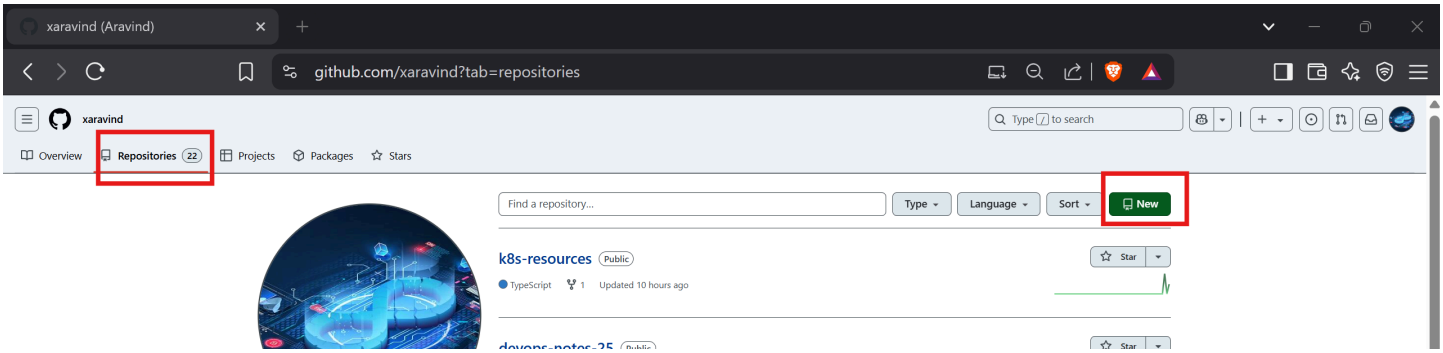
Full Project Setup Guide (GitHub → Docker → EC2 → EKS)

This guide walks through a real-world Kubernetes project from scratch — covering GitHub, Docker, EC2, private registry usage, and deployment on EKS using Kubernetes manifests.

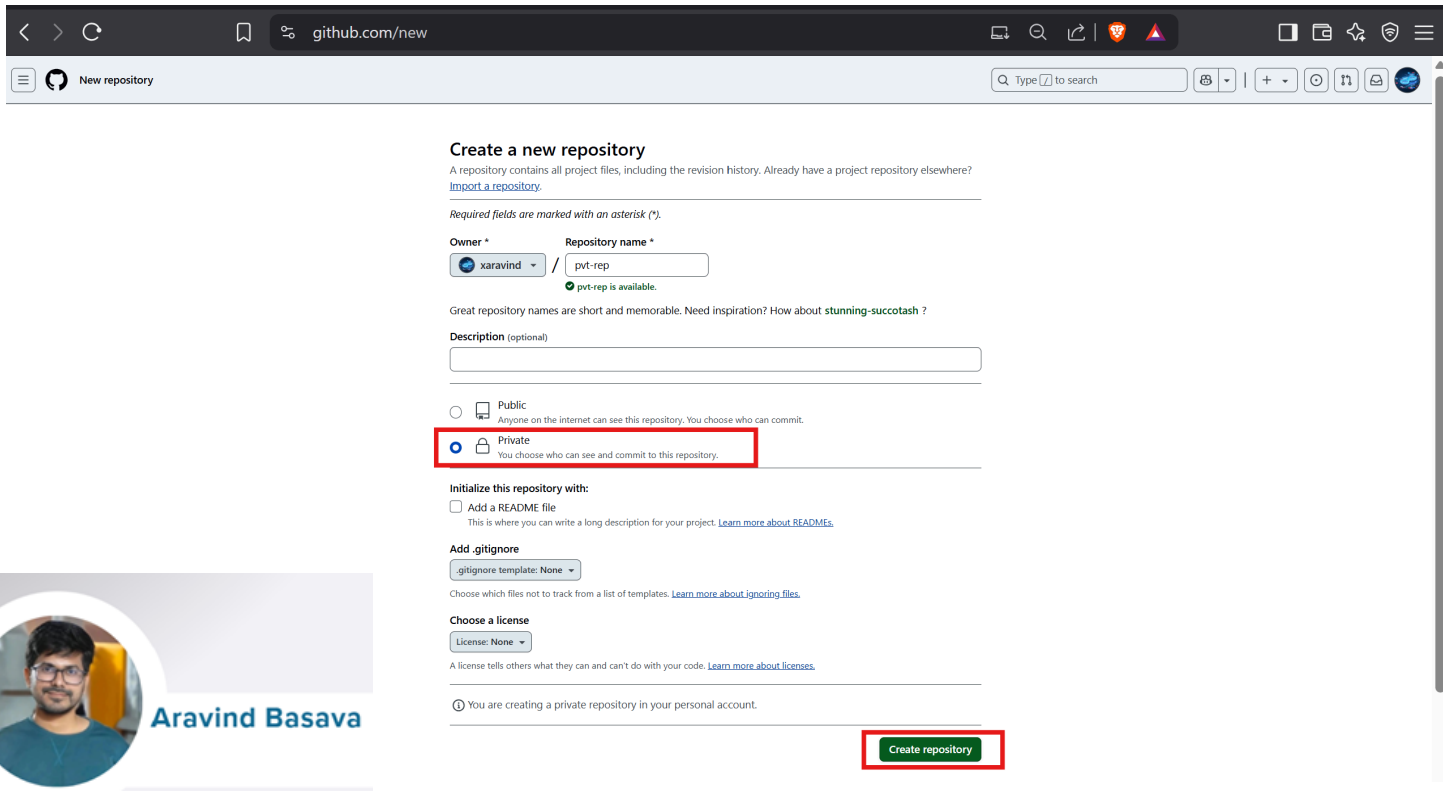
✅ First Stage: GitHub Setup

1. **Login** to your GitHub account.

`

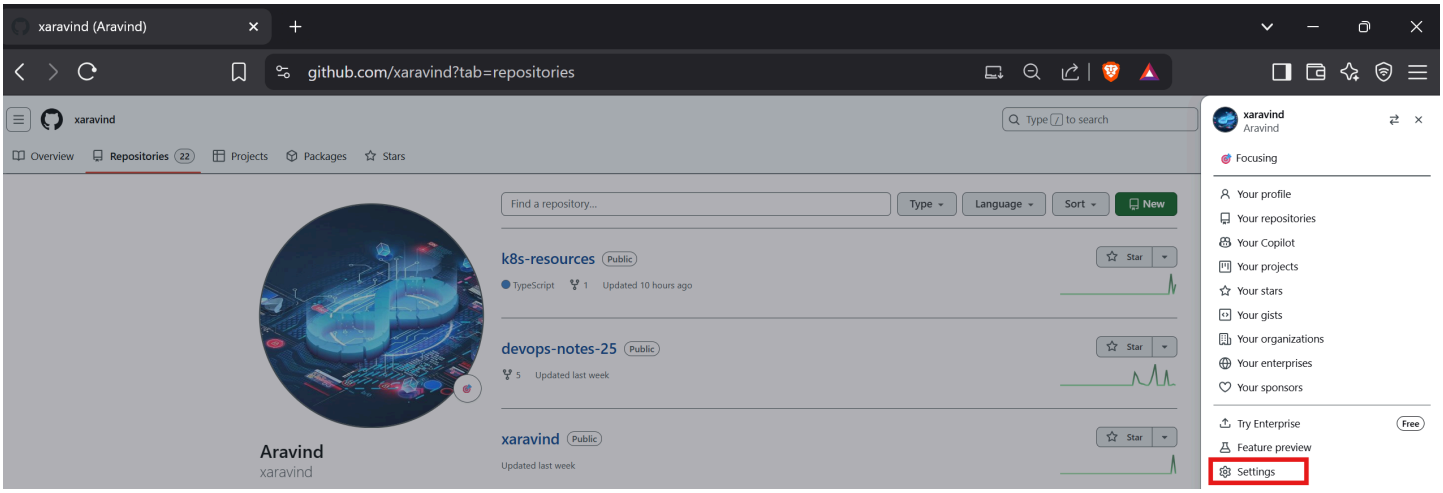


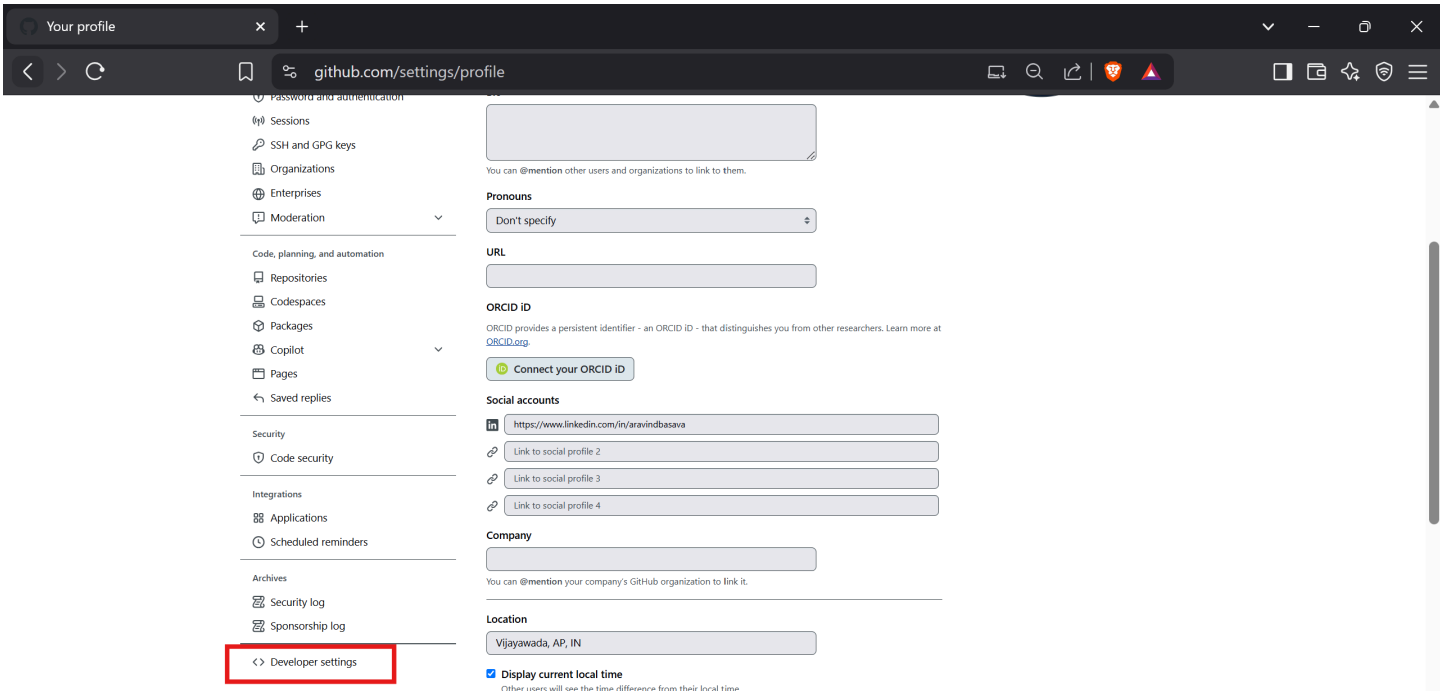
2. **Create a private repository**.

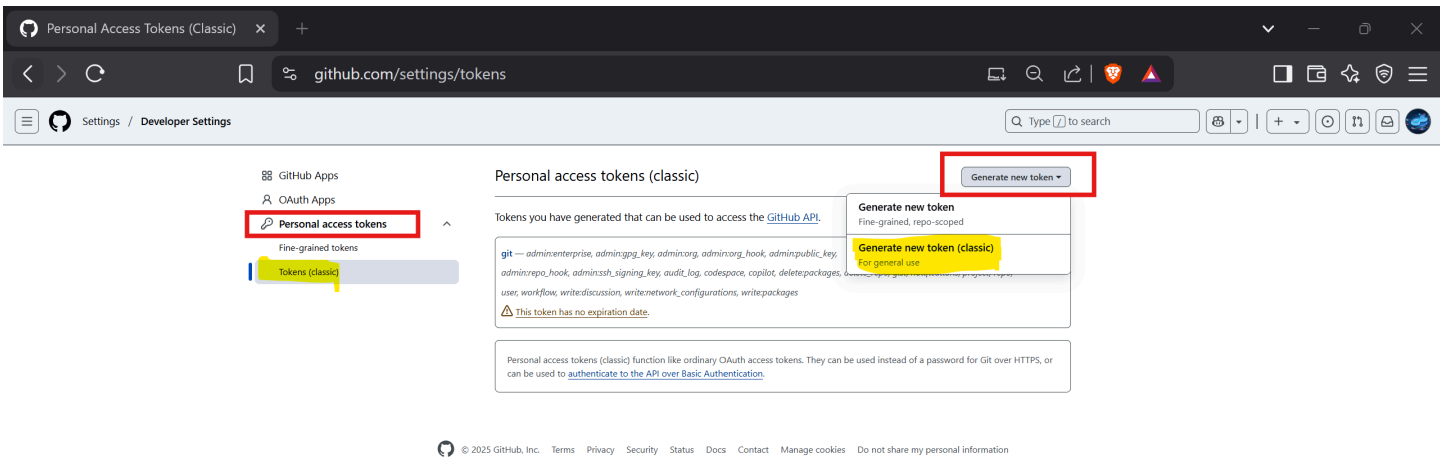


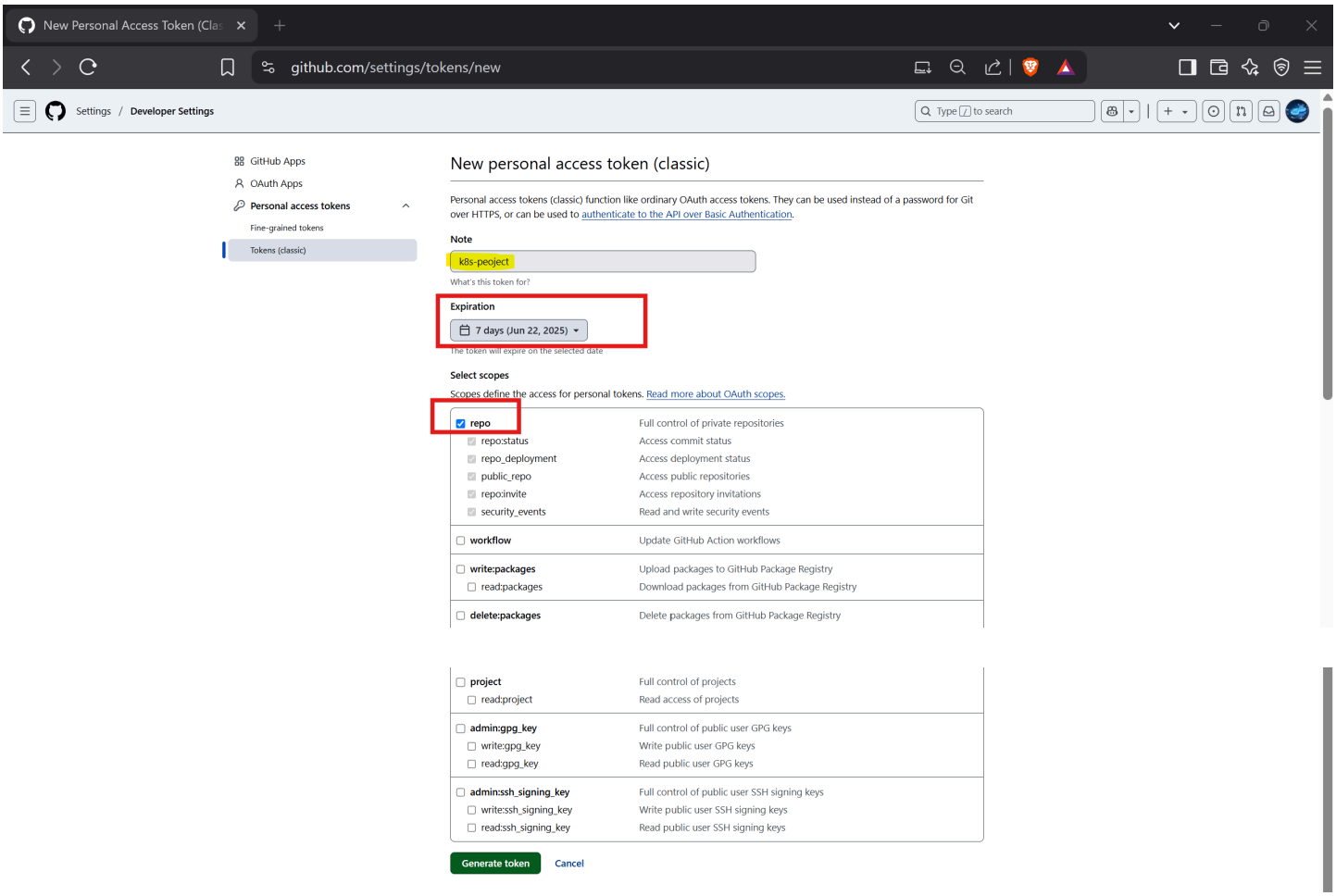
3. Go to the top right of your profile → `Settings` → `Developer Settings` → `Personal Access Tokens (classic)` .

- Click **Generate new token (classic)**.
- Select expiry, **grant only `repo` access**, and generate the token.
- > ⚠️ **Note:** Never share your credentials or tokens with anyone.

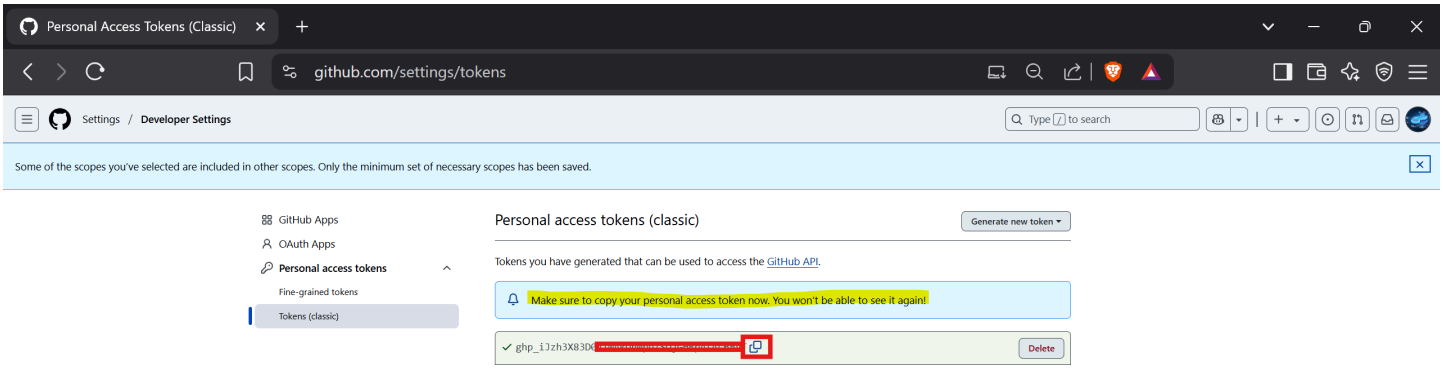




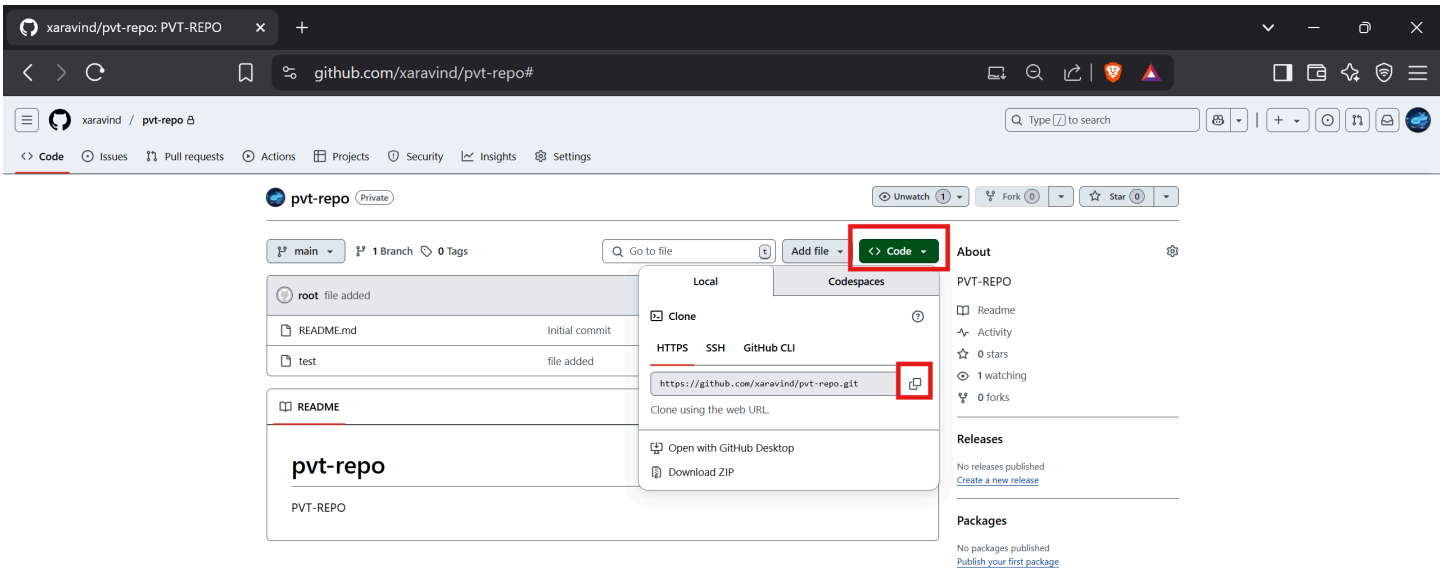




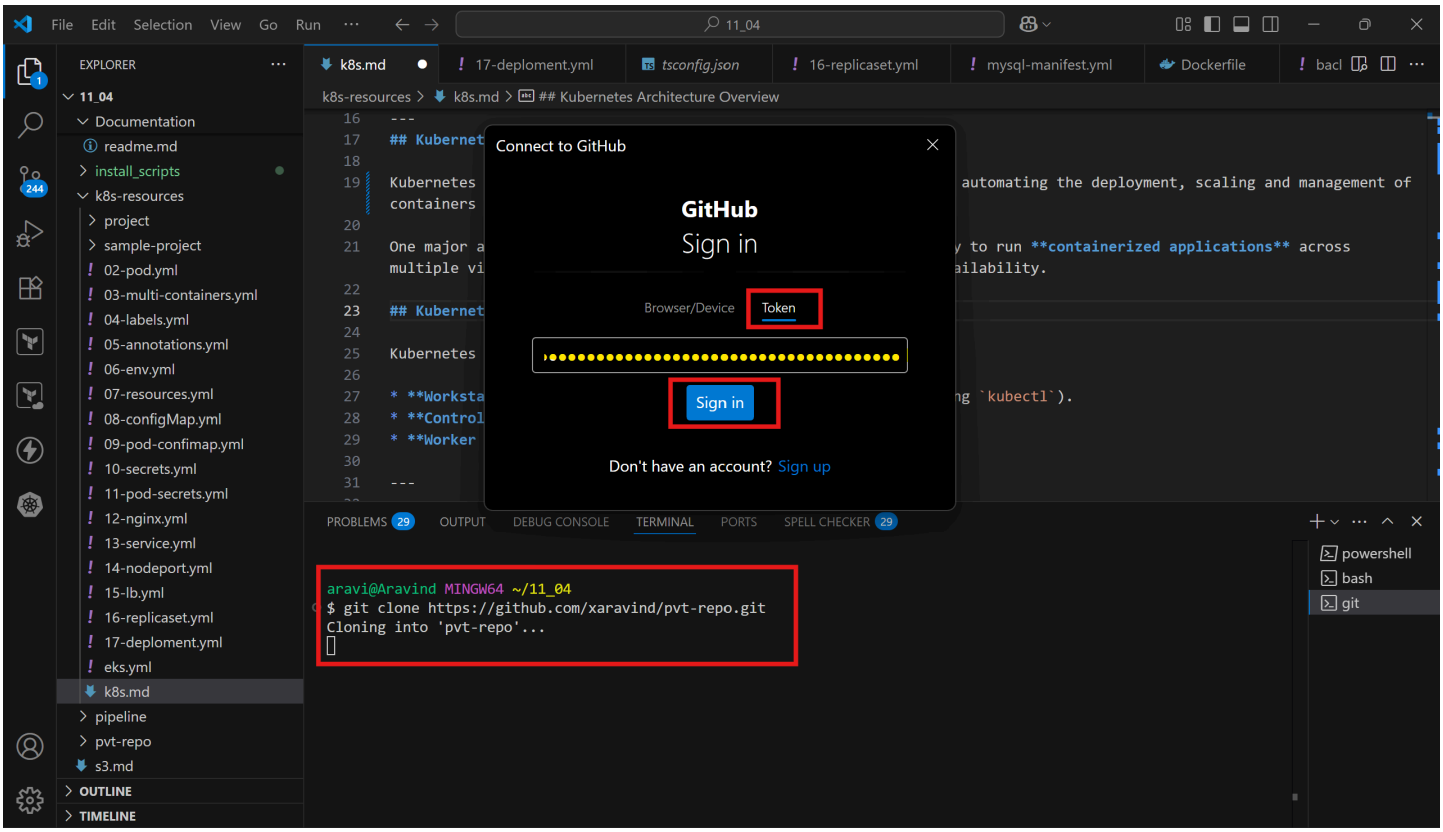
4. **Copy the token** and save it somewhere safe.



5. Go to your private repository and **copy the Git URL**.

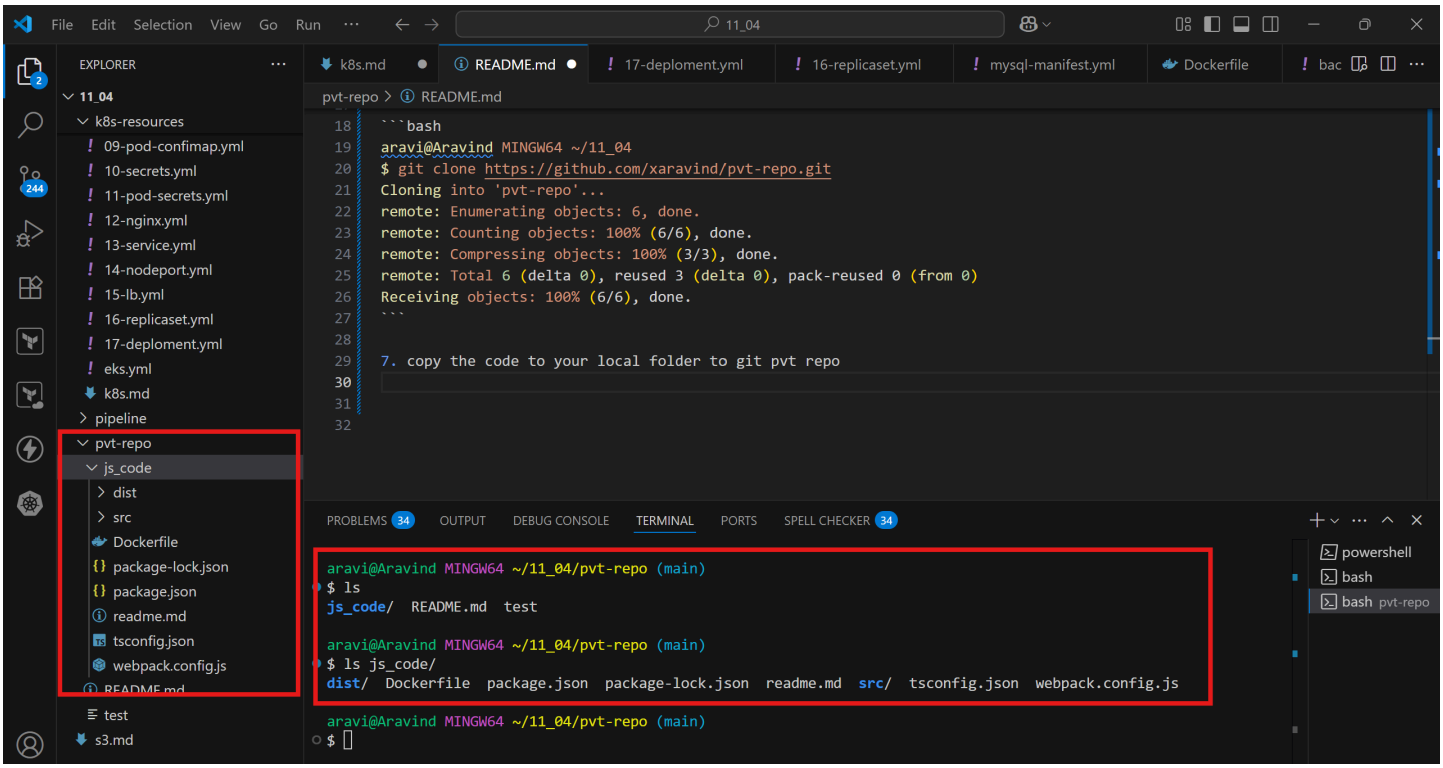


6. Open your **VS Code or local terminal** and **clone the repo using the token**.

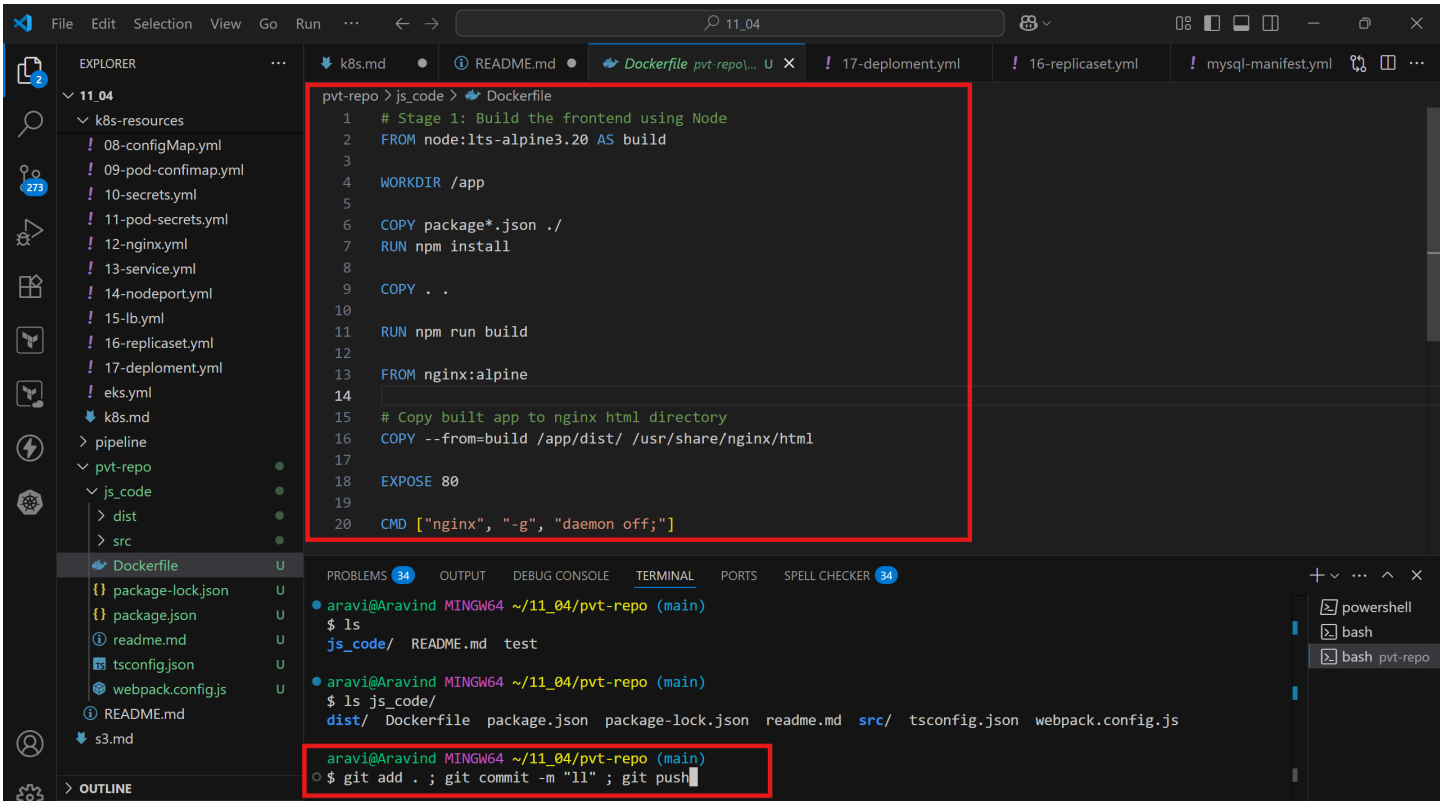



```bash
aravi@Aravind MINGW64 ~/11_04
$ git clone https://github.com/xaravind/pvt-repo.git
Cloning into 'pvt-repo'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
```

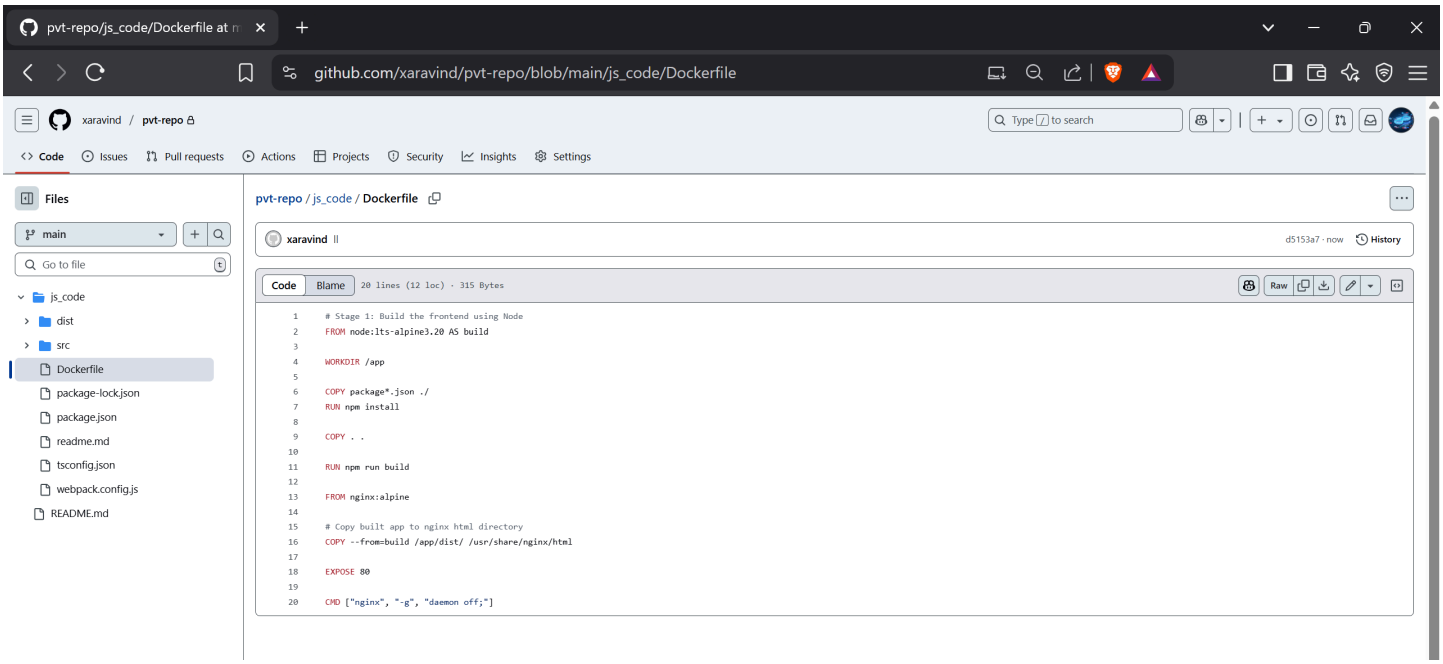
7. Copy your project code into the **cloned Git folder**.



8. Write a `Dockerfile` in your project, save it, and **push the code**.



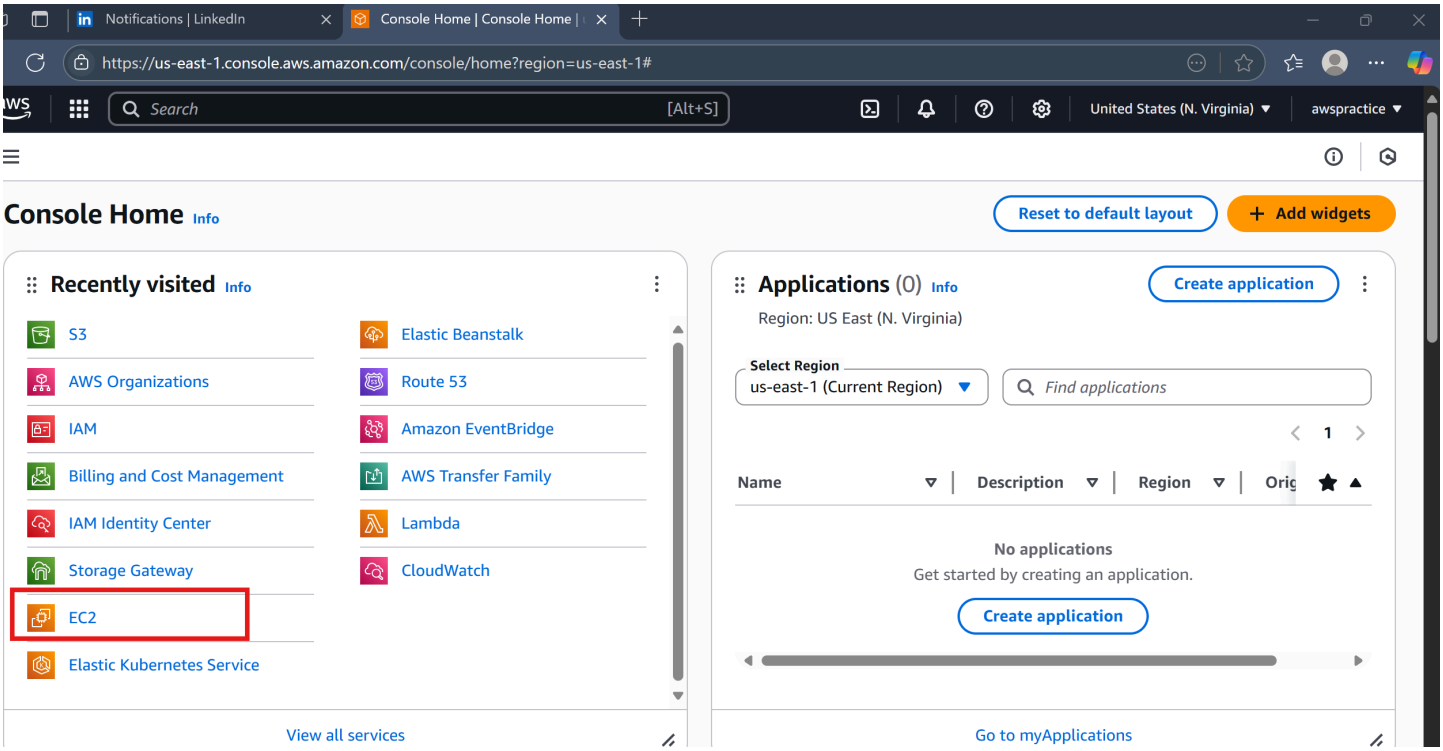
9. Open GitHub in your browser and check the files. You should see everything updated.



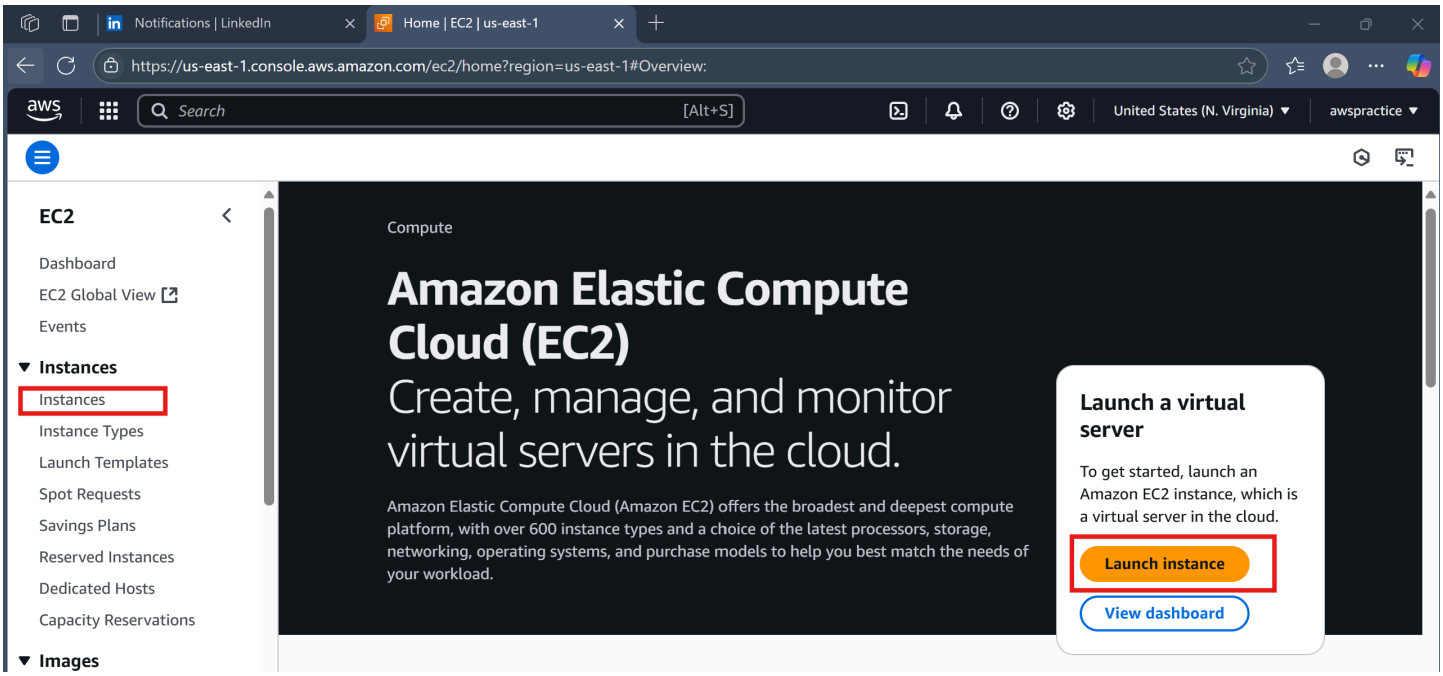
> ✅ We have now created a GitHub repo, generated a PAT key, cloned the repo locally, added code, created a Dockerfile, and pushed files to GitHub.

✅ Second Stage: Launch EC2 in AWS

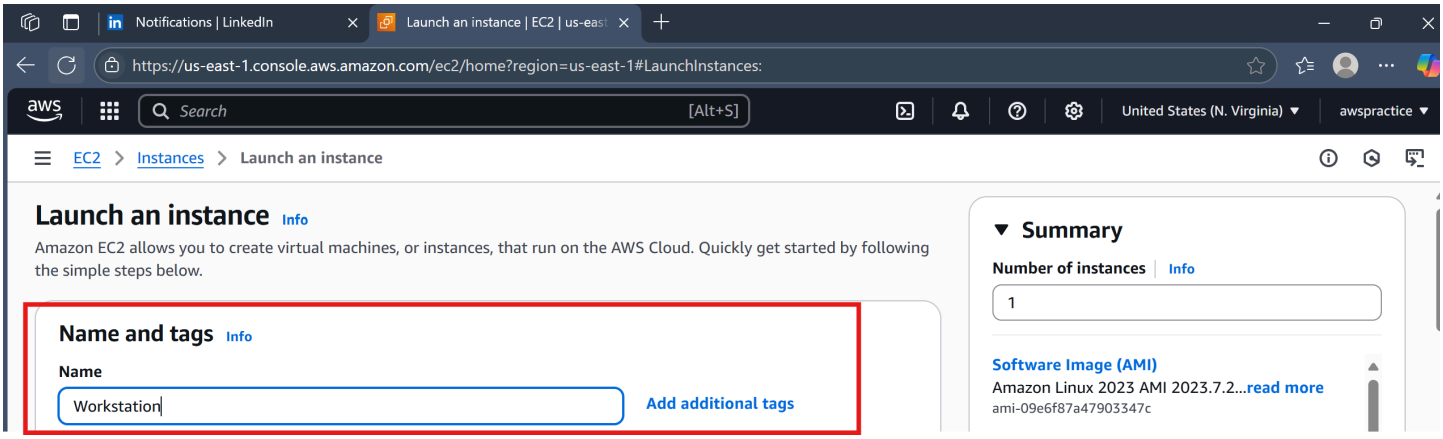
1. Login to your **AWS account**.

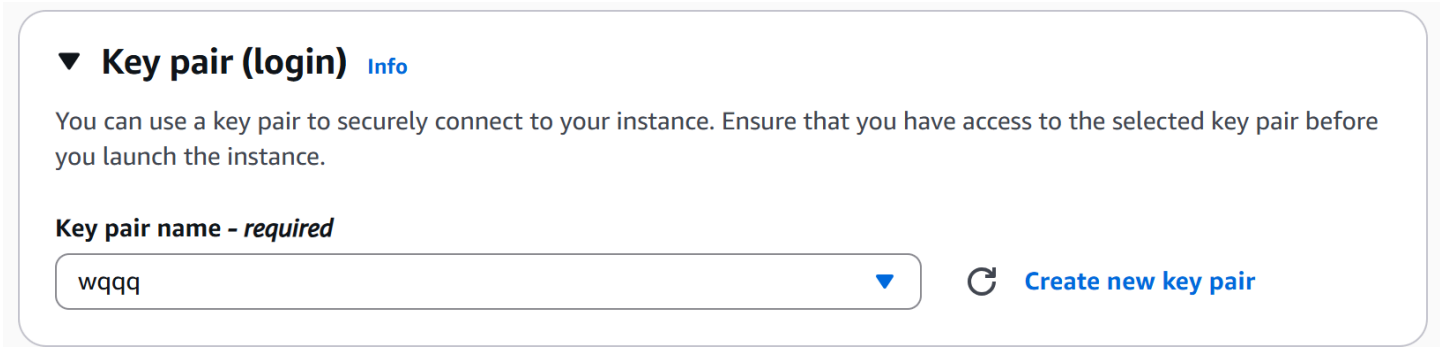


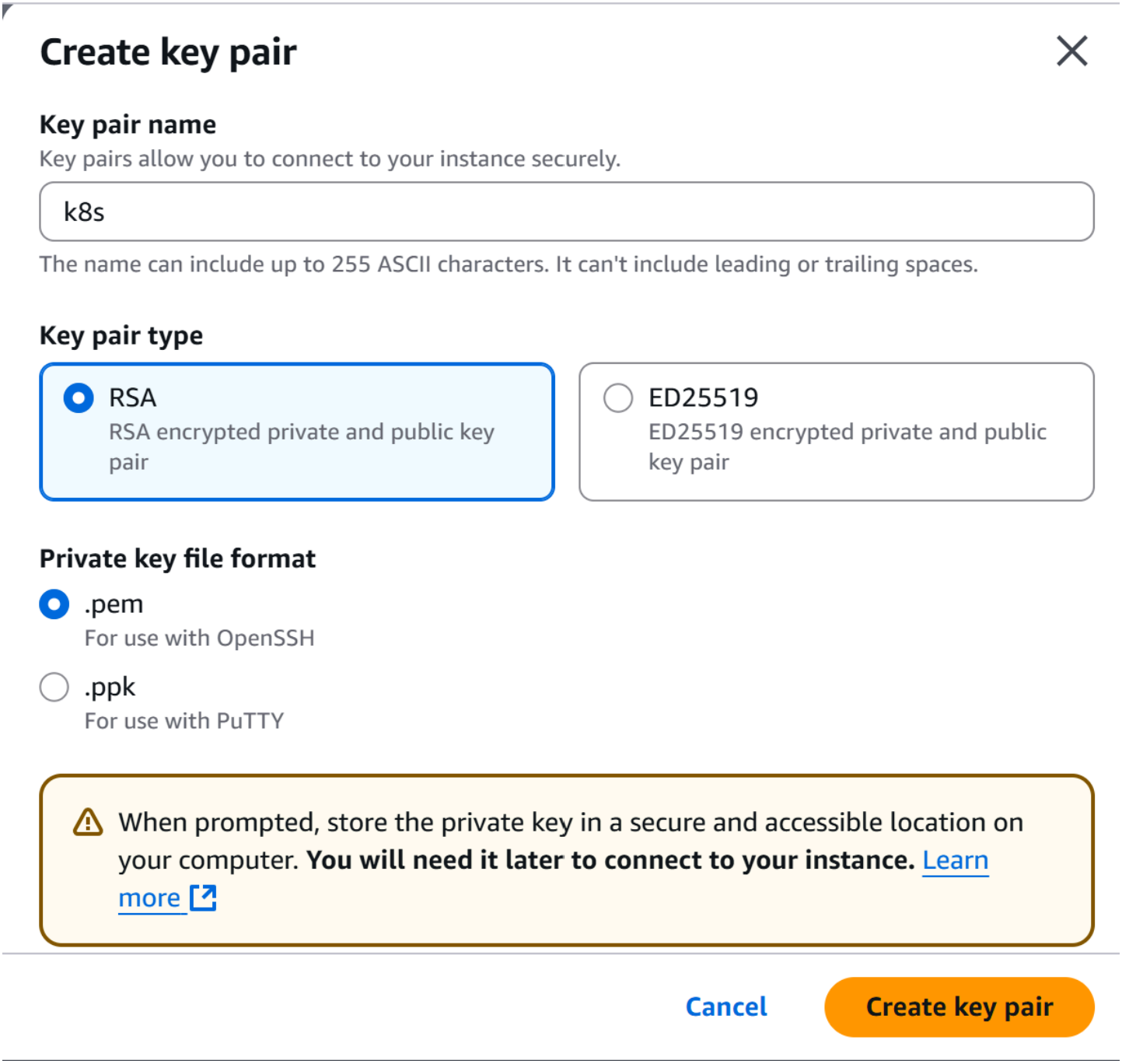
2. Go to **EC2** → **Instances** → **Launch Instance**.



3. Provide a name, **create a key pair**, and leave other settings as default.



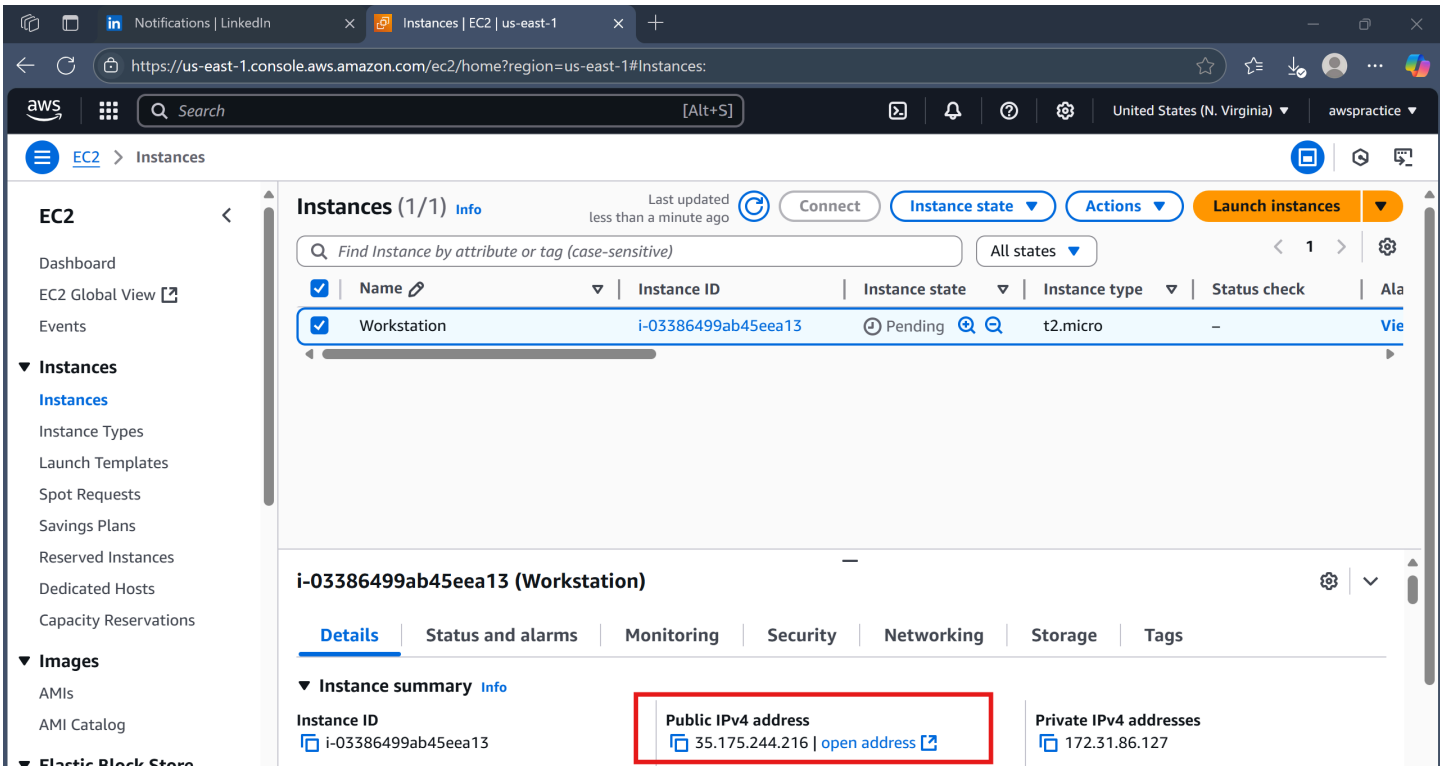


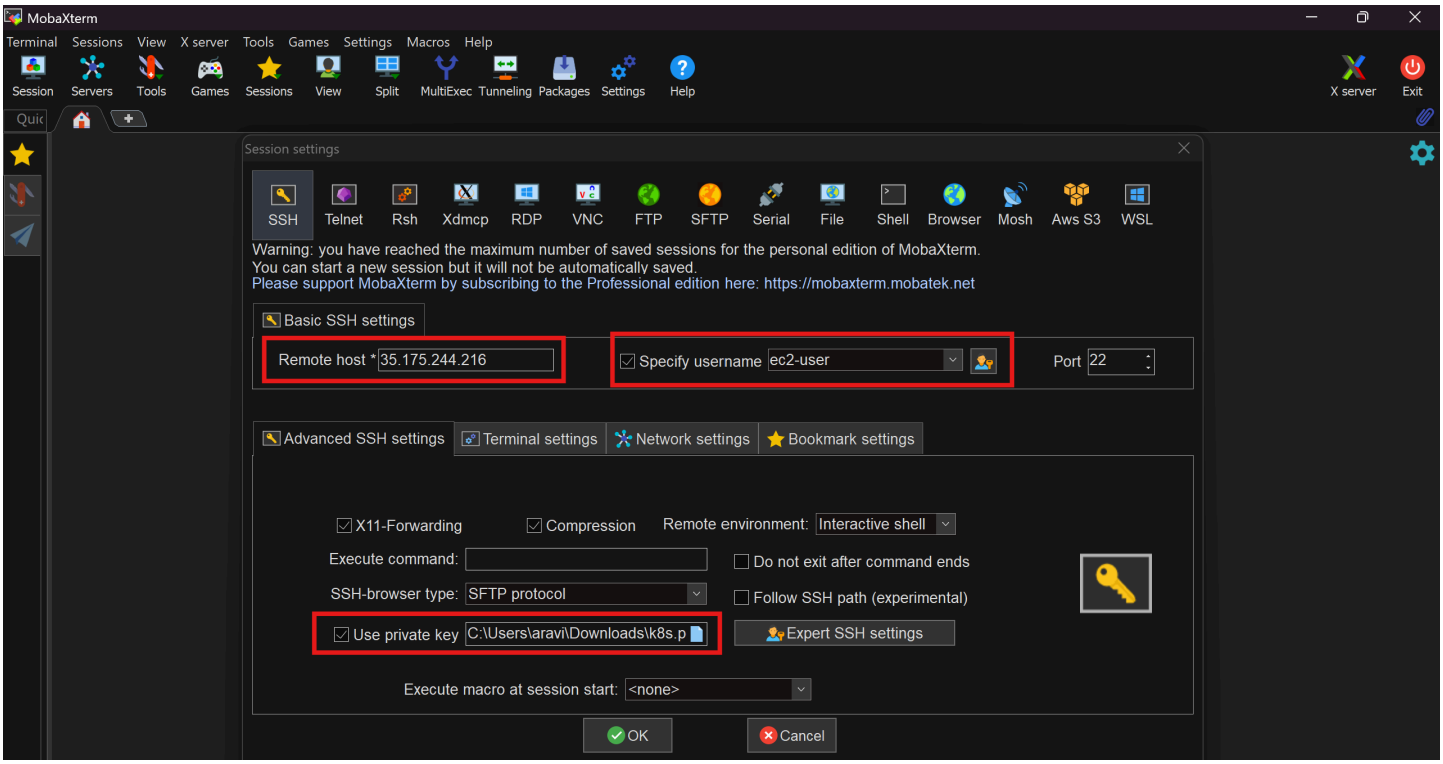


Keep all other settings as **default**. This VM will be used to build images, create EKS clusters, and communicate with the cluster.

4. Select **Amazon Linux AMI** (RHEL-based) and launch.

5. Copy the public IP, and connect via **Putty/MobaXterm** using your PEM key.





6. Switch to root user and **update the server**:

```
sudo su -
yum update -y
```

> ✅ In this stage, we created an EC2 instance, generated a `.pem` key, launched the instance with default settings, connected to it, and updated the server.

✅ Third Stage: Docker Setup & Push Image

1. Install Git and Docker, then start Docker:

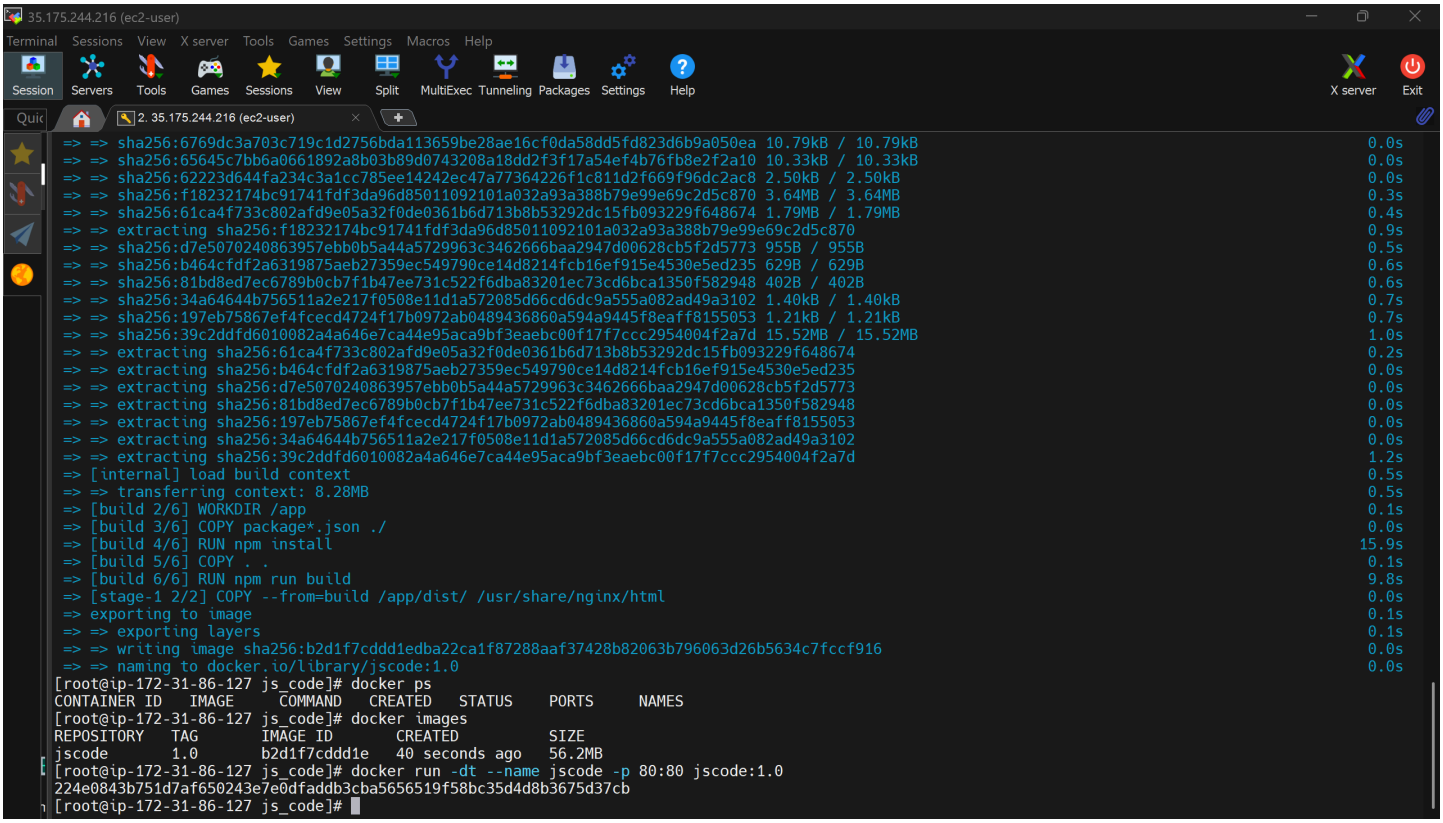
```
yum install -y git docker
systemctl start docker
```

2. Clone the **private repo** using your GitHub username and the **PAT key** as password.

3. Switch to the project directory and **build the Docker image**.

4. Check the image and run the container:

```
docker ps
docker run -dt --name jscode -p 80:80 jscode:1.0
```



> ✅ Our Docker image is just **56.2MB** as we used the Alpine image (lightweight). Here's an improved version of your steps (5–8) with emphasis added on **real-world relevance** and an **interview-style explanation** for the scenario you mentioned:

5. Go **inside the running container** and **manually update files** like `index.html` to simulate version changes (e.g., v1 → v2).

> ✅ This is useful in **emergency situations** where you need to quickly patch or debug an application without restarting the container or rebuilding the image.

```
[root@ip-172-31-86-127 js_code]# docker exec -it 224e0843b7 sh
/ # cd /usr/share/nginx/html
/usr/share/nginx/html # ls
50x.html    bundle.js   images      index.html
/usr/share/nginx/html # >index.html
/usr/share/nginx/html # vi index.html
```

6. To access the app using `<public-ip>:80`, make sure to **modify the EC2 security group** and **add port 80** to inbound rules.

7. Once your changes are tested inside the container, you can **create a new Docker image with all modifications**. This step is **very important** in real-world scenarios where you tweak live containers and want to preserve the state.

```
[root@ip-172-31-86-127 js_code]# docker commit 224e0843b751 jscode:2.0
sha256:afa8f59407427d1339a6140c73cdcdb4ca1153437ab007eb21b4e02980740cb5
[root@ip-172-31-86-127 js_code]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
jscode       2.0       afa8f5940742   5 seconds ago    56.2MB
jscode       1.0       b2d1f7cddd1e   13 minutes ago   56.2MB
[root@ip-172-31-86-127 js_code]# docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS          PORT
224e0843b751   jscode:1.0   "/docker-entrypoint.…"   11 minutes ago   Up 11 minutes   0.0.
```

📌 **Interview Scenario:**

Q: I did some modifications in a running container. Can I create a Docker image with those changes? If yes, how? ✅ **Yes, it's possible.** Here's how you do it:

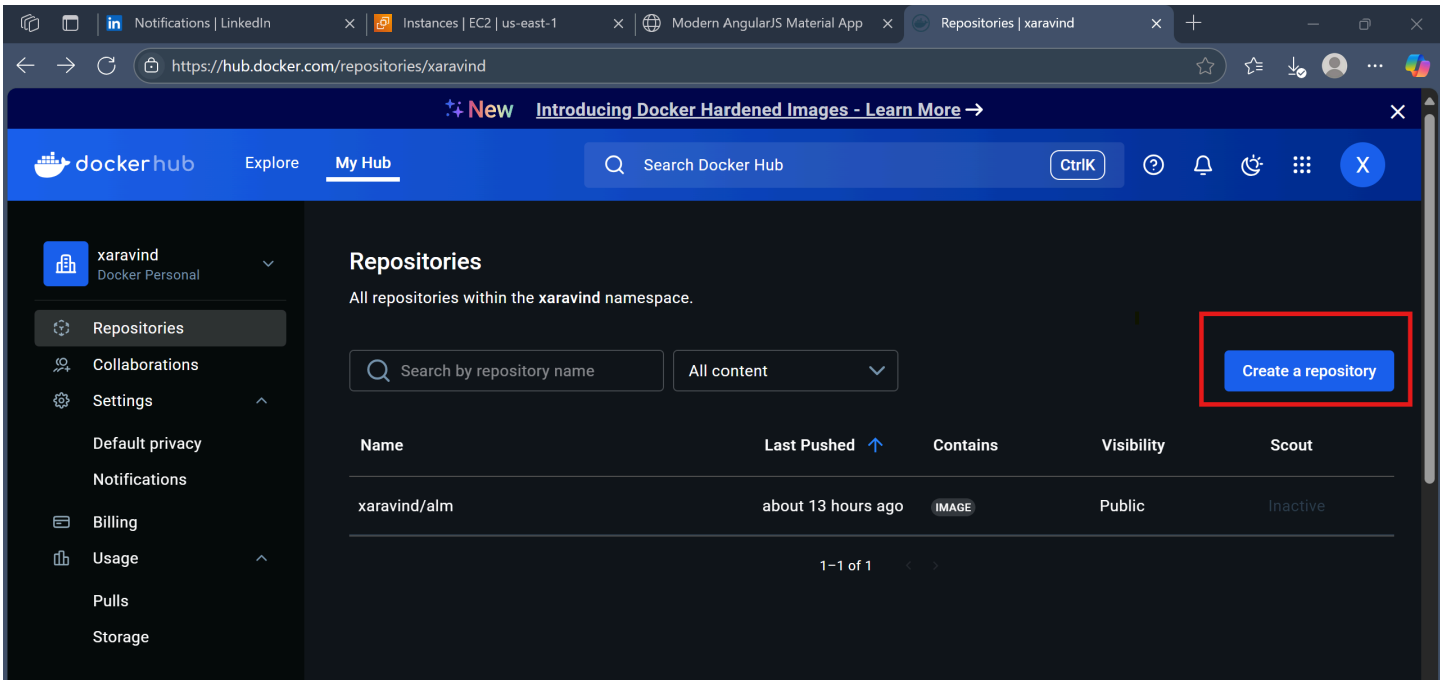
```
docker commit <container_id> <new_image_name>:<tag>
```

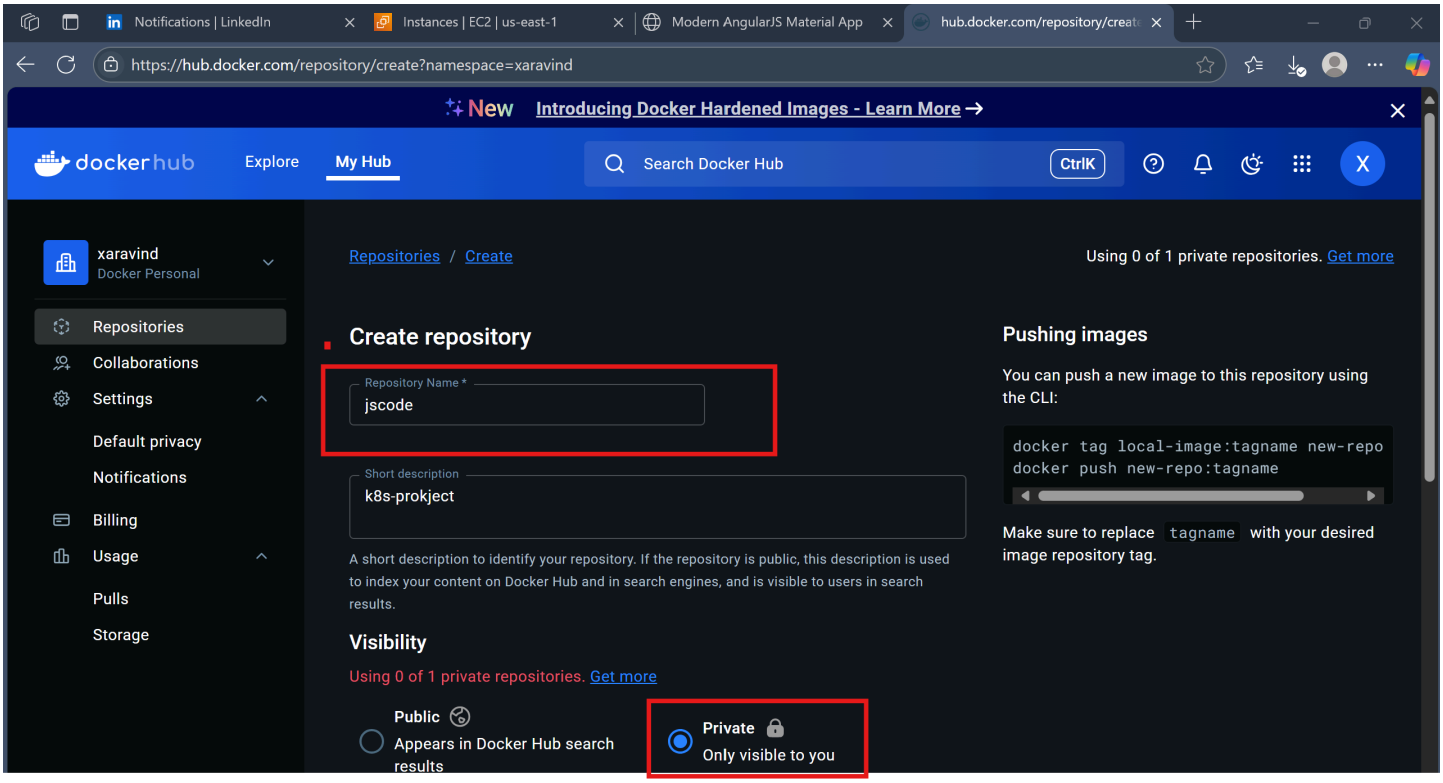
Example:

```
docker commit jscode xaravind/jscode:2.0
```

This creates a new image from the **current state** of the running container, including all manual edits or installed packages.

8. Now login to **Docker Hub**, and create a **private repository** to store your updated Docker images.





9. Tag images for pushing:

```
docker tag jscode:1.0 xaravind/jscode:v1
docker tag jscode:2.0 xaravind/jscode:v2
```

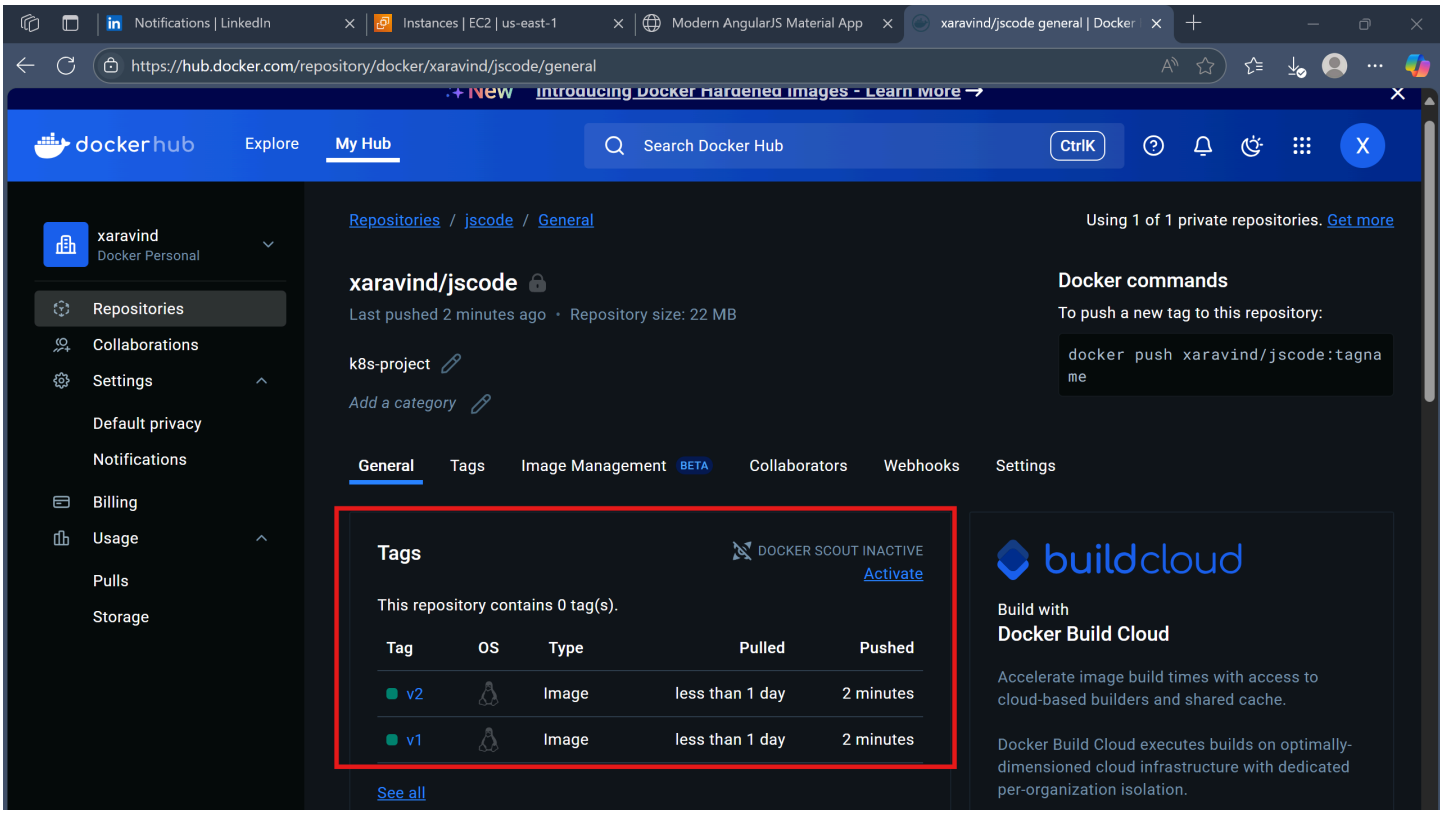
10. Login to Docker using your credentials. This will create `/root/.docker/config.json` which stores credentials. We will use this file to pull images in Kubernetes.

```
[root@ip-172-31-86-127 js_code]# docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If yo
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope

Username: xaravind
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

11. Push the tagged images to Docker Hub:

```
docker push xaravind/jscode:v1
docker push xaravind/jscode:v2
```

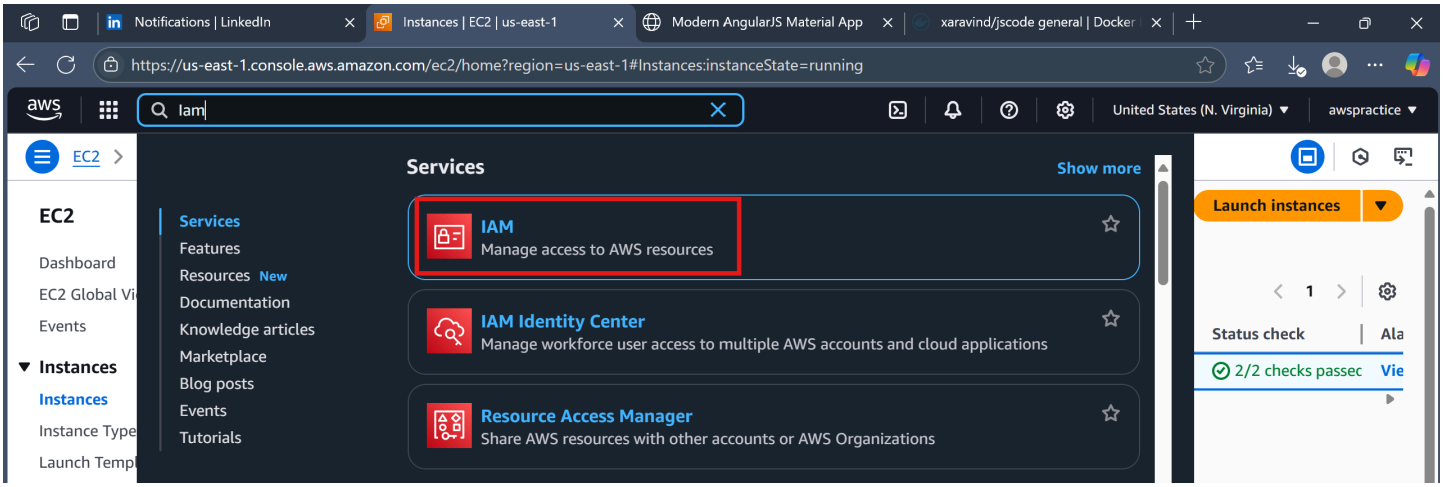
12. Open Docker Hub in browser and **verify images are uploaded**.



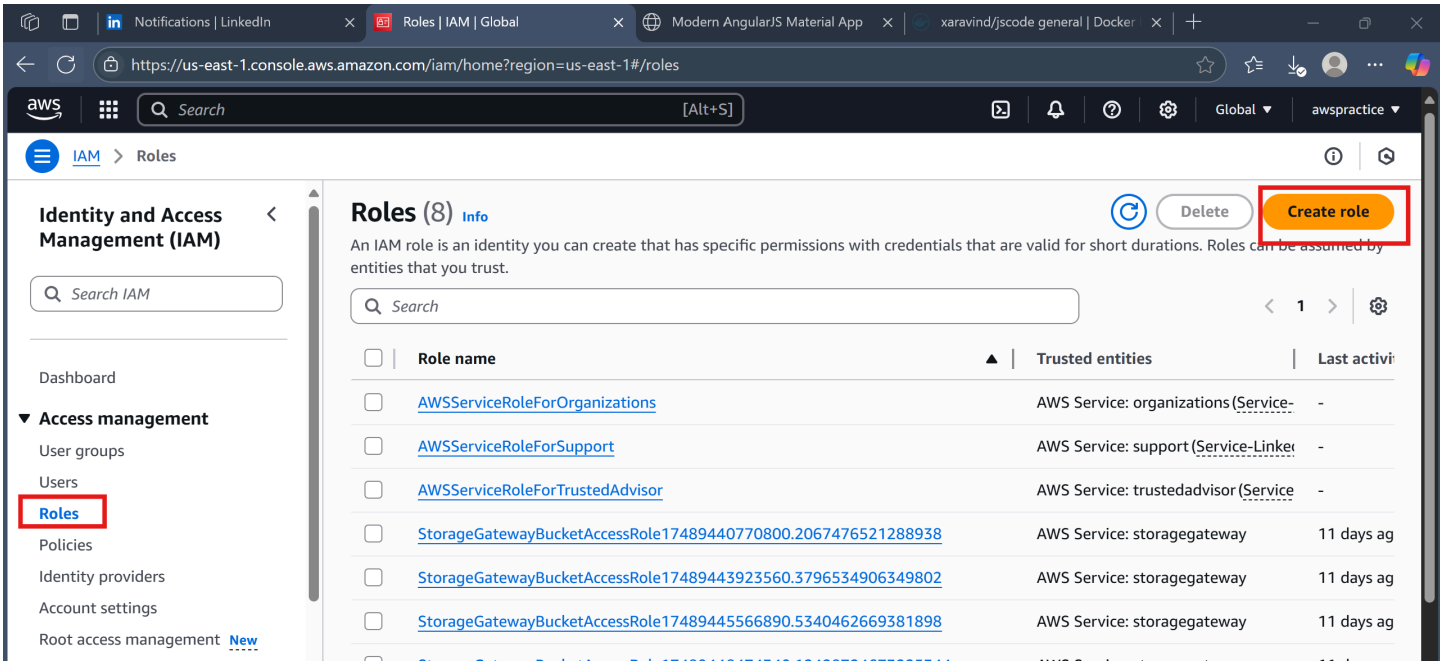
> ✅ In this stage, we installed packages, cloned the repo, built Docker images, ran and modified containers, and pushed final images to Docker Hub.

✅ Fourth Stage: EKS Setup with IAM Role

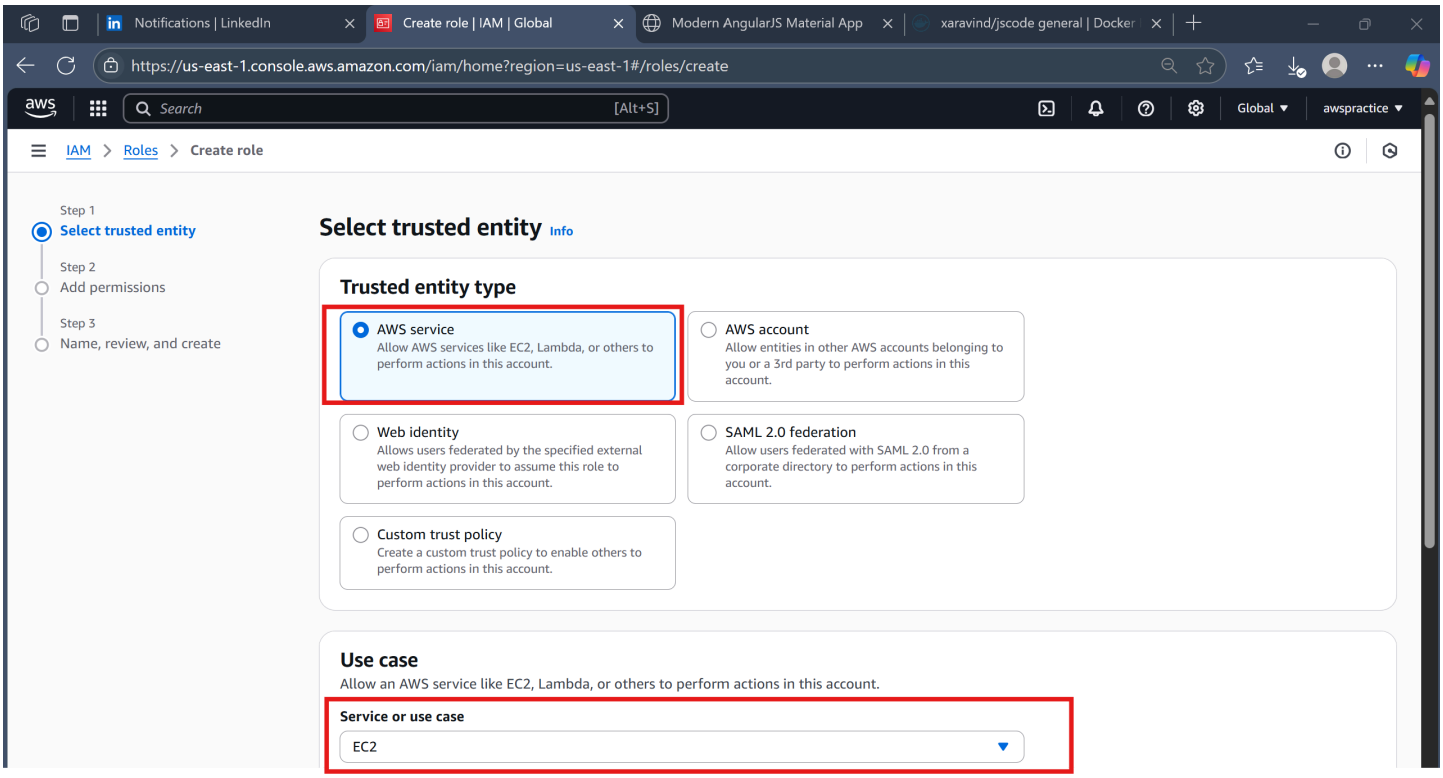
1. Go to AWS Console → Search for **IAM**.



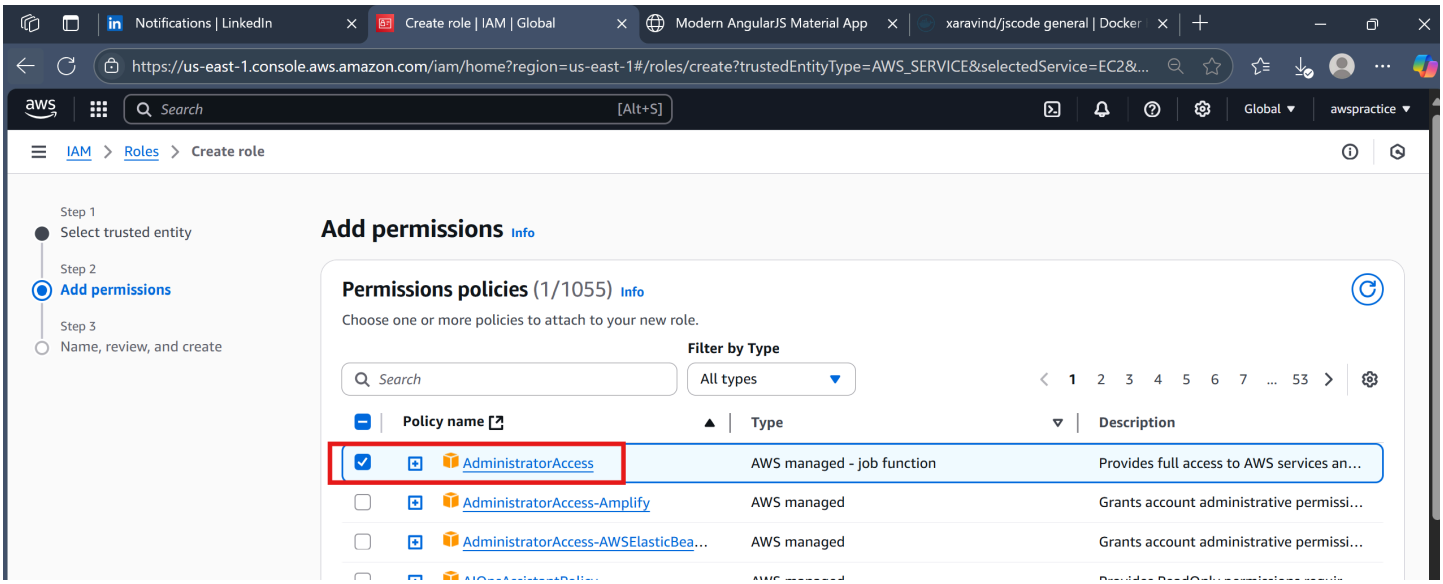
2. Click **Roles** → **Create role**.



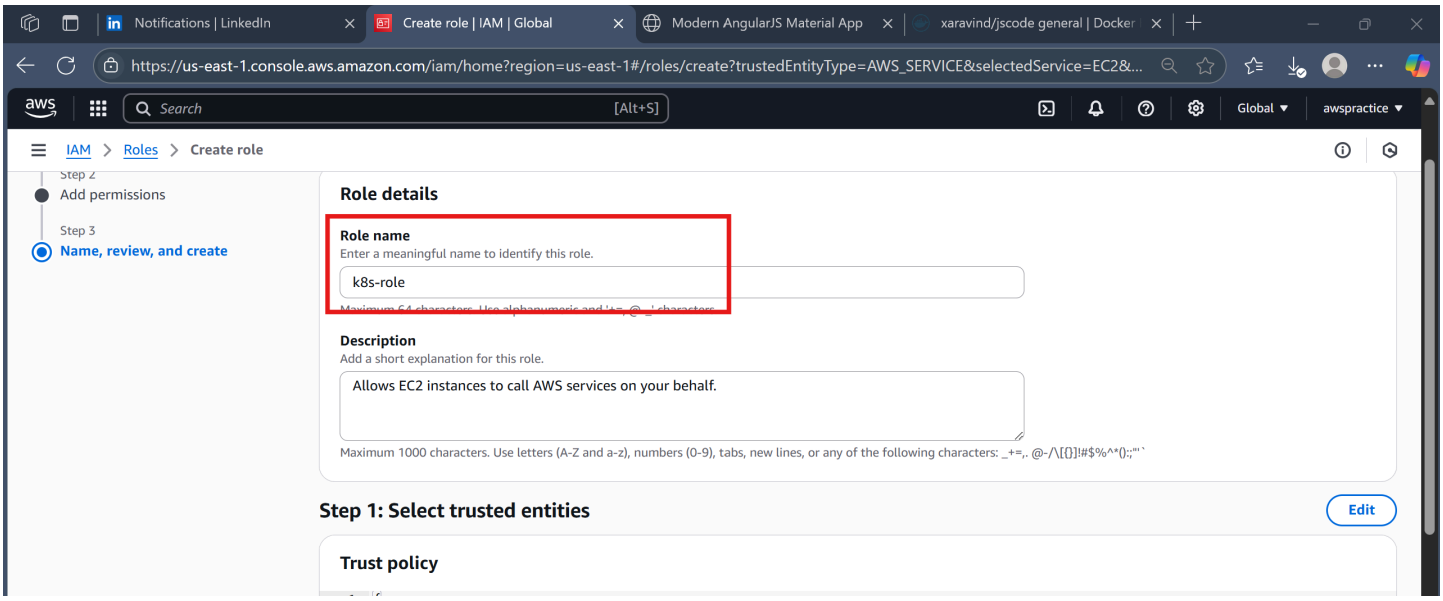
3. Select **Trusted entity type: AWS Service** and use case as **EC2**, then click **Next**.



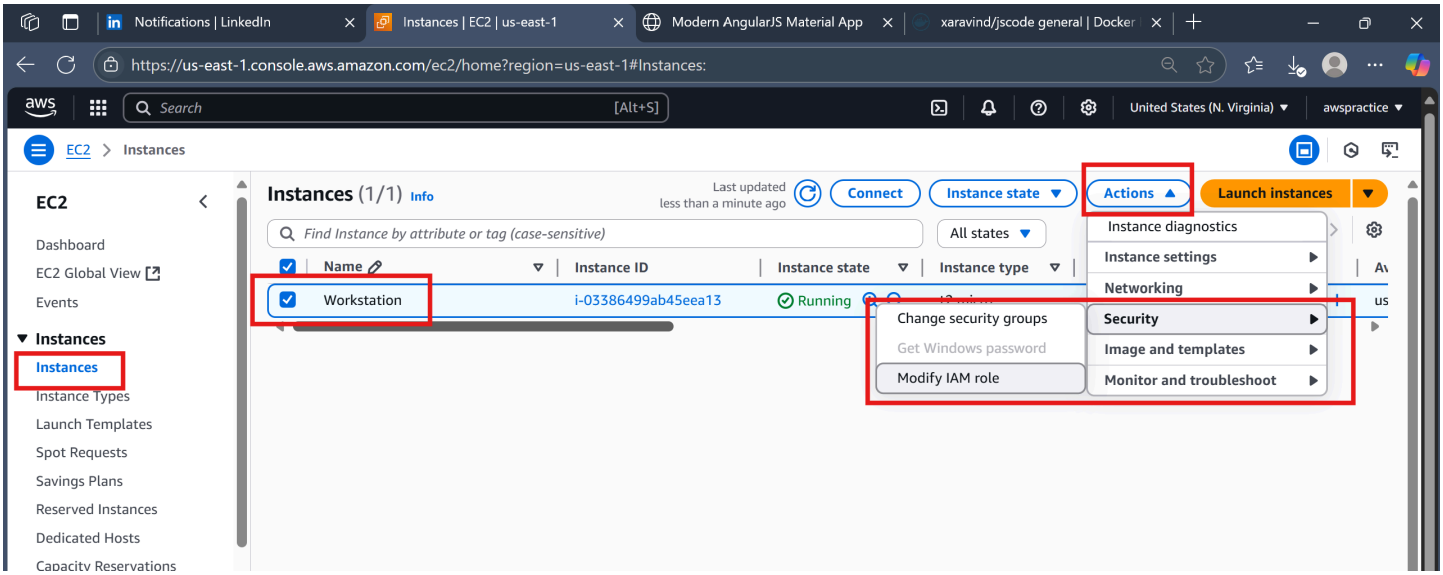
4. Choose **AdministratorAccess** policy (has full EC2 + EKS permissions), then click **Next**.



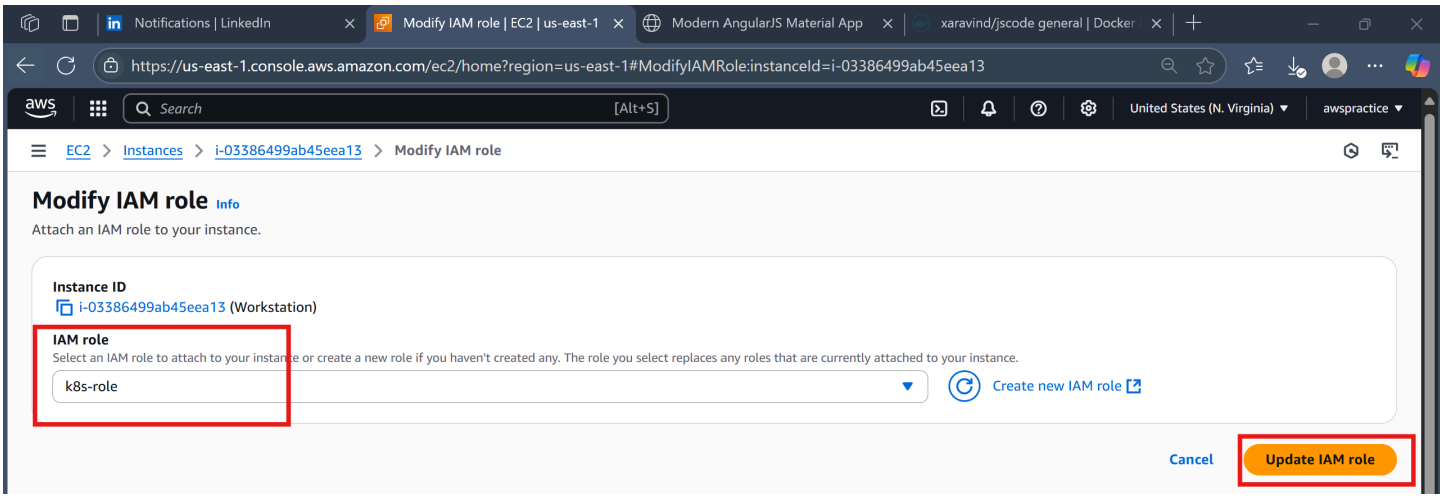
5. Name the role and click **Create Role**.



6. Go to **EC2 Instances**, select your instance → **Actions** → **Security** → **Modify IAM role**.



7. Attach the newly created **IAM role** to the instance.



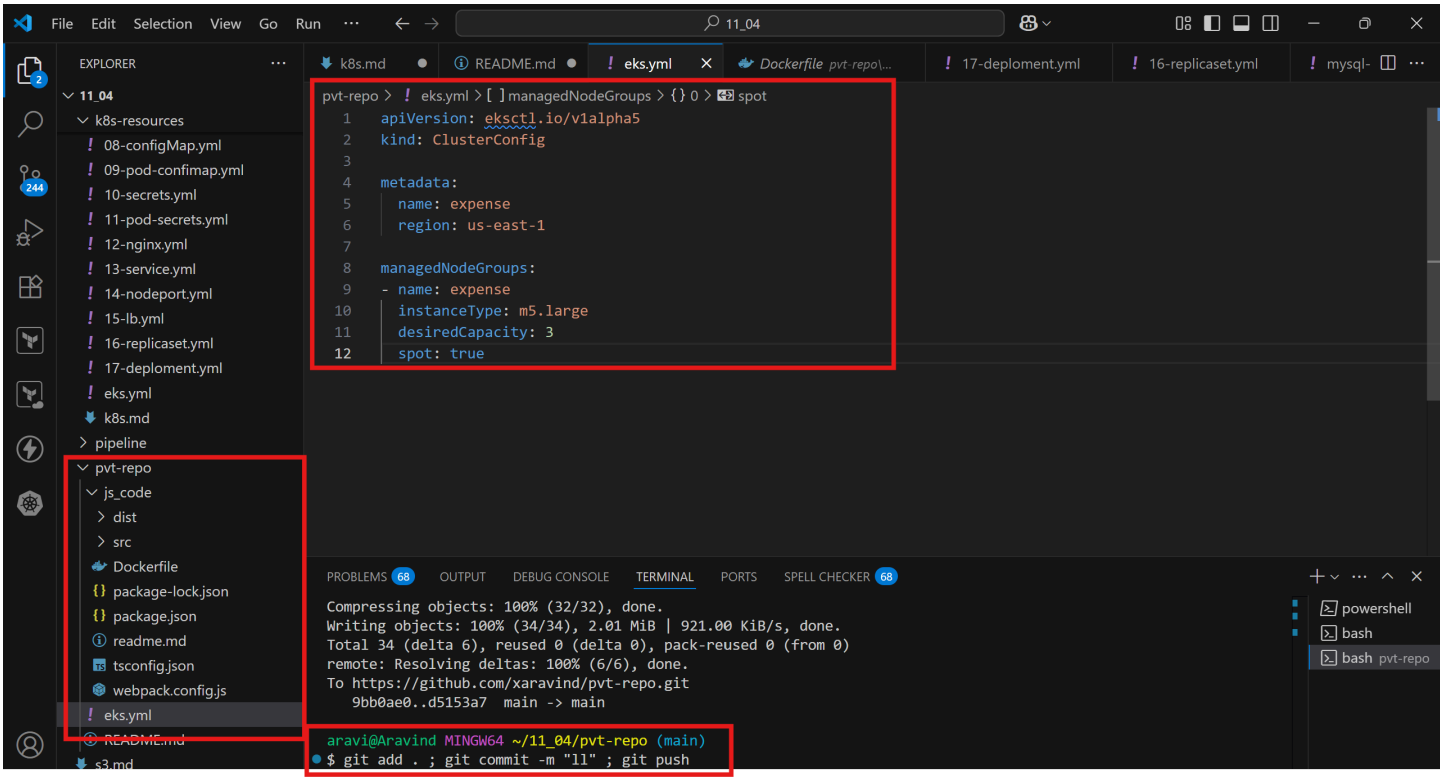
> ✅ Now your EC2 instance can control AWS services via AWS CLI.

8. AWS CLI comes pre-installed in Amazon Linux. For other OS, you'll need to install it manually.

9. Check AWS access:

```
aws s3 mb s3://test-k8s.js
aws s3 ls
```

10. Create an `eks.yml` file locally and push it to GitHub.



> We are using **spot instances** for cost-saving (\~90% discount), but they may be terminated

```bash
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
name: project
region: us-east-1

managedNodeGroups:
- name: project
instanceType: m5.large
desiredCapacity: 3
spot: true
```

> 🕐 This will provision VPCs, subnets, IAM roles, node groups — and may take several minutes.

11. Pull the updated code into the EC2 instance.

```
[root@ip-172-31-86-127 ~]# cd pvt-repo/
[root@ip-172-31-86-127 pvt-repo]# ls
README.md  js_code
[root@ip-172-31-86-127 pvt-repo]# git pull -a
[root@ip-172-31-86-127 pvt-repo]# ls
README.md  eks.yml  js_code
```

12. Install `eksctl` : Follow instructions from (https://eksctl.io/installation)

```
[root@ip-172-31-86-127 pvt-repo]# ls
README.md  eks.yml  js_code
[root@ip-172-31-86-127 pvt-repo]# # for ARM systems, set ARCH to: `arm64`, `armv6` or `arm
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH

curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.t

# (Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.tx

tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz

sudo mv /tmp/eksctl /usr/local/bin
eksctl_Linux_amd64.tar.gz: OK
rm: remove regular file 'eksctl_Linux_amd64.tar.gz'? y

[root@ip-172-31-86-127 pvt-repo]# eksctl version
0.210.0
```

13. Use the `eks.yml` file to **create an EKS cluster**.

> This takes time and will create required AWS services like nodes, VPCs, SGs, ASG, etc.

```
[root@ip-172-31-86-127 pvt-repo]# eksctl create cluster --config-file=eks.yml
2025-06-15 07:05:09 [ℹ]  eksctl version 0.210.0
2025-06-15 07:05:09 [ℹ]  using region us-east-1
2025-06-15 07:05:09 [ℹ]  setting availability zones to [us-east-1b us-east-1a]
2025-06-15 07:05:09 [ℹ]  subnets for us-east-1b - public:192.168.0.0/19 private:192.168.64
2025-06-15 07:05:09 [ℹ]  subnets for us-east-1a - public:192.168.32.0/19 private:192.168.9
2025-06-15 07:05:09 [ℹ]  nodegroup "project" will use "" [AmazonLinux2023/1.32]
2025-06-15 07:05:09 [ℹ]  using Kubernetes version 1.32
2025-06-15 07:18:00 [✓]  EKS cluster "project" in "us-east-1" region is ready
```

14. Install `kubectl` to interact with the cluster.

```
[root@ip-172-31-86-127 pvt-repo]# curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 57.3M  100 57.3M    0     0  21.2M      0  0:00:02  0:00:02 --:--:-- 21.2M
[root@ip-172-31-86-127 pvt-repo]# chmod +x ./kubectl
[root@ip-172-31-86-127 pvt-repo]# mv ./kubectl /usr/local/bin
[root@ip-172-31-86-127 pvt-repo]# kubectl version --client
Client Version: v1.33.0-eks-802817d
Kustomize Version: v5.6.0
```

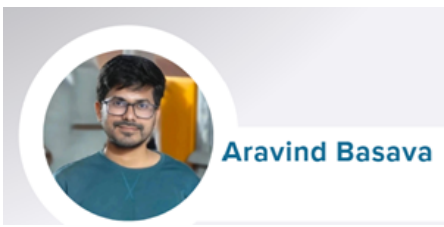
15. Test kubectl:

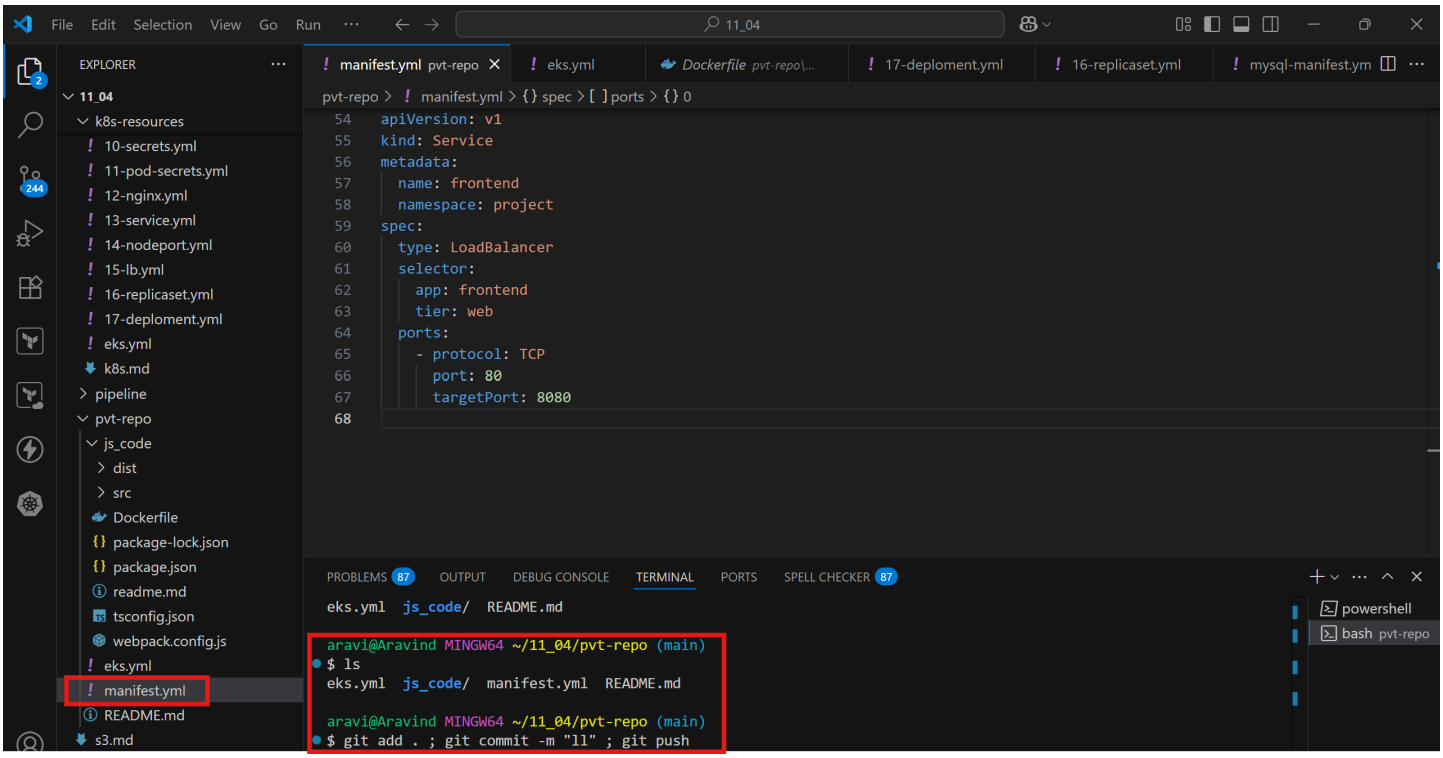
```
[root@ip-172-31-86-127 pvt-repo]#  kubectl api-resources | wc -l
66
[root@ip-172-31-86-127 pvt-repo]# kubectl get nodes
NAME                              STATUS   ROLES    AGE    VERSION
ip-192-168-27-169.ec2.internal   Ready    <none>   9m7s   v1.32.3-eks-473151a
ip-192-168-55-115.ec2.internal   Ready    <none>   9m8s   v1.32.3-eks-473151a
ip-192-168-63-135.ec2.internal   Ready    <none>   9m7s   v1.32.3-eks-473151a
```

16. Create a base64 secret from Docker credentials to use in Kubernetes secrets:

```
[root@ip-172-31-86-127 pvt-repo]# cat ~/.docker/config.json | base64 -w 0
ewoJImF1dGhzIjxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxCn0=
```

17. Create a manifest file locally, push it to GitHub, and pull it on the EC2 instance.






```bash
[root@ip-172-31-86-127 pvt-repo]# cat manifest.yml
apiVersion: v1
kind: Secret
metadata:
name: regcred
namespace: project
type: kubernetes.io/dockerconfigjson
data:
.dockerconfigjson: ewoJImF1dGhzIjogewoJCSJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOiB7CgkJCSJhdXR
---
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
name: frontend
namespace: project
labels:
    app: frontend
    environment: project
spec:
replicas: 3
selector:
    matchLabels:
    app: frontend
    tier: web
template:
    metadata:
    labels:
        app: frontend
        tier: web
    spec:
```

```
    imagePullSecrets:
        - name: regcred
    containers:
        - name: frontend
        image: xaravind/jscode:v2
        ports:
            - containerPort: 80
        resources:
            requests:
            memory: "64Mi"
            cpu: "250m"
            limits:
            memory: "256Mi"
            cpu: "500m"
        readinessProbe:
            httpGet:
            path: /
            port: 80
            initialDelaySeconds: 5
            periodSeconds: 10
        livenessProbe:
            httpGet:
            path: /
            port: 80
            initialDelaySeconds: 15
            periodSeconds: 20
---
# Service
apiVersion: v1
kind: Service
metadata:
name: frontend
namespace: project
spec:
type: LoadBalancer
selector:
    app: frontend
    tier: web
ports:
    - protocol: TCP
    port: 80
    targetPort: 80
---
```

Above manifest file defines everything required to deploy a containerized application securely and publicly in a Kubernetes cluster.

1. Docker Registry Secret

- **Purpose**: Allows Kubernetes to pull an image from a **private Docker Hub repository** using a base64-encoded Docker config (`~/.docker/config.json`).

- **Key:** `regcred` (referenced in the deployment under `imagePullSecrets`).

2. Deployment

- **Replicas:** `3` – Ensures high availability by running 3 pods.
- **Image:** `xaravind/jscode:v1` –
- **Probes:**
  - **Readiness Probe** – Checks if the container is ready to serve traffic.
  - **Liveness Probe** – Monitors container health and restarts if unhealthy.
- **Resources:**
  - **Requests:** Memory: `64Mi`, CPU: `250m`
  - **Limits:** Memory: `256Mi`, CPU: `500m`
- **Image Pull Secret:** Uses the previously created `regcred` secret to authenticate with Docker Hub.

3. LoadBalancer Service

- **Type:** `LoadBalancer` – Automatically provisions an **external load balancer** (like AWS ELB when running on EKS).
- **Port Mapping:**
  - **port:** `80` (exposed publicly)
  - **targetPort:** `80` (mapped to the container port)

✅ Summary

✔ **Pulls a private image** securely using a Docker Hub token

✔ **Deploys the frontend application** as a replicated, monitored set of pods

✔ **Uses probes** to maintain app health and availability

✔ **Exposes the app** to the public internet via a load balancer

18. Create a namespace and apply the manifest file:

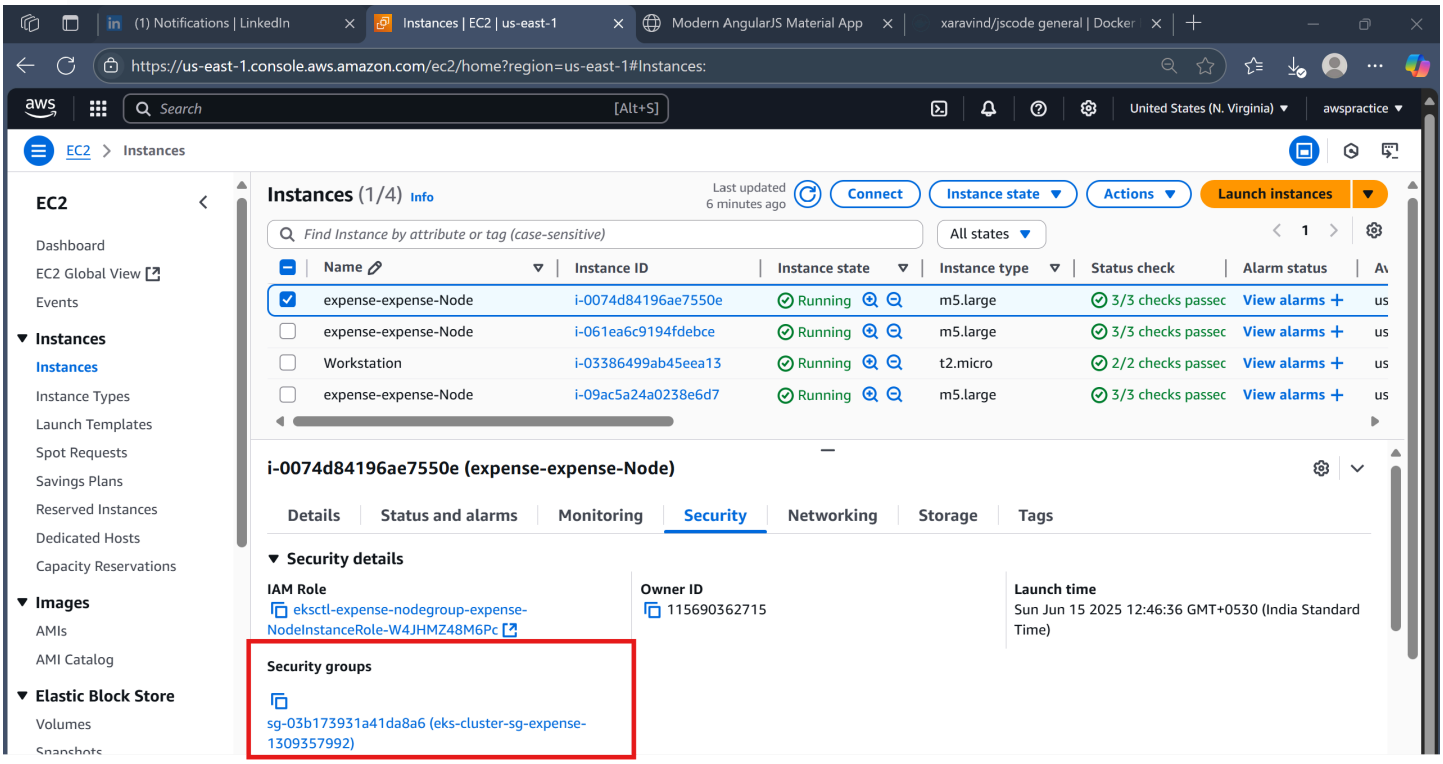
```
kubectl create namespace frontend
kubectl apply -f manifest.yaml -n frontend
```

19. Check pods and services. Kubernetes will create a **LoadBalancer service** with a random **NodePort (30000–32767)**.

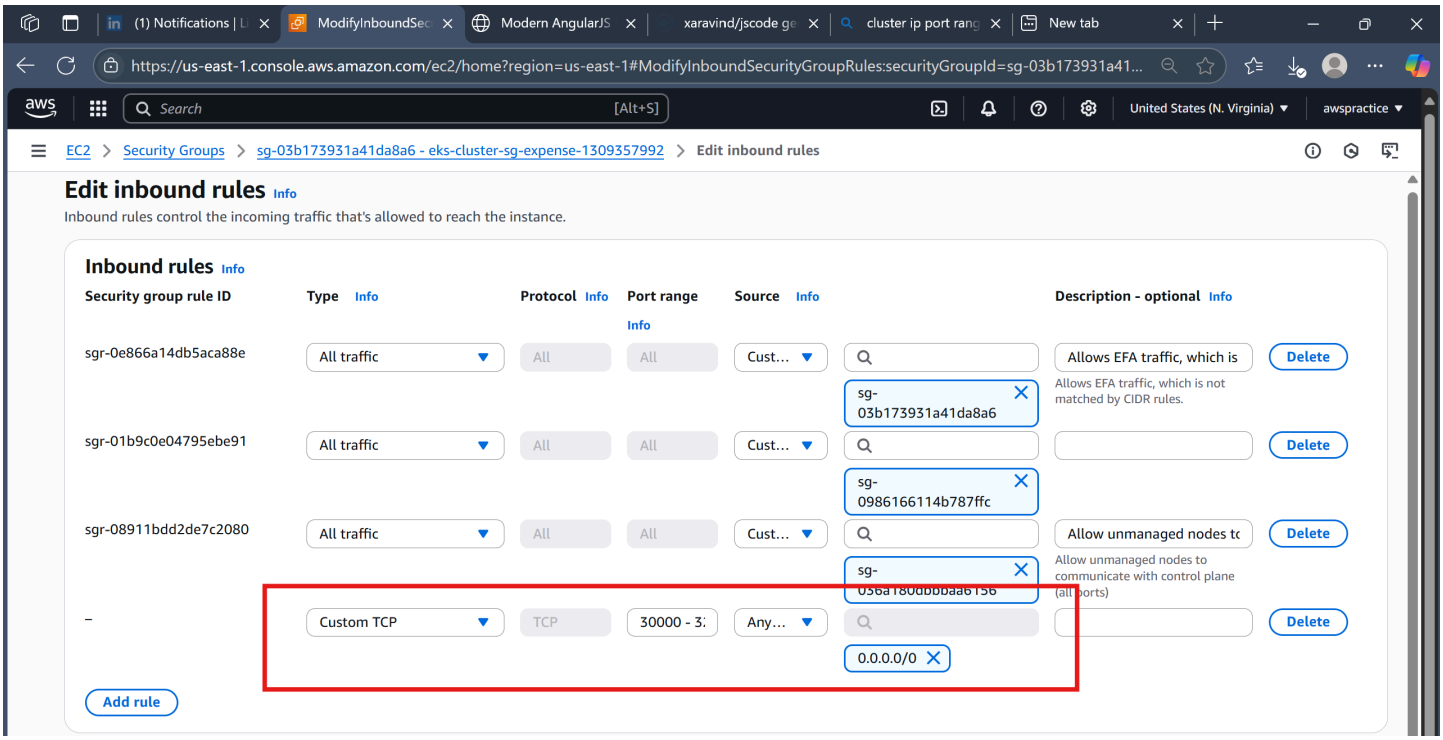
```
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -n project
NAME                        READY   STATUS             RESTARTS      AGE
frontend-644df5c8b4-97gbj   0/1     Running            1 (32s ago)   92s
```

```
frontend-6467b94fb8-hghzx   0/1     Running            0              11s
frontend-6467b94fb8-t95fs   1/1     Running            0              25s
frontend-6d78b8b5b6-xw8hl   0/1     CrashLoopBackOff   5 (21s ago)    6m22s
[root@ip-172-31-86-127 pvt-repo]# kubectl describe pod frontend-6467b94fb8-hghzx -n projec
Name:             frontend-6467b94fb8-hghzx

Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:
Type     Reason     Age   From               Message
----     ------     ----  ----               -------
Normal   Scheduled  28s   default-scheduler  Successfully assigned project/frontend-6467b94
Normal   Pulled     27s   kubelet            Container image "xaravind/jscode:v1" already p
Normal   Created    27s   kubelet            Created container: frontend
Normal   Started    27s   kubelet            Started container frontend
[root@ip-172-31-86-127 pvt-repo]# kubectl get svc frontend -n project
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP
```

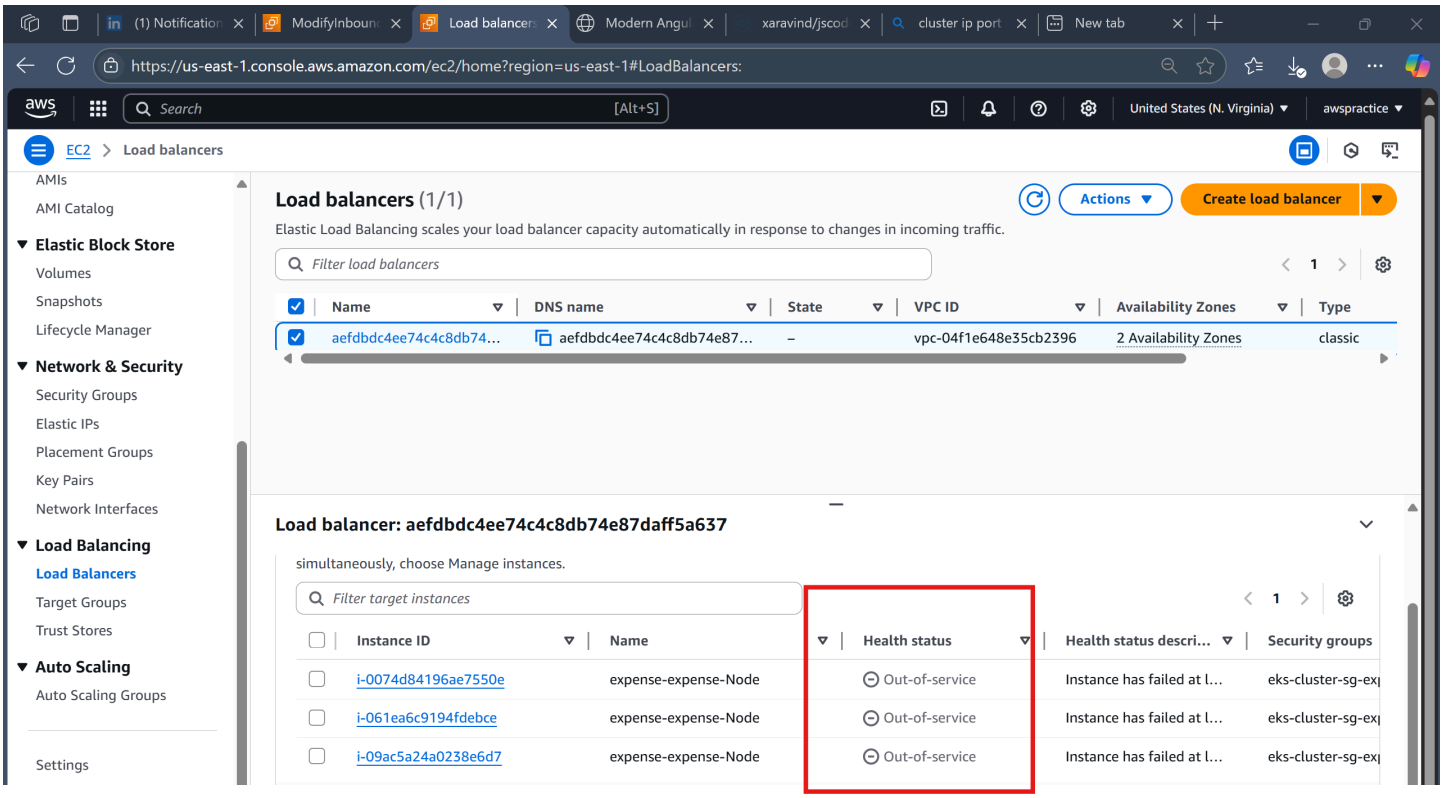
20. To access the app, **add inbound rule for NodePort range (30000–32767)** in the security group.

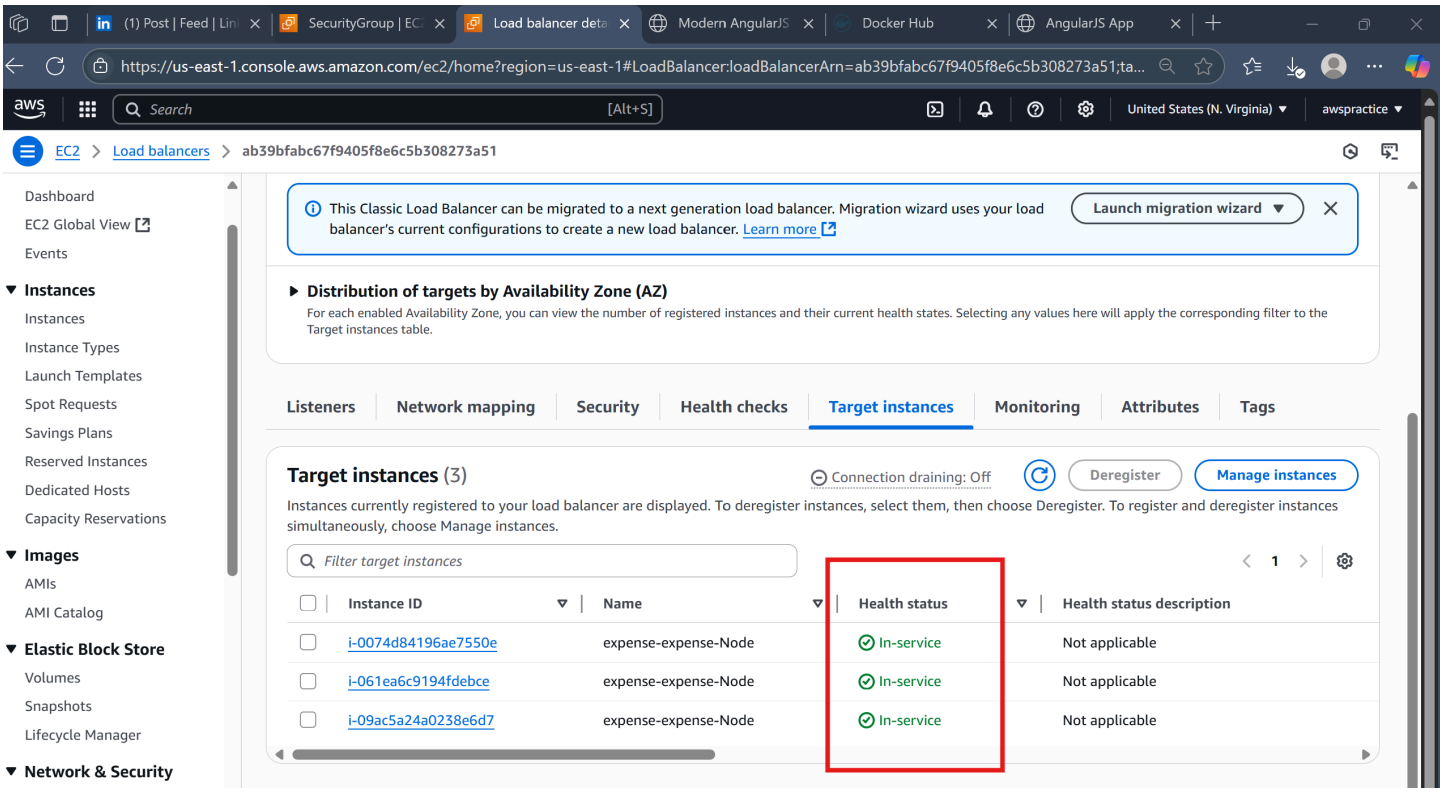


21. Go to EC2 → Nodes → Security Group → Edit Inbound Rules → Add port range.

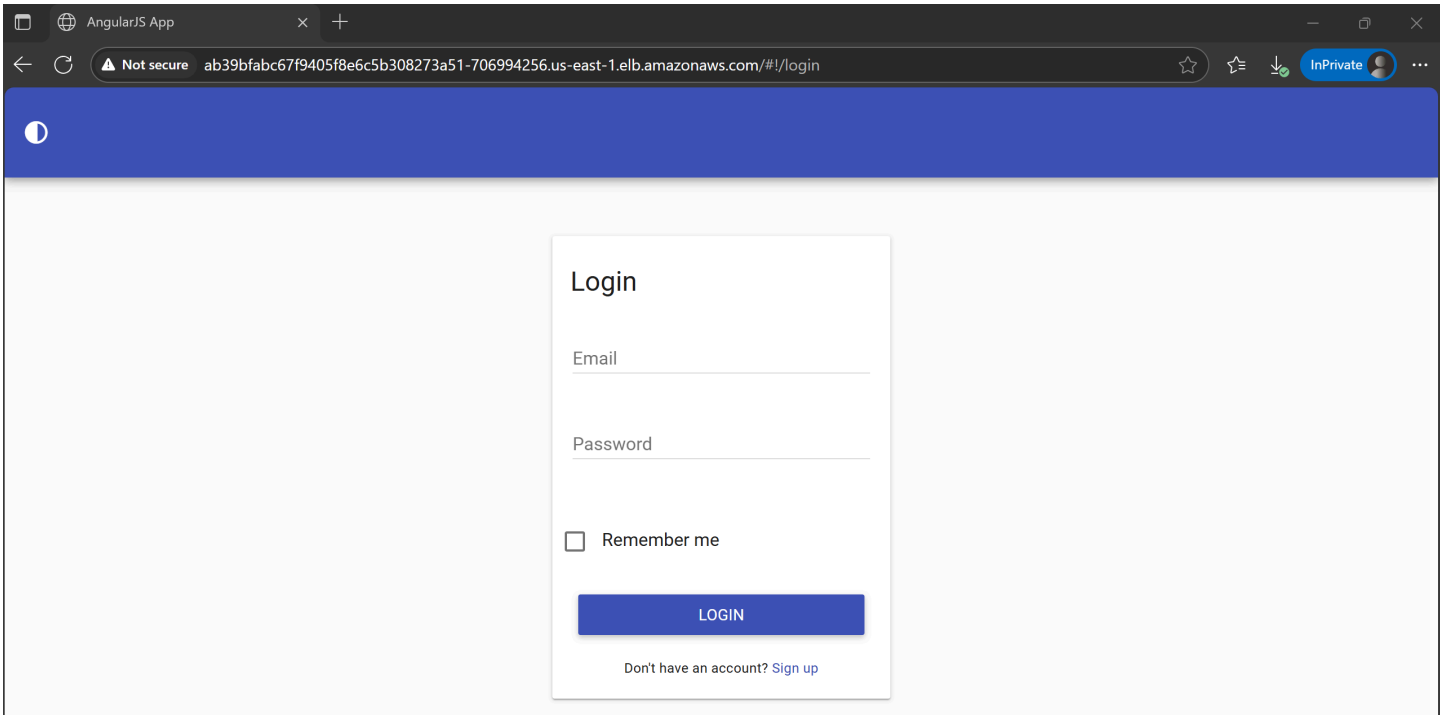


22. Access the application via LoadBalancer or NodePort.






```bash
curl http://<EXTERNAL-IP>/
curl http://ab39bfabc67f9405f8e6c5b308273a51-706994256.us-east-1.elb.amazonaws.com/
```



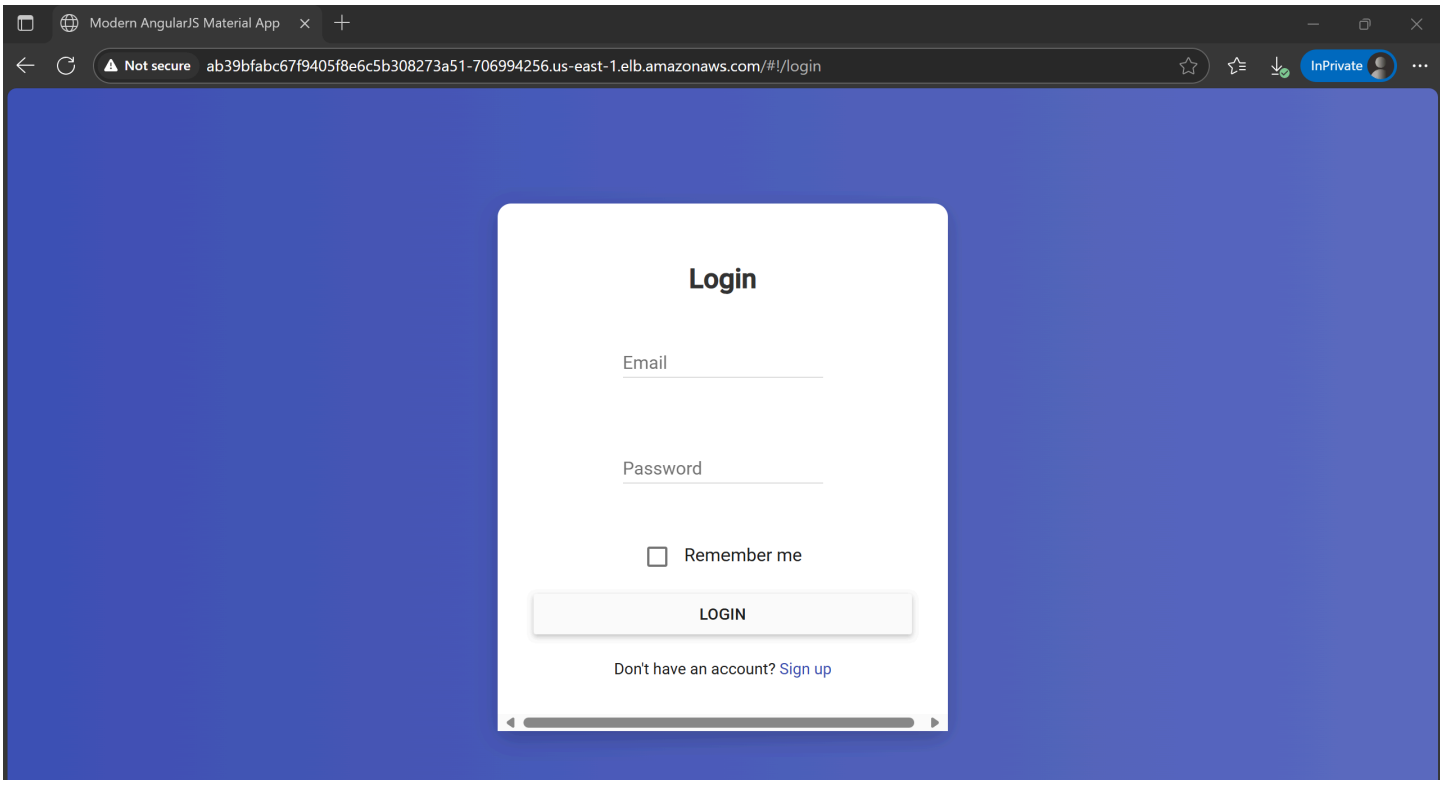
23. Update mainfest file with v2 image - xaravind/jscode:v2

24. repply the changes

```
[root@ip-172-31-86-127 pvt-repo]# vi manifest.yml
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -o wide -n project
NAME                        READY   STATUS    RESTARTS   AGE     IP               NODE
frontend-6467b94fb8-hghzx   1/1     Running   0          7m12s   192.168.62.210   ip-192-1
frontend-6467b94fb8-hs94j   1/1     Running   0          6m59s   192.168.43.93    ip-192-1
frontend-6467b94fb8-t95fs   1/1     Running   0          7m26s   192.168.29.173   ip-192-1
[root@ip-172-31-86-127 pvt-repo]# kubectl apply -f manifest.yml
secret/regcred unchanged
deployment.apps/frontend configured
service/frontend unchanged
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -o wide -n project
NAME                        READY   STATUS    RESTARTS   AGE     IP               NODE
frontend-6467b94fb8-hghzx   1/1     Running   0          7m55s   192.168.62.210   ip-192-1
frontend-6467b94fb8-hs94j   1/1     Running   0          7m42s   192.168.43.93    ip-192-1
frontend-6467b94fb8-t95fs   1/1     Running   0          8m9s    192.168.29.173   ip-192-1
frontend-f8f8b89df-zn2gn    0/1     Running   0          4s      192.168.2.38     ip-192-1
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -o wide -n project
NAME                        READY   STATUS    RESTARTS   AGE     IP               NODE
frontend-6467b94fb8-hghzx   1/1     Running   0          8m3s    192.168.62.210   ip-192-1
frontend-6467b94fb8-hs94j   1/1     Running   0          7m50s   192.168.43.93    ip-192-1
frontend-6467b94fb8-t95fs   1/1     Running   0          8m17s   192.168.29.173   ip-192-1
frontend-f8f8b89df-zn2gn    0/1     Running   0          12s     192.168.2.38     ip-192-1
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -o wide -n project
NAME                        READY   STATUS    RESTARTS   AGE     IP               NODE
frontend-6467b94fb8-hghzx   1/1     Running   0          8m10s   192.168.62.210   ip-192-1
frontend-6467b94fb8-hs94j   1/1     Running   0          7m57s   192.168.43.93    ip-192-1
frontend-f8f8b89df-ggncq    0/1     Running   0          5s      192.168.47.194   ip-192-1
frontend-f8f8b89df-zn2gn    1/1     Running   0          19s     192.168.2.38     ip-192-1
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -o wide -n project
NAME                        READY   STATUS    RESTARTS   AGE   IP               NODE
frontend-f8f8b89df-ggncq    1/1     Running   0          61s   192.168.47.194   ip-192-168-
frontend-f8f8b89df-xmjqn    1/1     Running   0          49s   192.168.39.242   ip-192-168-
frontend-f8f8b89df-zn2gn    1/1     Running   0          75s   192.168.2.38     ip-192-168-
```

> ✅ Observe how Kubernetes rolls out the new version of your app without downtime – this is called a **rolling update**.

25. Access the application, we easily updated the code with zeo downtime. and with single command.



✅ Final Cleanup

- Once done, **clean up your project** to avoid charges:
  - Delete EKS cluster
  - Remove EC2 instance
  - Delete S3 buckets and Docker images if not needed

```
eksctl delete cluster --config-file=eks.yml
[root@ip-172-31-86-127 pvt-repo]# eksctl delete cluster --config-file=eks.yml
2025-06-15 08:25:07 [ℹ]  deleting EKS cluster "project"
2025-06-15 08:25:07 [ℹ]  will drain 0 unmanaged nodegroup(s) in cluster "project"
2025-06-15 08:25:07 [ℹ]  starting parallel draining, max in-flight of 1
2025-06-15 08:25:07 [ℹ]  deleted 0 Fargate profile(s)
2025-06-15 08:25:08 [✓]  kubeconfig has been updated
2025-06-15 08:25:08 [ℹ]  cleaning up AWS load balancers created by Kubernetes objects
2025-06-15 08:25:33 [ℹ]
2 sequential tasks: { delete nodegroup "project", delete cluster control plane "projec
}
2025-06-15 08:25:33 [ℹ]  will delete stack "eksctl-project-nodegroup-project"
2025-06-15 08:25:33 [ℹ]  waiting for stack "eksctl-project-nodegroup-project" to get d
2025-06-15 08:25:33 [ℹ]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:26:03 [ℹ]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:26:48 [ℹ]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:27:40 [ℹ]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:28:30 [ℹ]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
```

```
2025-06-15 08:29:31 [i]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:30:55 [i]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:32:25 [i]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:34:07 [i]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:35:46 [i]  waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:35:46 [i]  will delete stack "eksctl-project-cluster"
```

> ✅ In this stage, we **created an IAM role** with Administrator access, **attached the role** to the EC2 instance, **installed `eksctl` and `kubectl` packages**, **created an EKS cluster**, **set up a namespace**, **applied the manifest file**, **accessed the application**, **updated the deployment with image v2**, and finally **performed cleanup** of all resources.