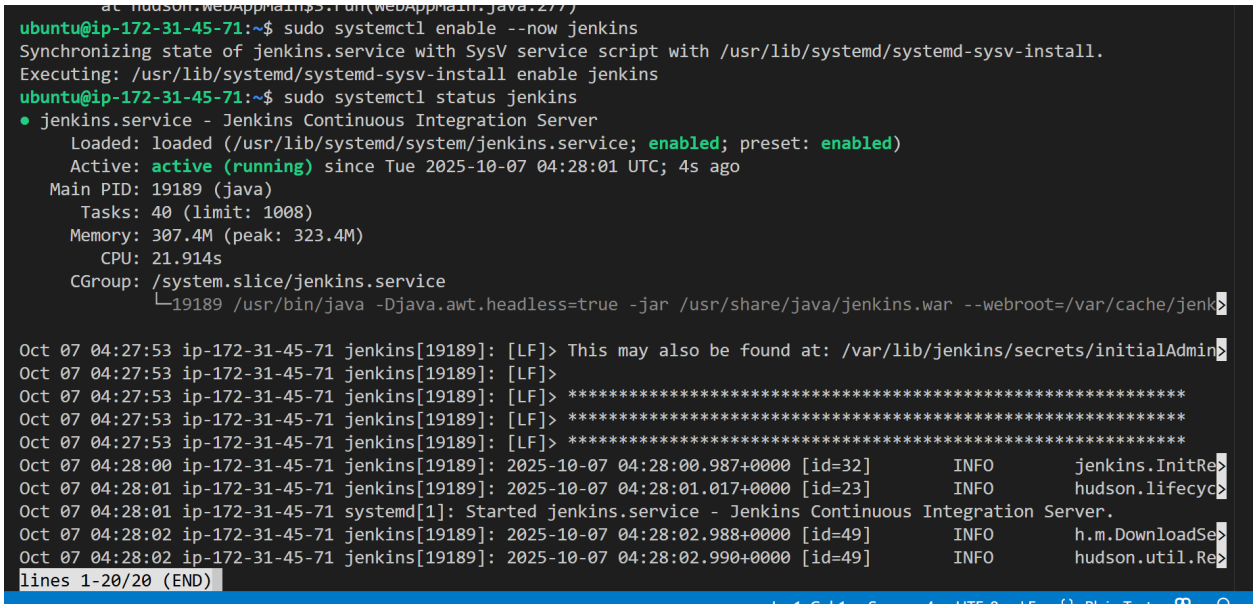
Install Jenkins (official repo) and start service

add Jenkins repo (modern method)

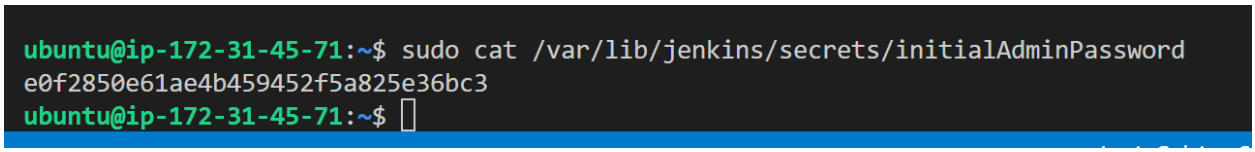
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

```
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


sudo apt update

sudo apt install -y jenkins


# enable and start

sudo systemctl enable --now jenkins

sudo systemctl status jenkins
```

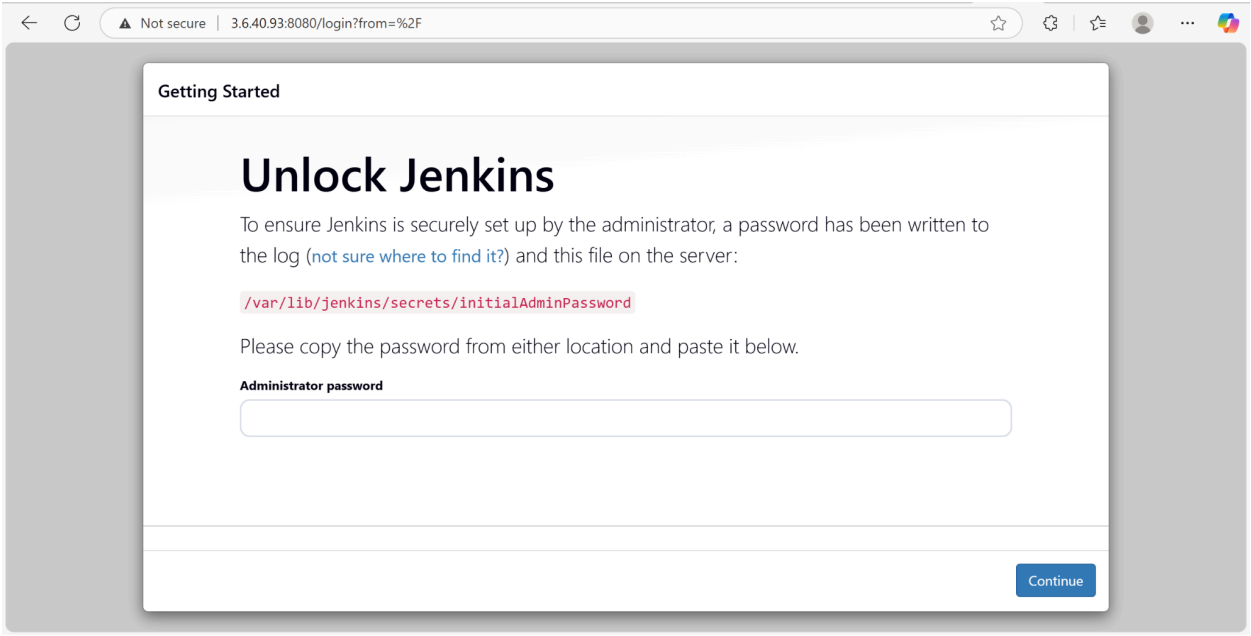




Get initial admin password:



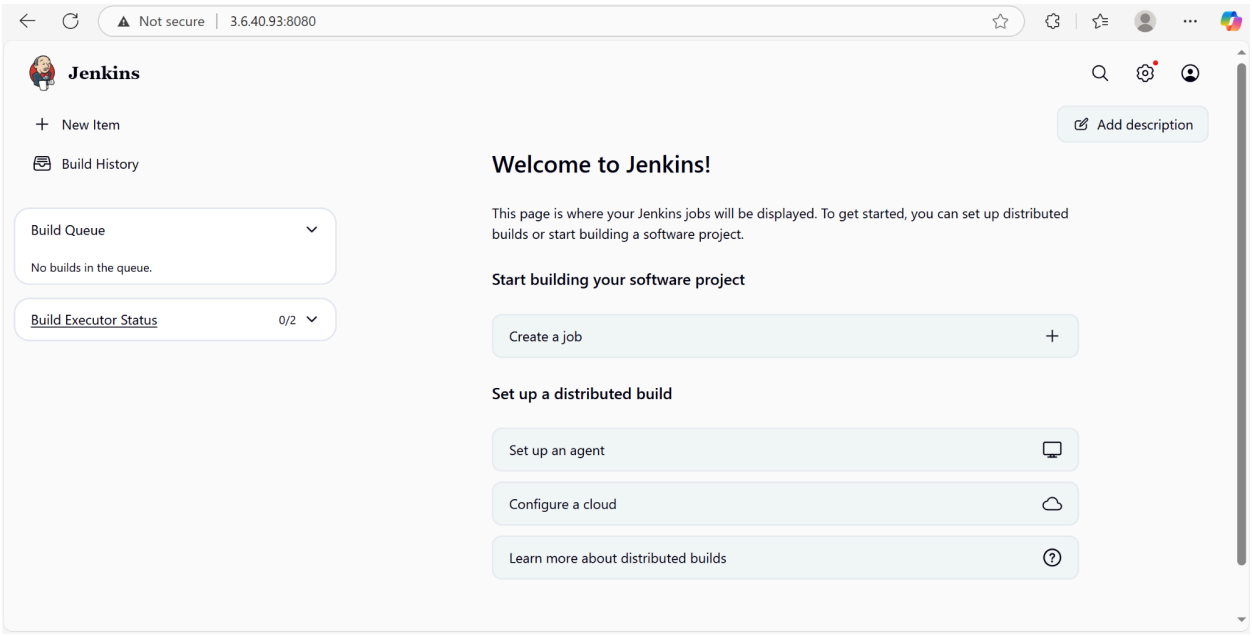
Now open ec2 and click on ipv4 address with port 8080 in browser we get this



Install recommended Jenkins plugins (first login)

1. Visit `http://<EC2_PUBLIC_IP>:8080` and complete initial setup with the password above.
2. When asked, select **Install suggested plugins**.
3. Ensure these are installed (Manage Jenkins → Manage Plugins → Installed):
    - Git plugin
    - Pipeline (workflow-aggregator)
    - GitHub plugin / GitHub Branch Source
    - Credentials Binding plugin
    - NodeJS plugin (optional)
    - Generic Webhook Trigger (optional but helpful)
    - Blue Ocean (optional)

(You can add others later.)



Give Jenkins ability to deploy to `deployer`

We want Jenkins to be able to copy files and restart pm2 **as** `deployer`. Create this sudoers rule:

1. Check pm2 path for accuracy:

```
which pm2

# e.g. /usr/bin/pm2
```

2. Create sudoers entry so `jenkins` can run a few commands as `deployer` without password:

```
sudo bash -c 'echo "jenkins ALL=(deployer) NOPASSWD:
/usr/bin/pm2, /usr/bin/rsync, /usr/bin/systemctl, /bin/chown,
/bin/cp, /bin/mv" > /etc/sudoers.d/jenkins_deployer'

sudo chmod 0440 /etc/sudoers.d/jenkins_deployer
```

If pm2 path differs, update /usr/bin/pm2 accordingly.

```
ubuntu@ip-172-31-45-71:~$ which pm2
/usr/bin/pm2
```

```
ubuntu@ip-172-31-45-71:~$ sudo bash -c 'echo "jenkins ALL=(deployer) NOPASSWD: /usr/bin/pm2, /usr/bin/rsync, /usr/bin/s
ystemctl, /bin/chown, /bin/cp, /bin/mv" > /etc/sudoers.d/jenkins_deployer'
ubuntu@ip-172-31-45-71:~$ sudo chmod 0440 /etc/sudoers.d/jenkins_deployer
ubuntu@ip-172-31-45-71:~$
```

Step 6 — Add deploy keys (two options) so Jenkins & deployer can access GitHub

Option A — recommended: Use SSH keys

1. Generate SSH key for jenkins (on the EC2):

   ```
   sudo -u jenkins ssh-keygen -t ed25519 -C "jenkins@$(hostname)" -f /var/lib/jenkins/.ssh/id_ed25519 -N ""

   sudo chown -R jenkins:jenkins /var/lib/jenkins/.ssh

   sudo chmod 700 /var/lib/jenkins/.ssh

   sudo cat /var/lib/jenkins/.ssh/id_ed25519.pub
   ```

2. Copy the printed public key and add it on GitHub:

   - For each repo → Settings → Deploy keys → **Add deploy key** and allow **Write access** if Jenkins needs to push. For read-only pulls, write is not needed.

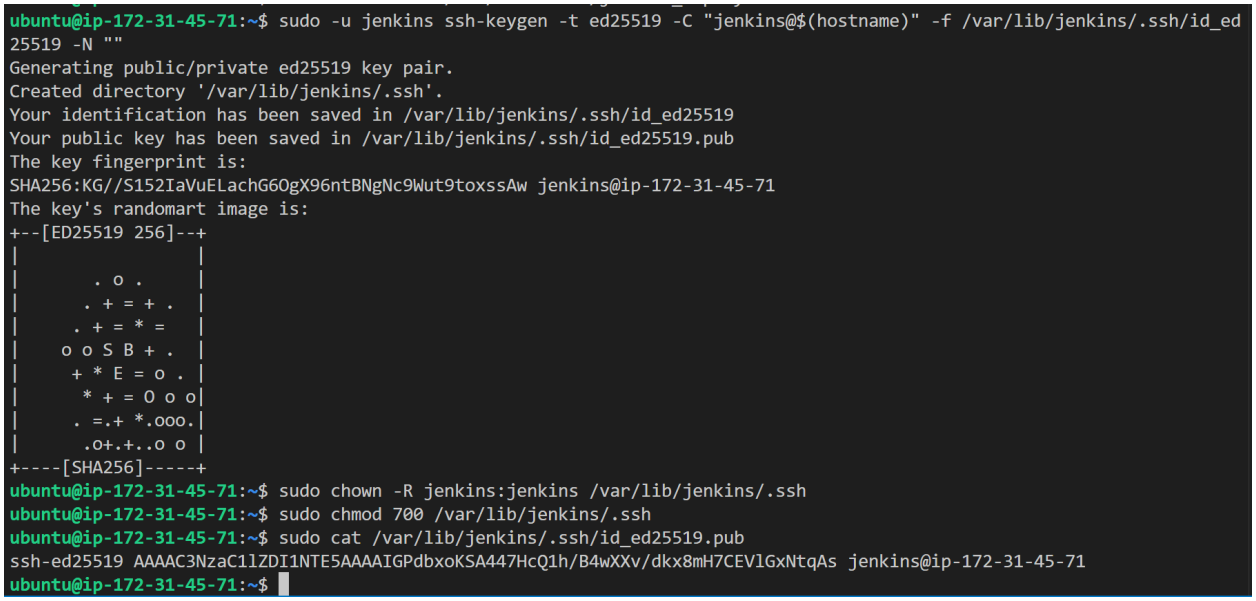
3. Generate SSH key for deployer (used if you want deployer to clone directly):

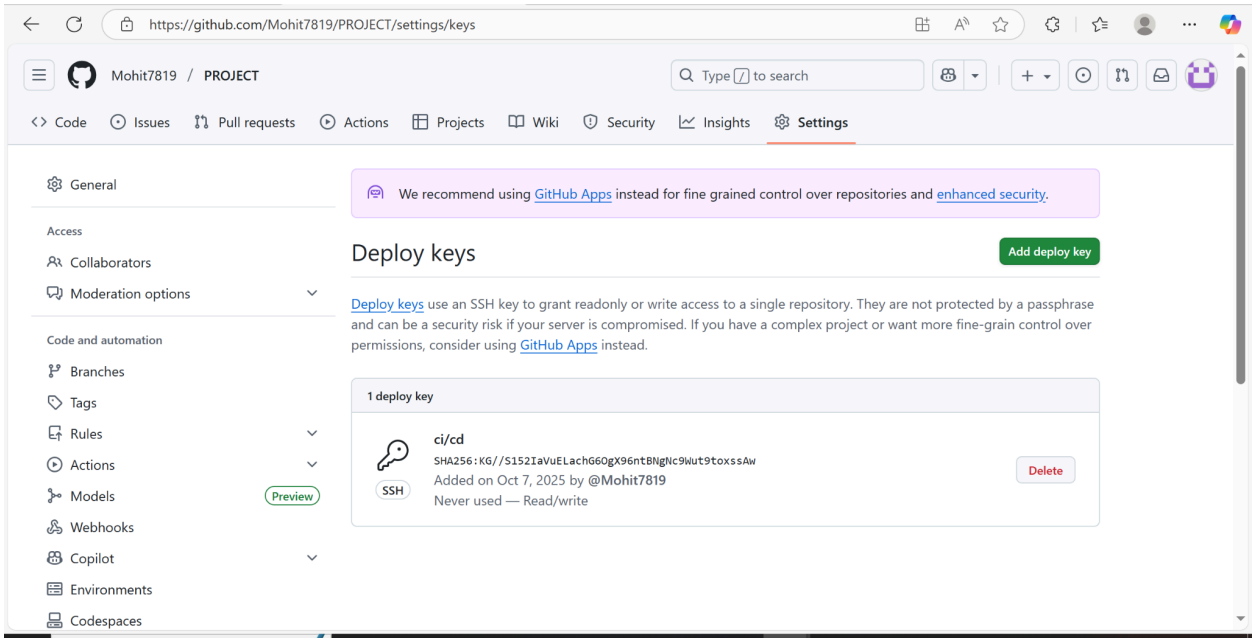
   ```
   sudo -u deployer ssh-keygen -t ed25519 -C "deployer@$(hostname)" -f /home/deployer/.ssh/id_ed25519 -N ""
   ```

```
sudo cat /home/deployer/.ssh/id_ed25519.pub

# Add deployer's public key to GitHub repository (as deploy key
or to your GitHub account)
```

Option B — Use GitHub Personal Access Token (PAT)

If using HTTPS with PAT, save PAT in Jenkins Credentials as "Secret text" or "Username/password" (username = your GitHub user, password = PAT). I recommend SSH keys.





Step 7 – Prepare an initial manual deploy (one-time) for both apps under deployer

This ensures pm2 process is created and systemd startup is configured.

Flask (example uses Gunicorn and a wsgi.py / wsgi:app)

```
# as deployer
sudo -u deployer -i bash <<'EOS'
cd /home/deployer
git clone git@github.com:YOU/flask-repo.git flask-app
cd flask-app
python3 -m venv venv
. venv/bin/activate
pip install -r requirements.txt
# Start via gunicorn (adjust module name if different)
```

```
pm2 start gunicorn --name flask-app --bind 0.0.0.0:5000
"wsgi:app"

pm2 save

# Setup pm2 systemd startup for deployer

pm2 startup systemd -u deployer --hp /home/deployer

# The last command prints a sudo command — run it:

sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u deployer --hp
/home/deployer

sudo systemctl enable pm2-deployer

EOS
```

- Confirm app: `curl http://localhost:5000` (on server) or `http://<EC2_PUBLIC_IP>:5000` from browser (open security group port 5000).

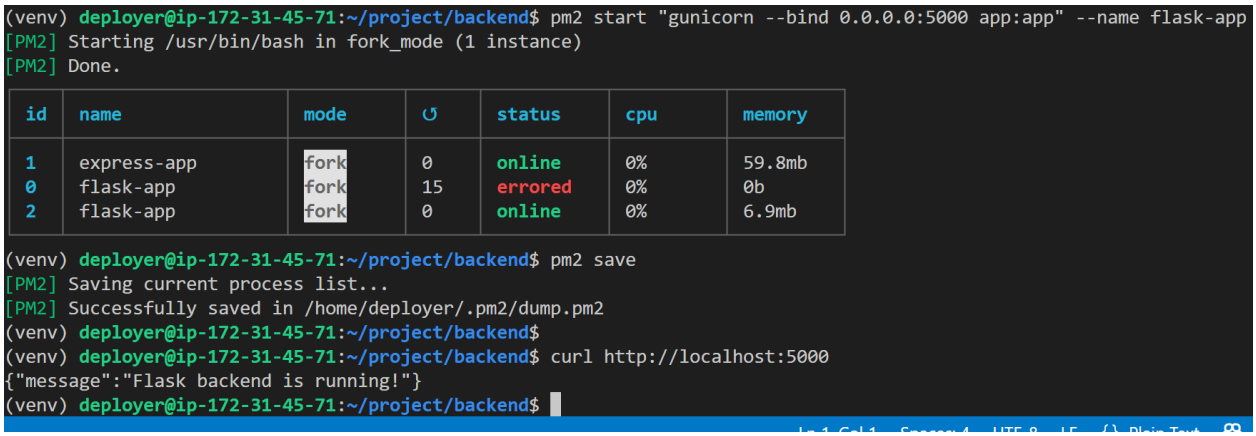
Express

```
sudo -u deployer -i bash <<'EOS'

cd /home/deployer

git clone git@github.com:YOU/express-repo.git express-app

cd express-app

npm ci

pm2 start npm --name express-app -- start

pm2 save

pm2 startup systemd -u deployer --hp /home/deployer
```

```
sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u deployer --hp
/home/deployer

sudo systemctl enable pm2-deployer

EOS
```

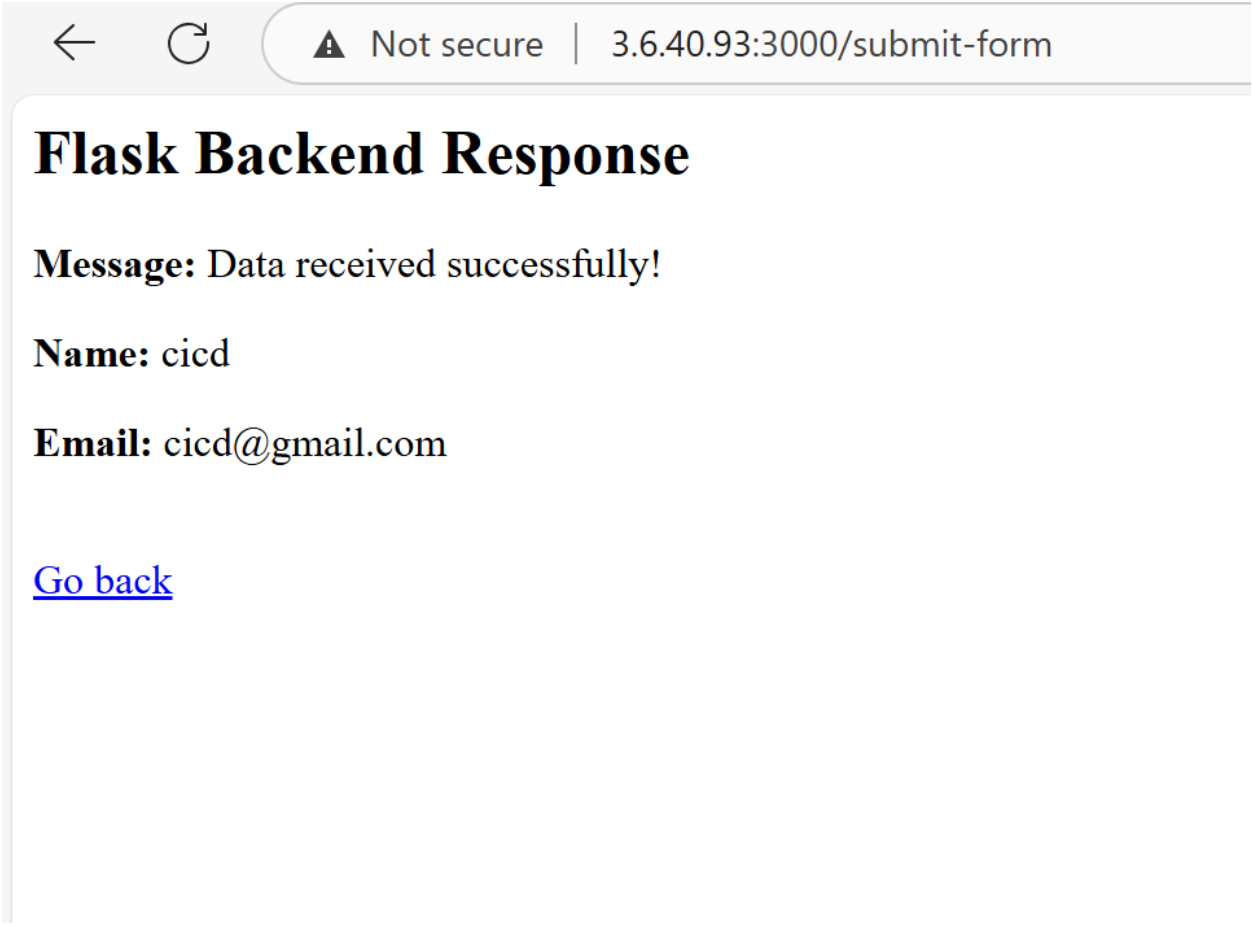
- Confirm: `curl http://localhost:3000` or `http://<EC2_PUBLIC_IP>:3000`.

(If you prefer **nginx** as reverse proxy: install nginx and proxy ports 80 -> 3000 and 5000. I can give nginx config if you want.)



Now open the ec2_public_ip:3000 we get

Fill the form and click on submit button



Step 8 — Put Jenkinsfile in each repo (I give exact files to copy & commit)

Flask Jenkinsfile (save to repo root as Jenkinsfile)

```
pipeline {
  agent any
  environment {
    APP_DIR = '/home/deployer/flask-app'
    PM2_NAME = 'flask-app'
```

```
    GIT_BRANCH = 'main' // change if needed
  }
  stages {
    stage('Checkout') {
      steps {
        // For "Pipeline script from SCM" Jenkinsfile will
already be checked out.
        // If Jenkinsfile is configured separately, use:
        // git branch: "${env.GIT_BRANCH}", url:
'git@github.com:YOU/flask-repo.git', credentialsId:
'GIT_SSH_CREDENTIALS_ID'
        echo "source code checked out by pipeline"
      }
    }

    stage('Install deps & Test') {
      steps {
        sh '''
          set -e
          python3 -m venv venv
          . venv/bin/activate
          pip install -r requirements.txt
          # run tests (optional)
          pytest || true
```

```
        '''
      }
    }

    stage('Deploy to server') {
      steps {
        sh '''
          set -e
          # sync workspace to deploy directory as deployer (jenkins allowed to run rsync as deployer via sudo)
          sudo -u deployer rsync -av --delete ${WORKSPACE}/ ${APP_DIR}/
          # install server-side deps and restart app under deployer
          sudo -u deployer bash -lc "cd ${APP_DIR} && . venv/bin/activate && pip install -r requirements.txt && pm2 restart ${PM2_NAME} || pm2 start gunicorn --name ${PM2_NAME} --bind 0.0.0.0:5000 'wsgi:app'"
        '''
      }
    }
  }

  post {
    success { echo "Flask deploy succeeded" }
```

```
    failure { echo "Flask deploy failed" }
  }
}
```

Express Jenkinsfile (save to repo root as Jenkinsfile)

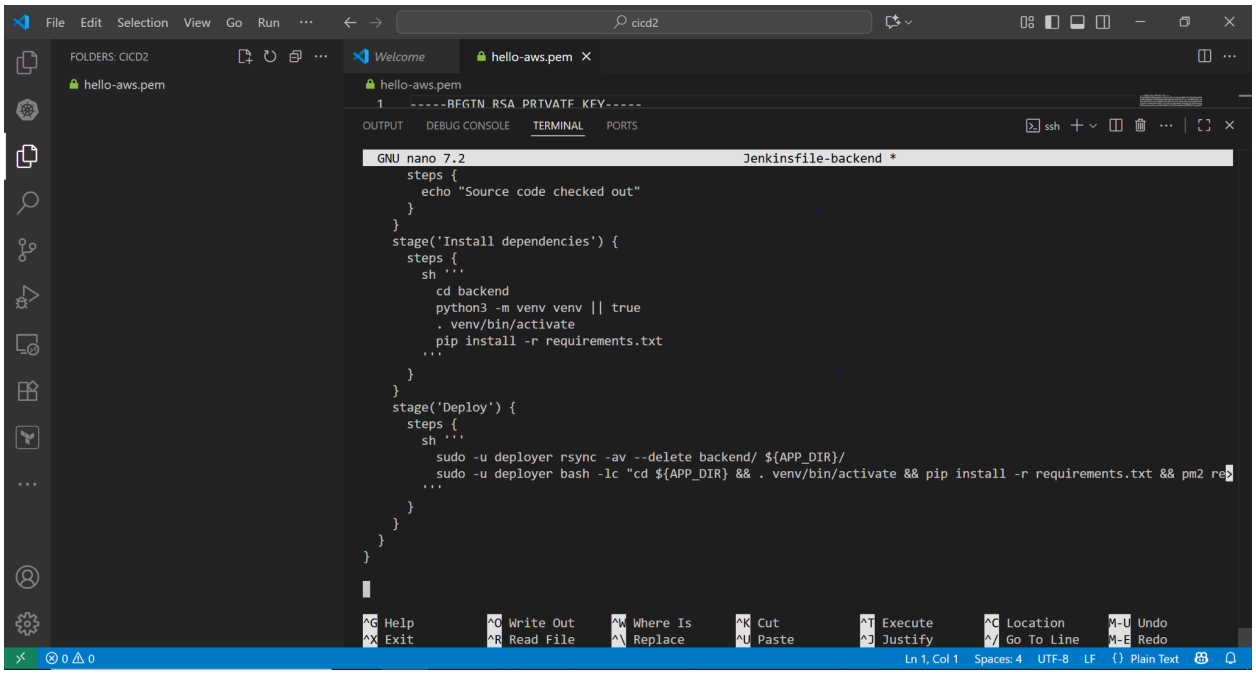
```
pipeline {
  agent any
  environment {
    APP_DIR = '/home/deployer/express-app'
    PM2_NAME = 'express-app'
    GIT_BRANCH = 'main'
  }
  stages {
    stage('Checkout') {
      steps { echo "code checked out" }
    }

    stage('Install & Test') {
      steps {
        sh '''
          set -e
          # install node deps inside workspace
```

```
          npm ci
          # optional tests
          npm test || true
        '''
      }
    }

    stage('Deploy') {
      steps {
        sh '''
          set -e
          sudo -u deployer rsync -av --delete ${WORKSPACE}/
${APP_DIR}/
          sudo -u deployer bash -lc "cd ${APP_DIR} && npm ci &&
pm2 restart ${PM2_NAME} || pm2 start npm --name ${PM2_NAME} --
start"
        '''
      }
    }
  }
  post {
    success { echo "Express deploy succeeded" }
    failure { echo "Express deploy failed" }
  }
```

```
}
```

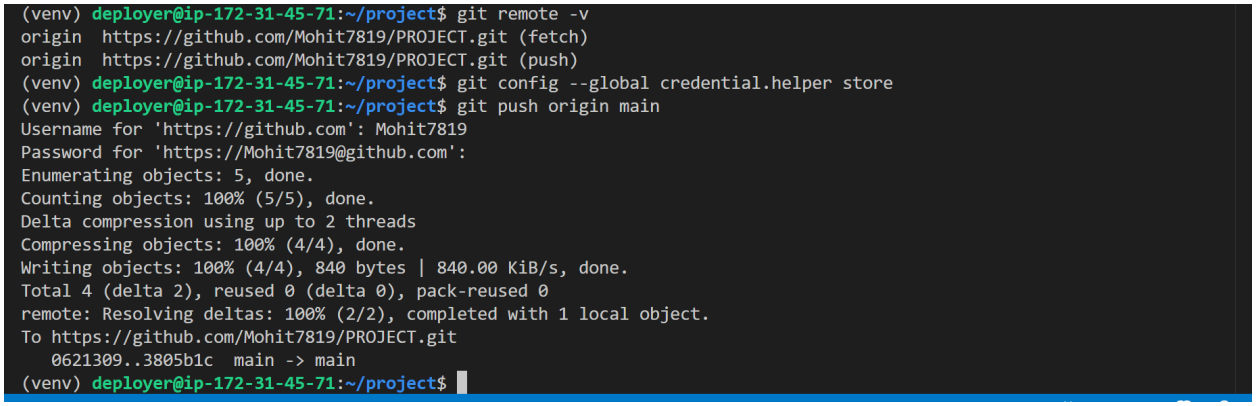
Important:

- Commit the appropriate `Jenkinsfile` to each repo root (same file name `Jenkinsfile`).
- Replace `YOU` and branch names as needed.
- `credentialsId` is only needed if Jenkins must authenticate to git; if you used the Jenkins SSH deploy key (plugin), checkout can use `git` step without credentials.



Creating two files

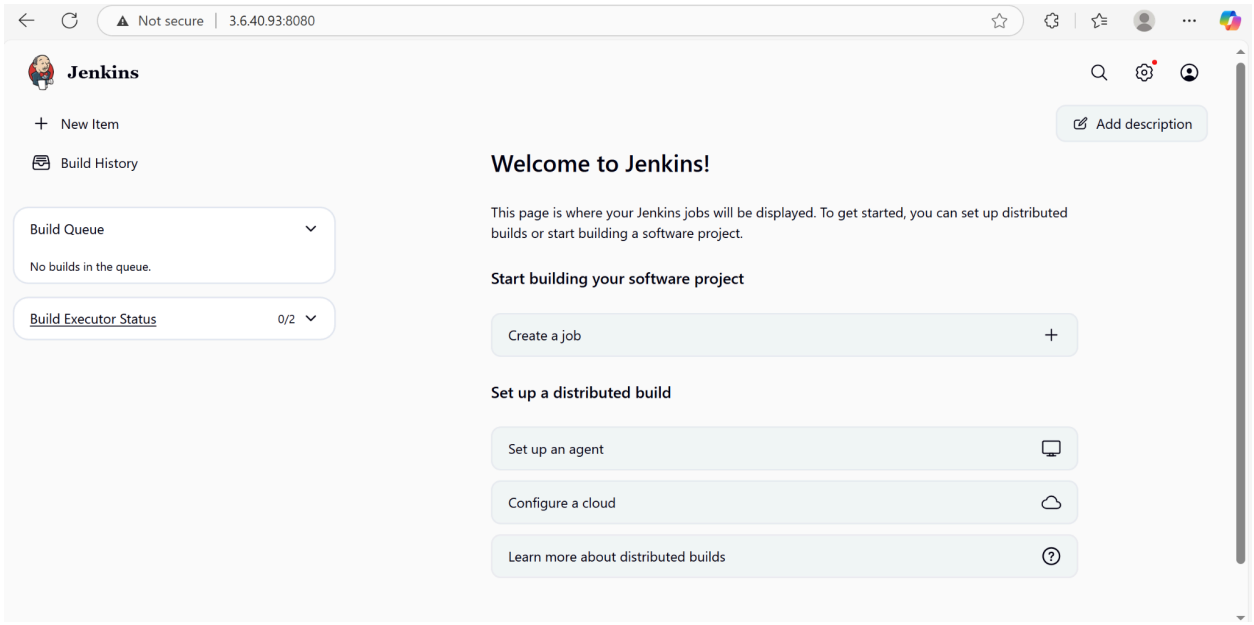
```
(venv) deployer@ip-172-31-45-71:~/project$ cd ~/project
(venv) deployer@ip-172-31-45-71:~/project$ nano Jenkinsfile-backend
(venv) deployer@ip-172-31-45-71:~/project$ nano Jenkinsfile-frontend
(venv) deployer@ip-172-31-45-71:~/project$ (venv) deployer@ip-172-31-45-71:~/project$
```



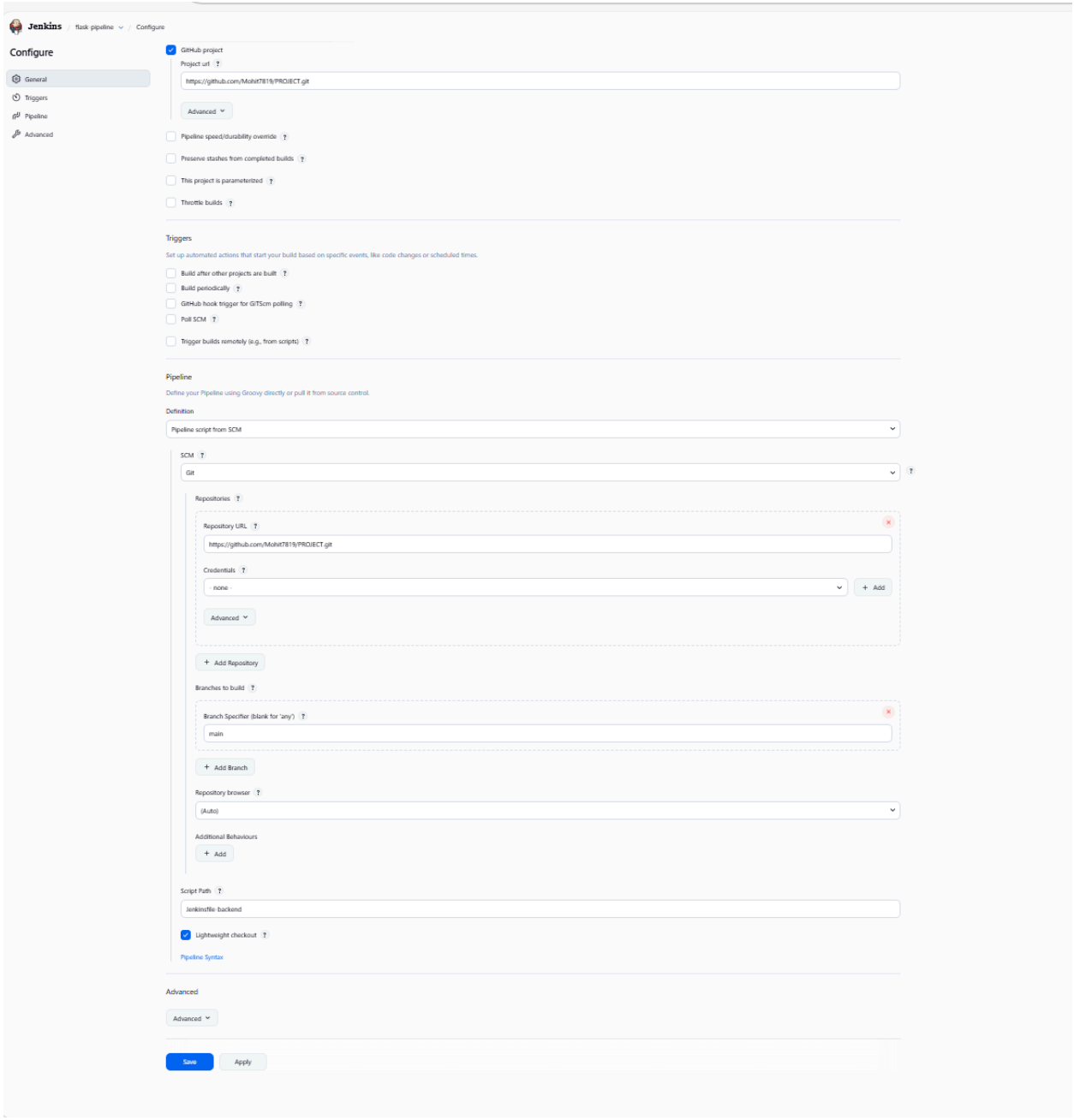
Create Jenkins Pipeline job (point to the repo Jenkinsfile)

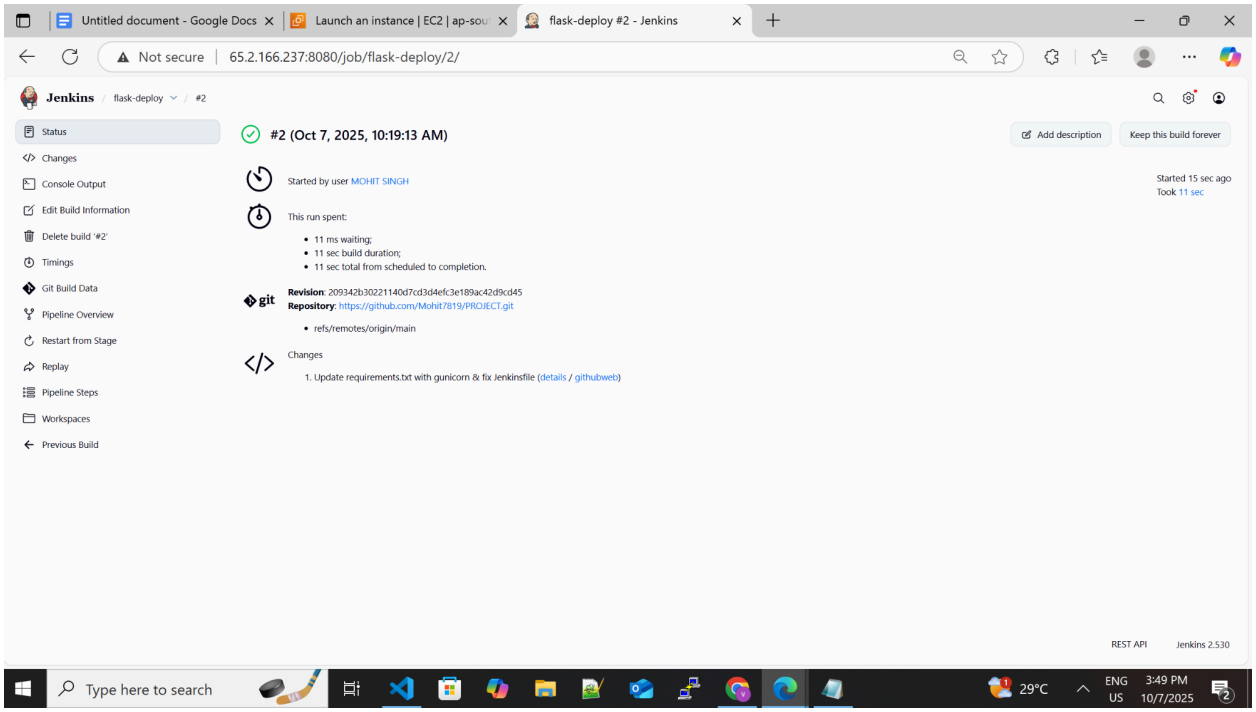
1. Jenkins UI → **New Item** → give name `flask-pipeline` → **Multibranch Pipeline** or **Pipeline**.
    - Easiest: choose **Pipeline**, then under **Definition** choose **Pipeline script from SCM** → SCM: **Git** → Repository URL `git@github.com:YOU/flask-repo.git` → Credentials: choose the SSH credentials you added → Branch: `main`.
    - Jenkins will pull the `Jenkinsfile` from the repo and run it.
2. Repeat for `express-pipeline`.

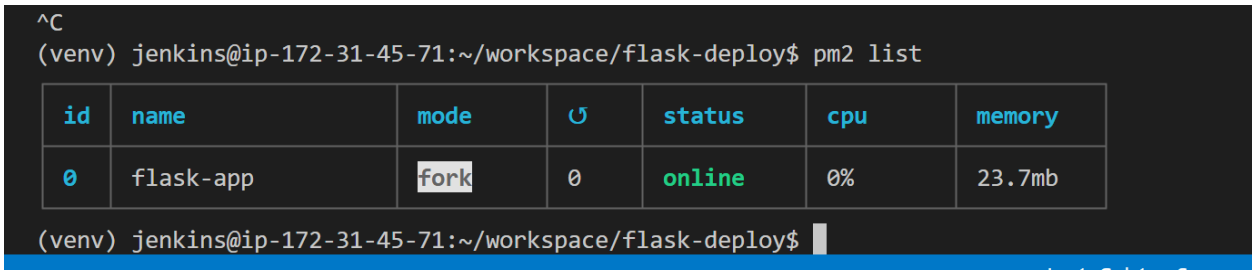
On job configuration, enable **GitHub hook trigger for GITScm polling** (checkbox) or use Generic Webhook Trigger plugin.


Not secure | 3.6.40.93:8080
Jenkins
New Item
Build History
Build Queue
No builds in the queue.
Build Executor Status
0/2
Add description
Welcome to Jenkins!
This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.
Start building your software project
Create a job
Set up a distributed build
Set up an agent
Configure a cloud
Learn more about distributed builds

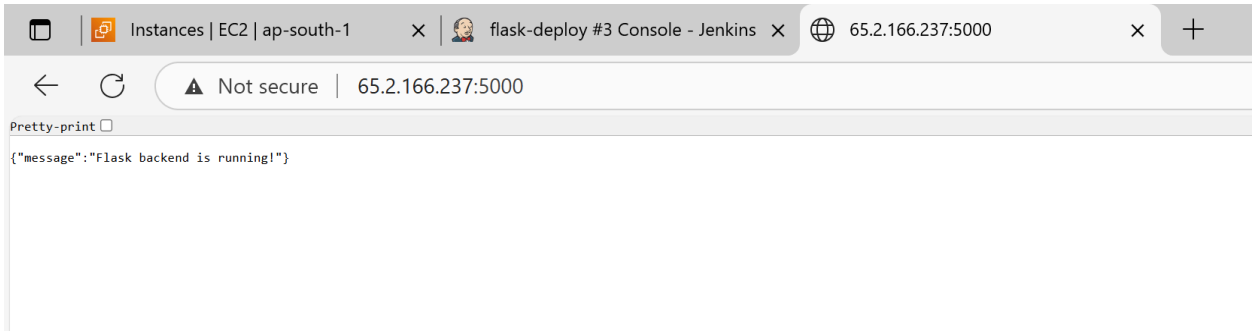
Click on new item



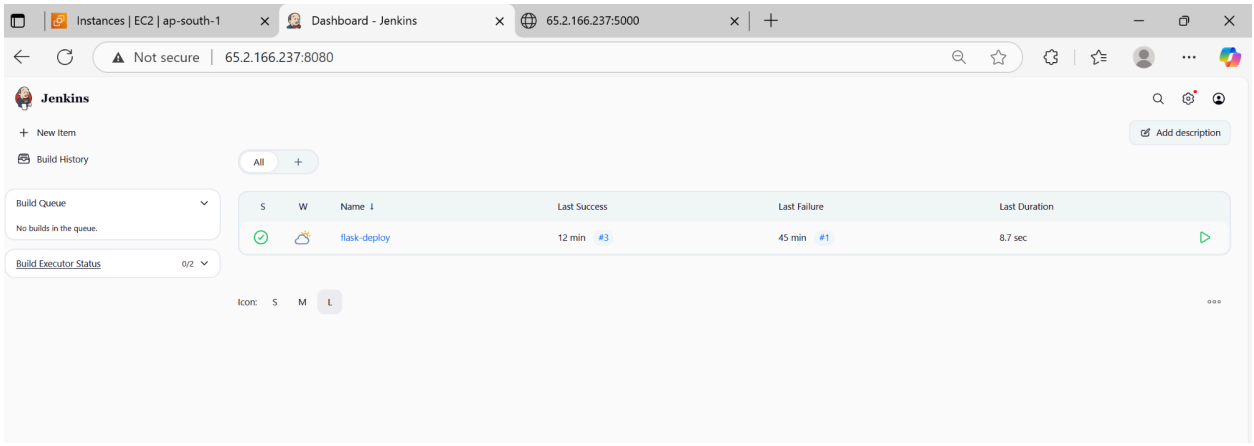


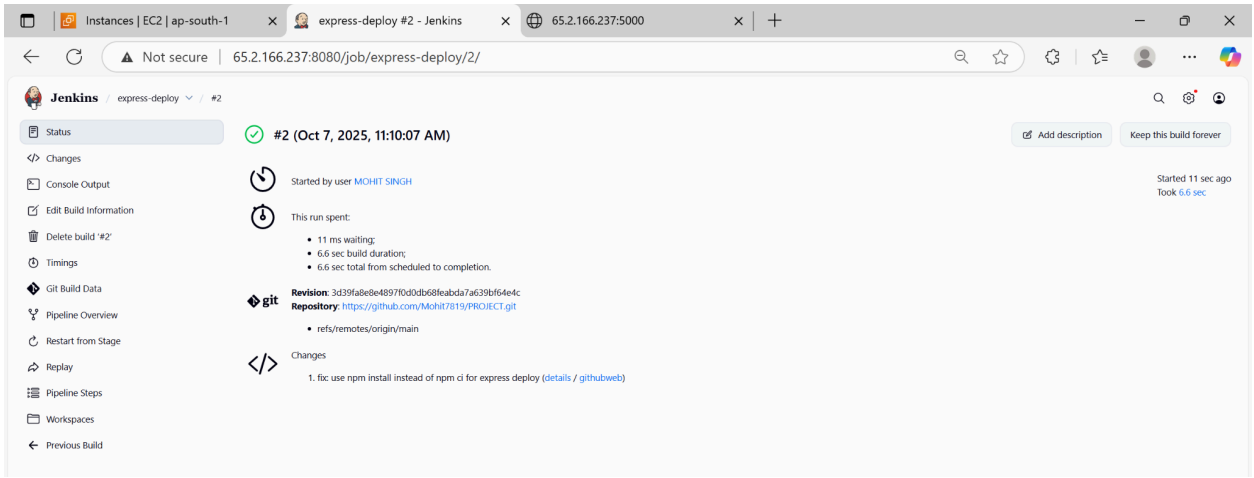


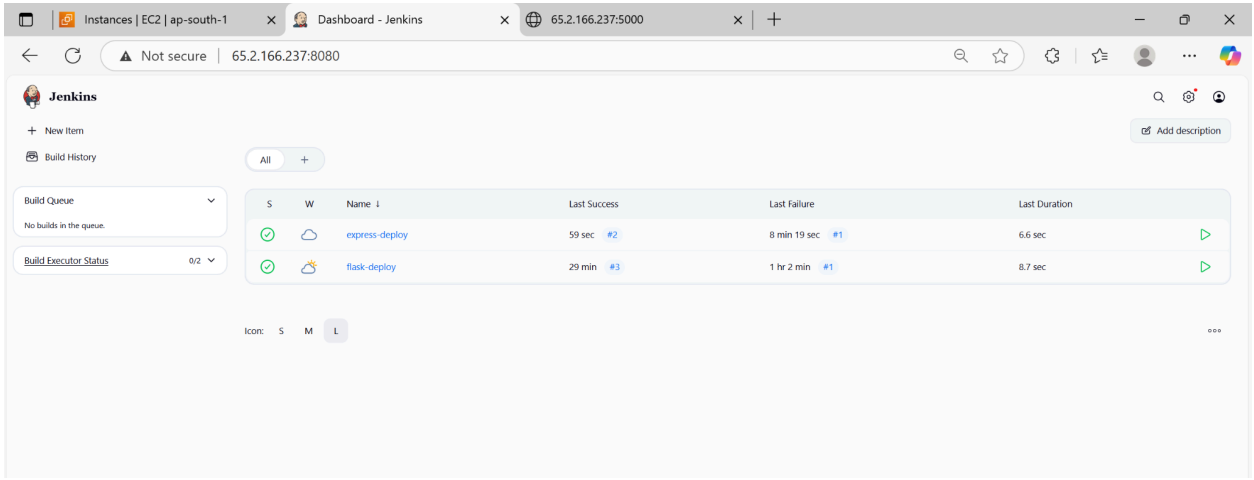
When go to browser we see our backend is running



Same steps for express-deploy jenkins click on new item







Now open http://65.2.166.237:3000 in browser we see

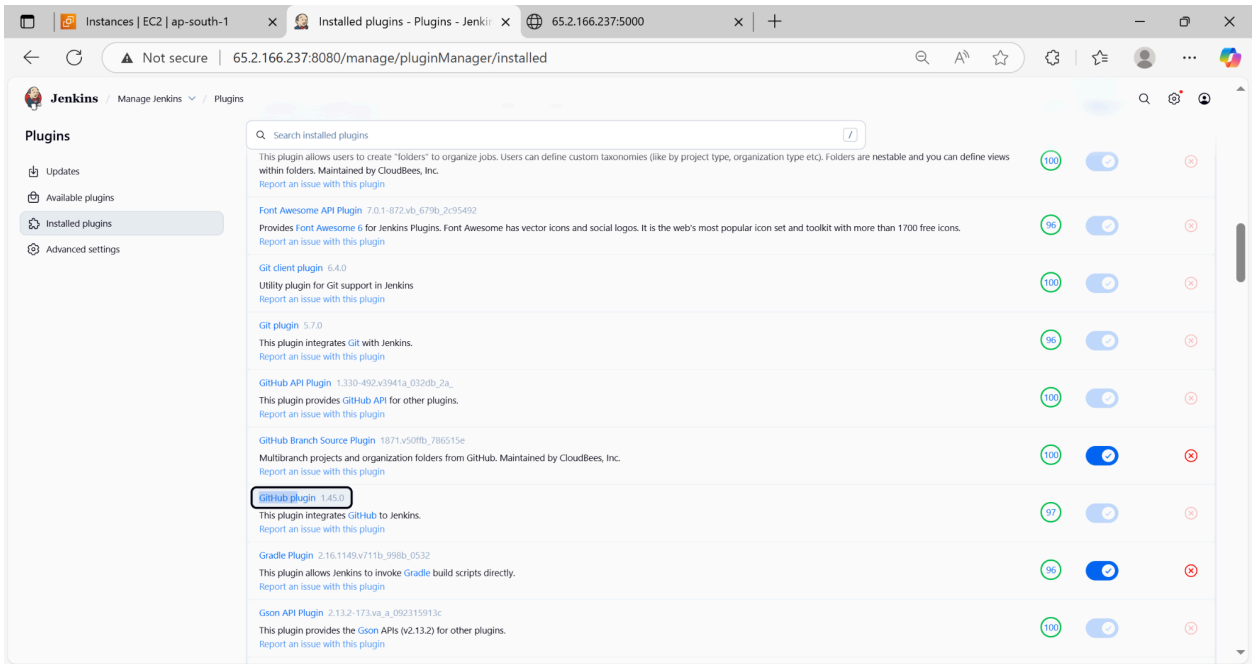
Fill form and click on submit button

Add GitHub webhook (so push triggers Jenkins)

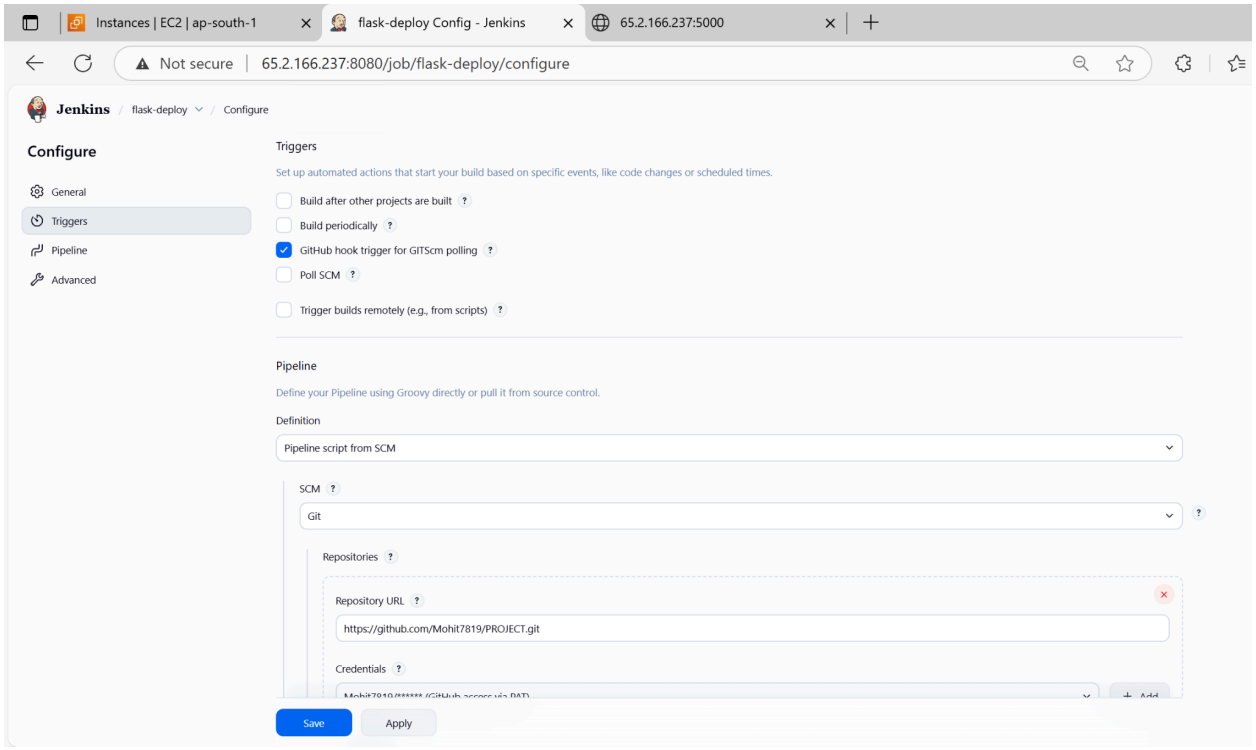
On each GitHub repo:

- Settings → Webhooks → Add webhook
    - Payload URL: `http://<EC2_PUBLIC_IP>:8080/github-webhook/`
    - Content type: `application/json`
    - Secret: optional (if you use it, configure the same secret in Jenkins GitHub plugin)
    - Which events: **Just the push event**
- Save.

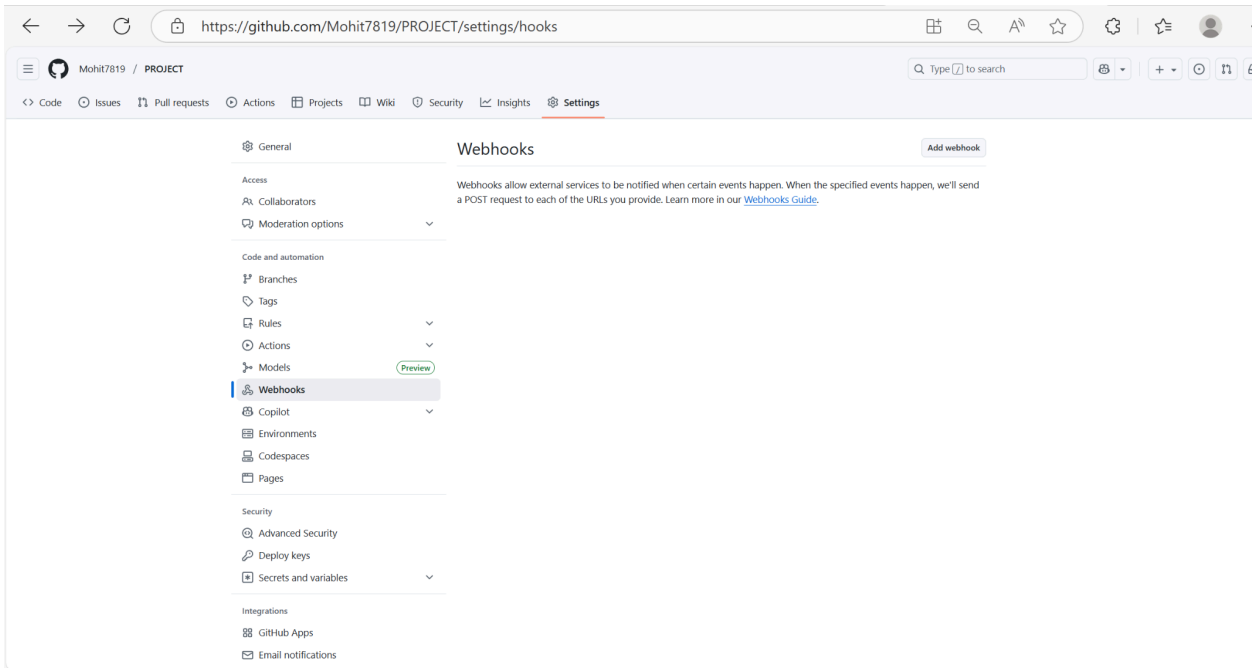
Now a `git push` to `main` should send webhook to Jenkins and trigger the pipeline.

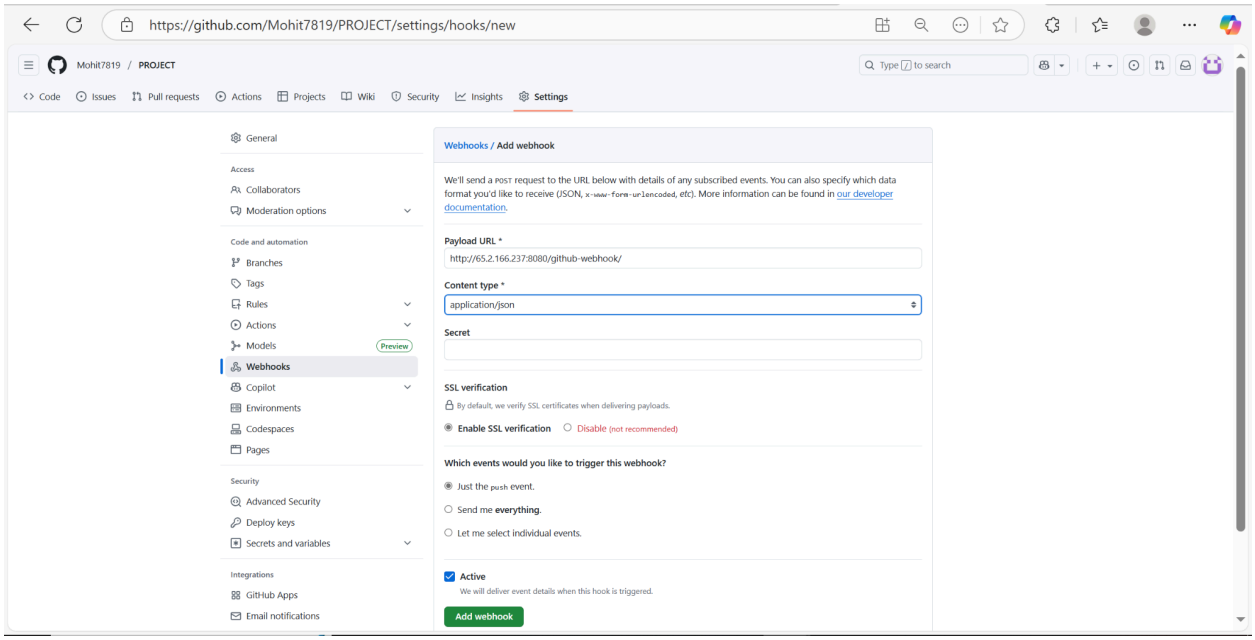


Click on config > triggers



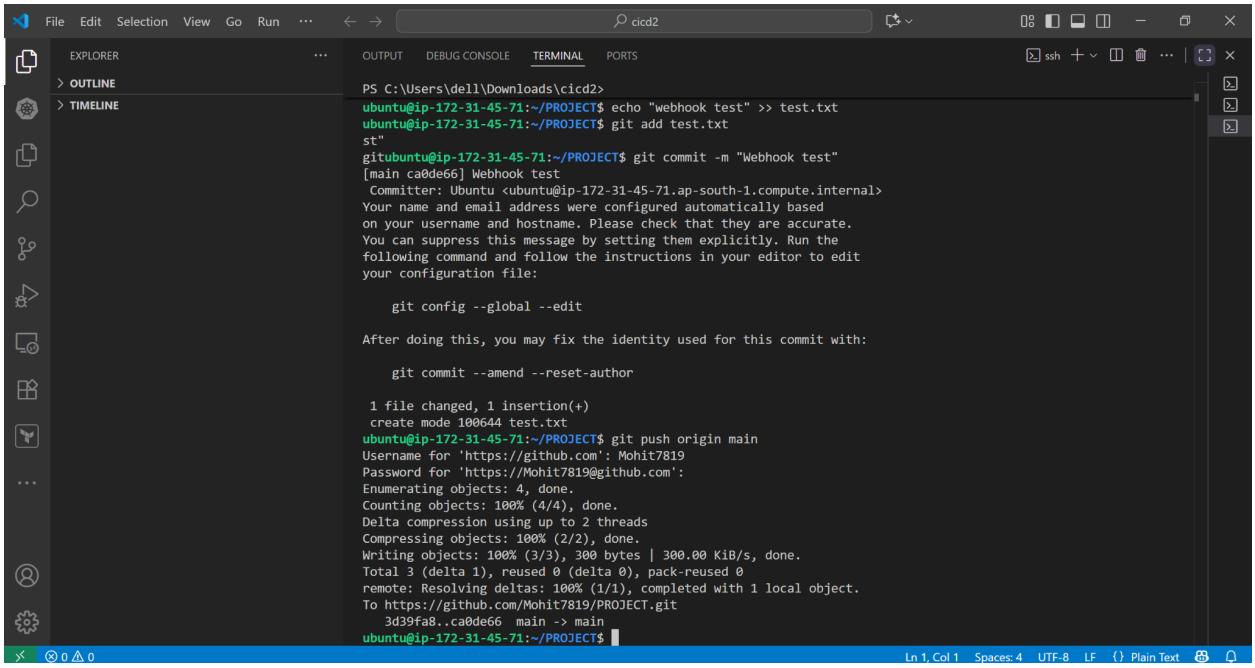
Now open github repo and open setting click on webhooks

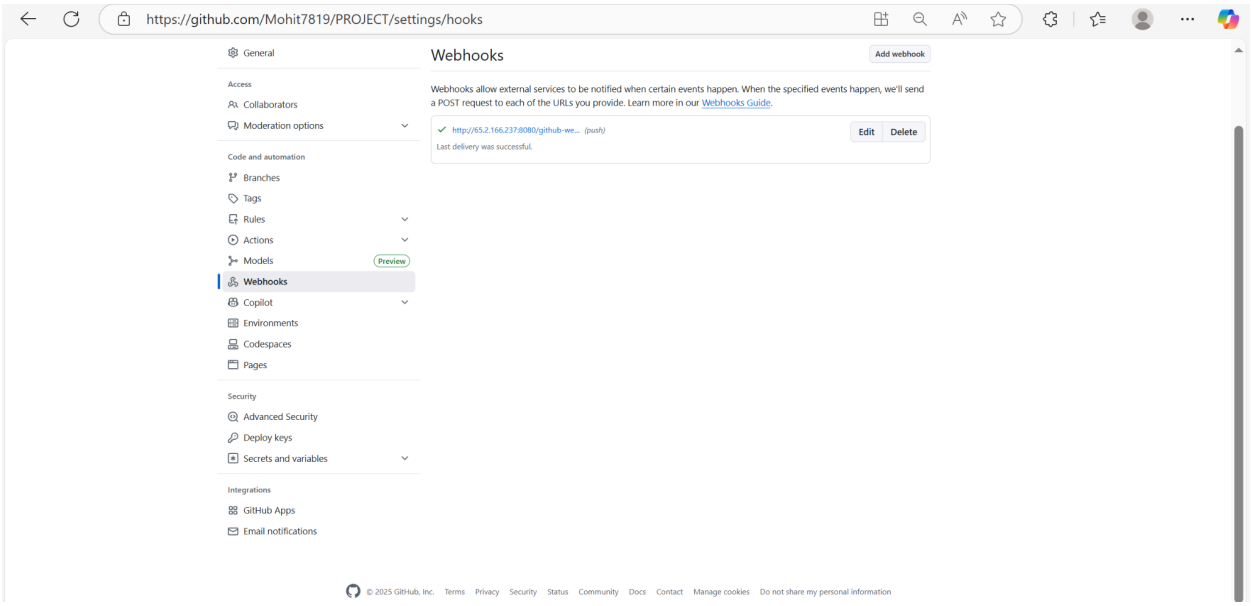


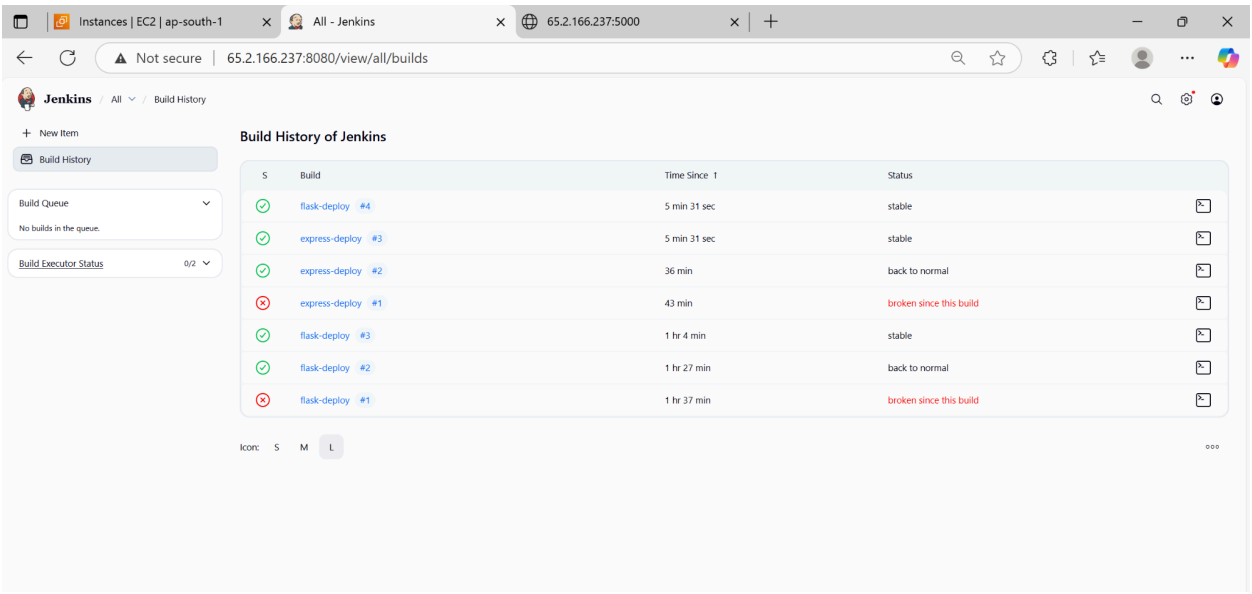


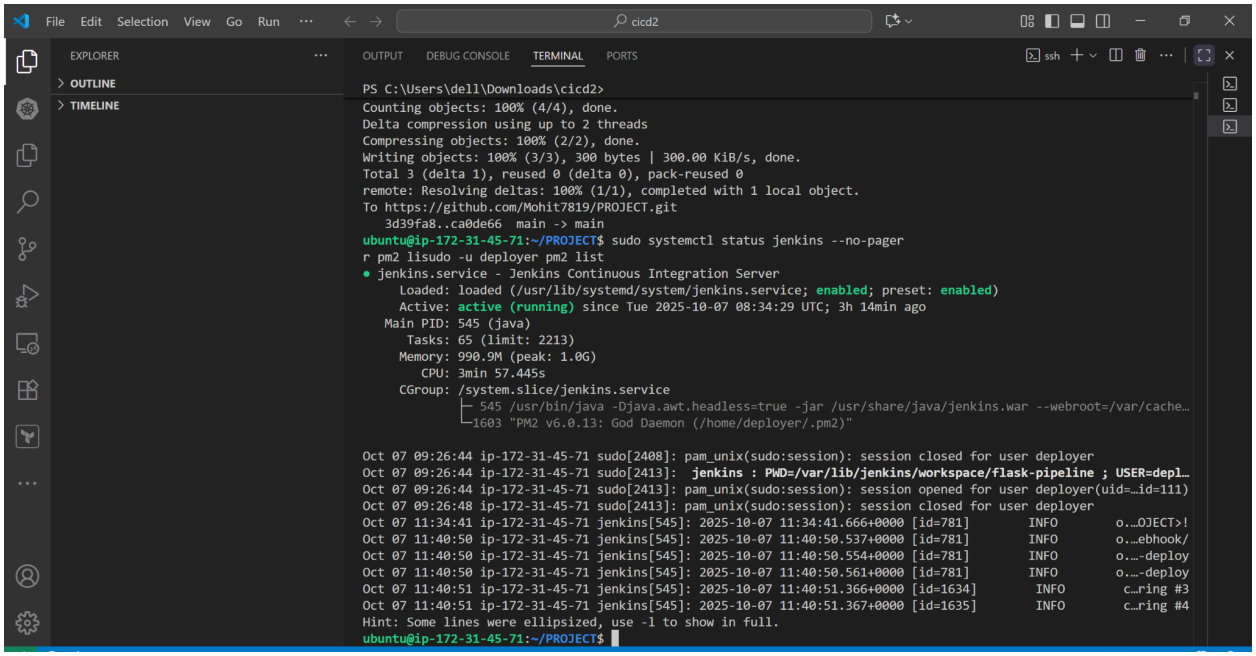
Now test webhooks

Creating a new file and push to git repo




https://github.com/Mohit7819/PROJECT/settings/hooks
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Models
Preview
Webhooks
Copilot
Environments
Codespaces
Pages
Security
Advanced Security
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Email notifications
Webhooks
Add webhook
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.
http://65.2.166.237:8080/github-we... (push)
Last delivery was successful.
Edit
Delete
© 2025 GitHub, Inc. Terms Privacy Security Status Community Docs Contact Manage cookies Do not share my personal information


Instances | EC2 | ap-south-1
All - Jenkins
65.2.166.237:5000
Not secure
65.2.166.237:8080/view/all/builds
Jenkins
All
Build History
New Item
Build History
Build Queue
No builds in the queue.
Build Executor Status
0/2
Build History of Jenkins
S
Build
Time Since ↑
Status
flask-deploy #4
5 min 31 sec
stable
express-deploy #3
5 min 31 sec
stable
express-deploy #2
36 min
back to normal
express-deploy #1
43 min
broken since this build
flask-deploy #3
1 hr 4 min
stable
flask-deploy #2
1 hr 27 min
back to normal
flask-deploy #1
1 hr 37 min
broken since this build
Icon: S M L


File Edit Selection View Go Run
cicd2
EXPLORER
OUTLINE
TIMELINE
OUTPUT DEBUG CONSOLE TERMINAL PORTS
ssh
PS C:\Users\dell\Downloads\cicd2>
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 300 bytes | 300.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Mohit7819/PROJECT.git
3d39fa8..ca0de66 main -> main
ubuntu@ip-172-31-45-71:~/PROJECT$ sudo systemctl status jenkins --no-pager
r pm2 lisudo -u deployer pm2 list
● jenkins.service - Jenkins Continuous Integration Server
Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
Active: active (running) since Tue 2025-10-07 08:34:29 UTC; 3h 14min ago
Main PID: 545 (java)
Tasks: 65 (limit: 2213)
Memory: 990.9M (peak: 1.0G)
CPU: 3min 57.445s
CGroup: /system.slice/jenkins.service
├─ 545 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache…
└─1603 "PM2 v6.0.13: God Daemon (/home/deployer/.pm2)"
Oct 07 09:26:44 ip-172-31-45-71 sudo[2408]: pam_unix(sudo:session): session closed for user deployer
Oct 07 09:26:44 ip-172-31-45-71 sudo[2413]: jenkins : PWD=/var/lib/jenkins/workspace/flask-pipeline ; USER=depl…
Oct 07 09:26:44 ip-172-31-45-71 sudo[2413]: pam_unix(sudo:session): session opened for user deployer(uid=…id=111)
Oct 07 09:26:48 ip-172-31-45-71 sudo[2413]: pam_unix(sudo:session): session closed for user deployer
Oct 07 11:34:41 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:34:41.666+0000 [id=781] INFO o.…OJECT>!
Oct 07 11:40:50 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:40:50.537+0000 [id=781] INFO o.…ebhook/
Oct 07 11:40:50 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:40:50.554+0000 [id=781] INFO o.…-deploy
Oct 07 11:40:50 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:40:50.561+0000 [id=781] INFO o.…-deploy
Oct 07 11:40:51 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:40:51.366+0000 [id=1634] INFO c…ring #3
Oct 07 11:40:51 ip-172-31-45-71 jenkins[545]: 2025-10-07 11:40:51.367+0000 [id=1635] INFO c…ring #4
Hint: Some lines were ellipsized, use -l to show in full.
ubuntu@ip-172-31-45-71:~/PROJECT$