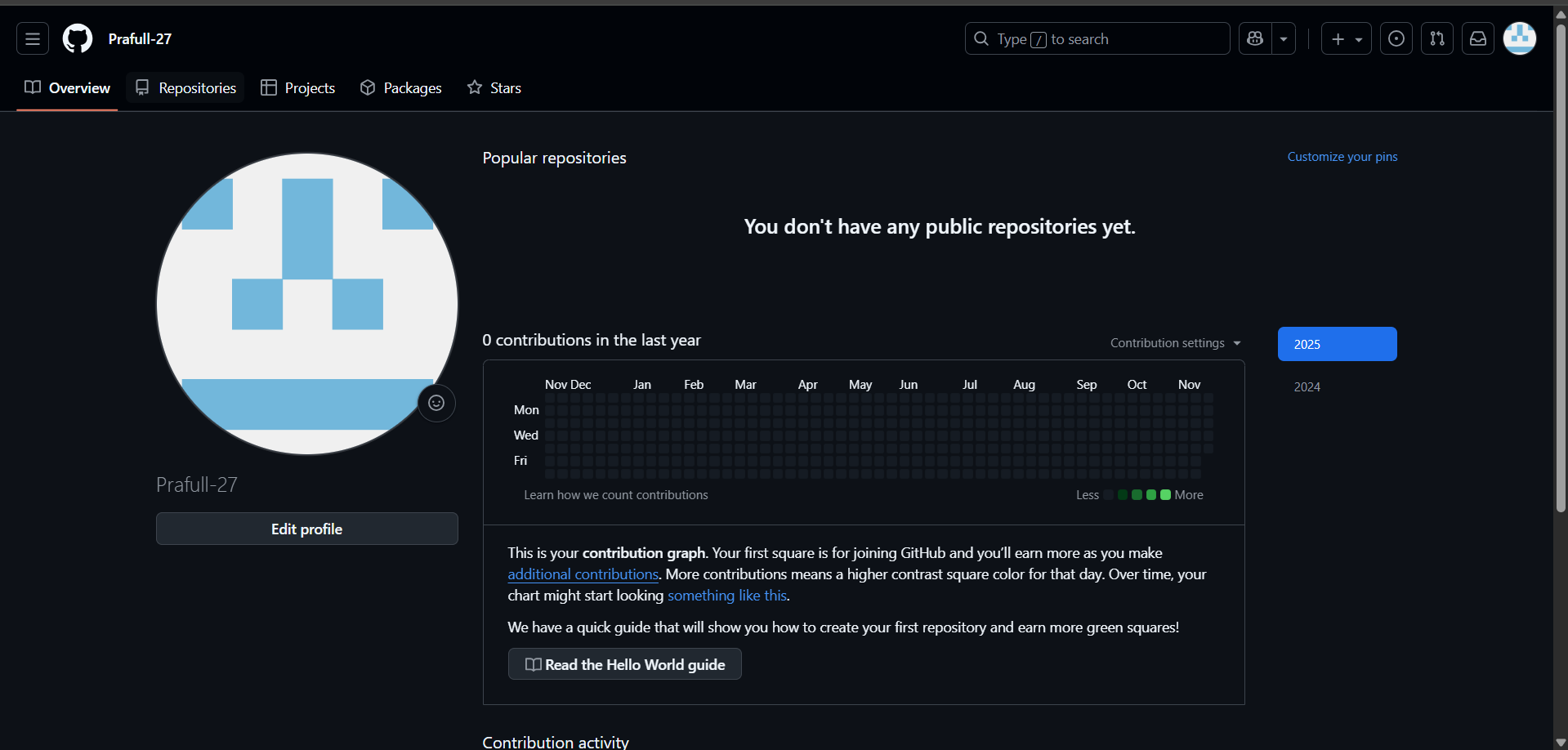
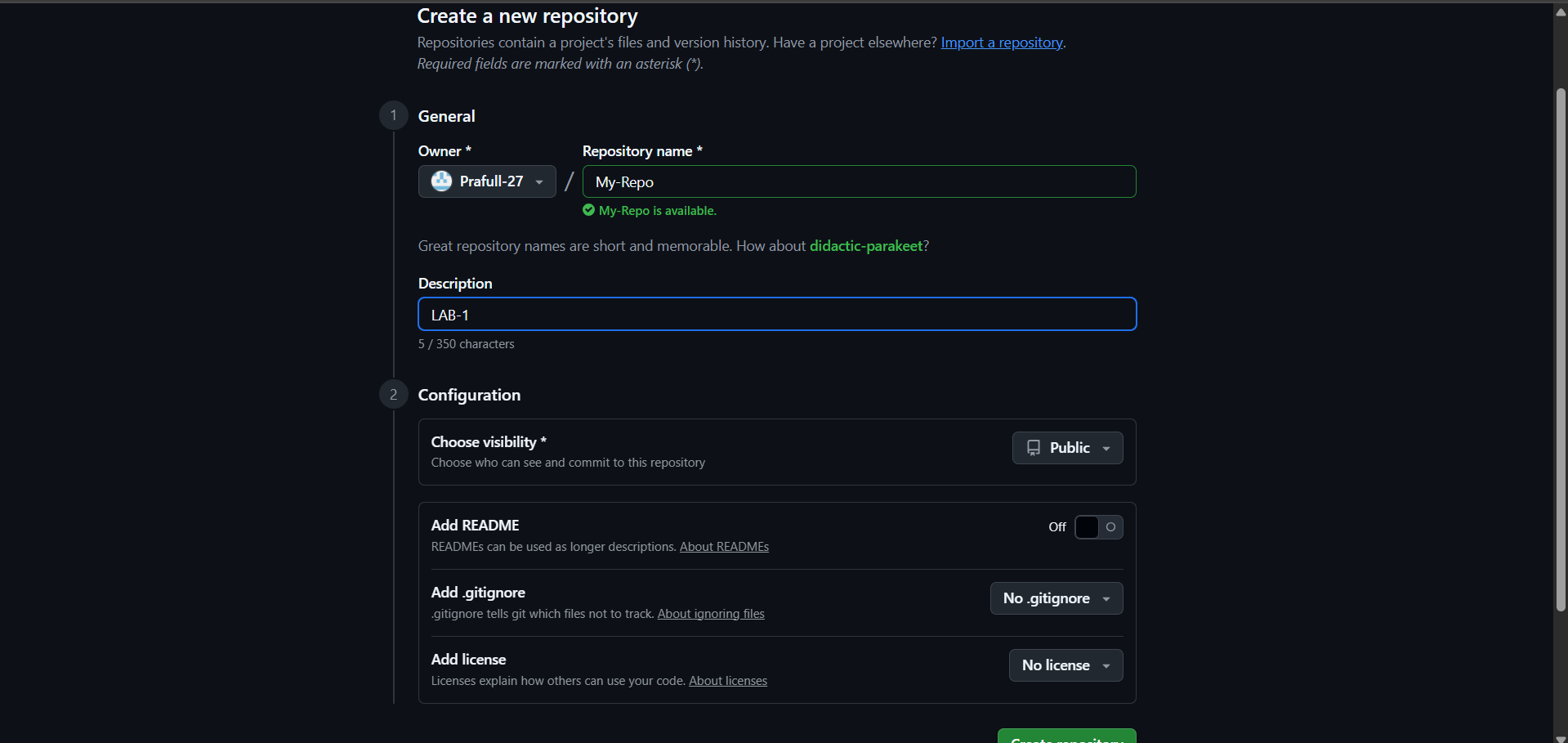
1. Create account on GitHub

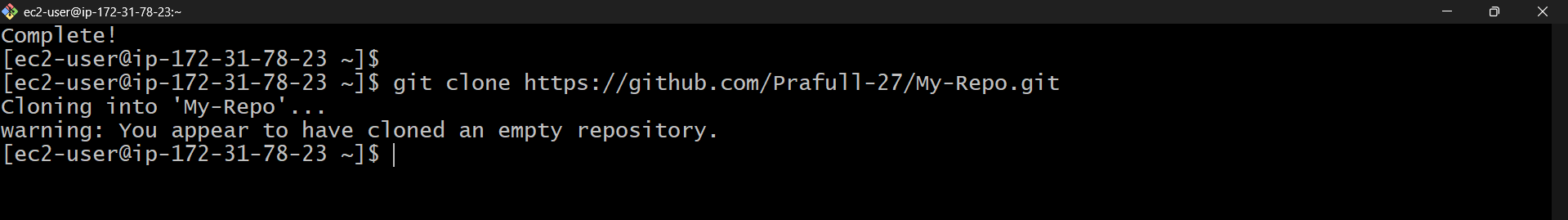


1. Then Go to Repositories and Create New Repository

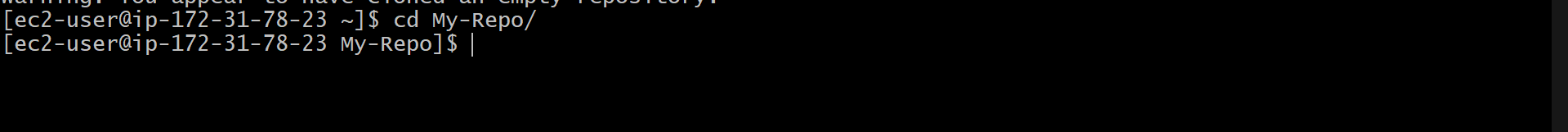


1. Now Clone This repository with Your system (ec2-instance).

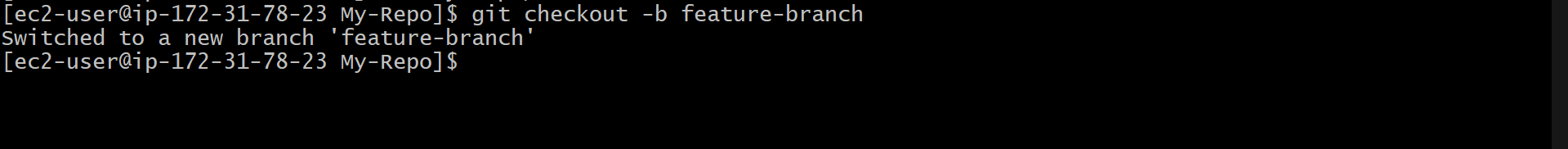
$ git clone <https://github.com/Prafull-27/My-Repo.git>



1. Now our pwd to repository .



1. Now Create New Branch.

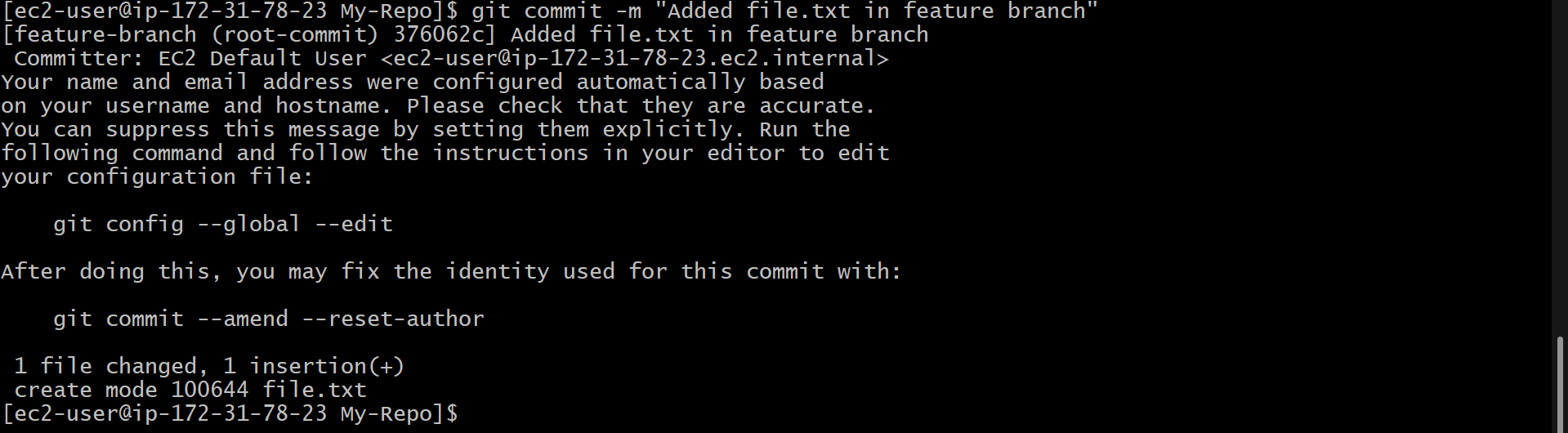


1. Make changes in your branch:

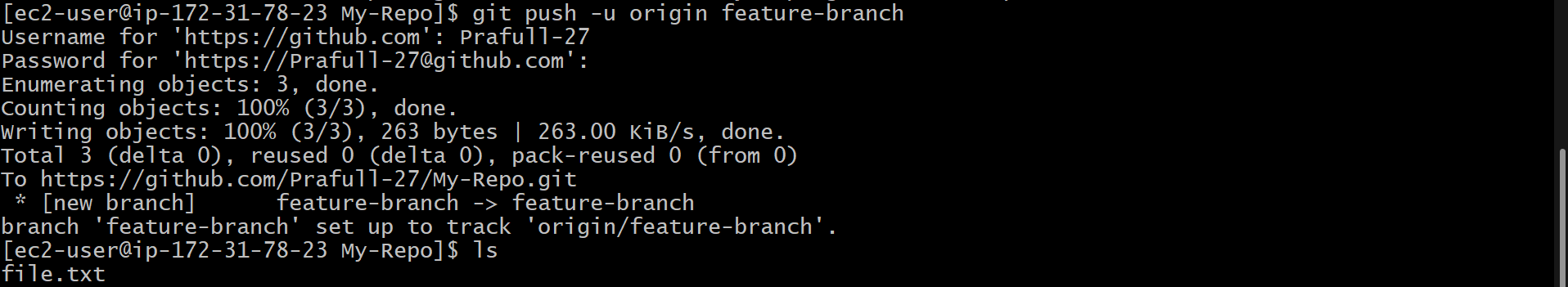
Edit or create any file.



1. Add and commit your changes

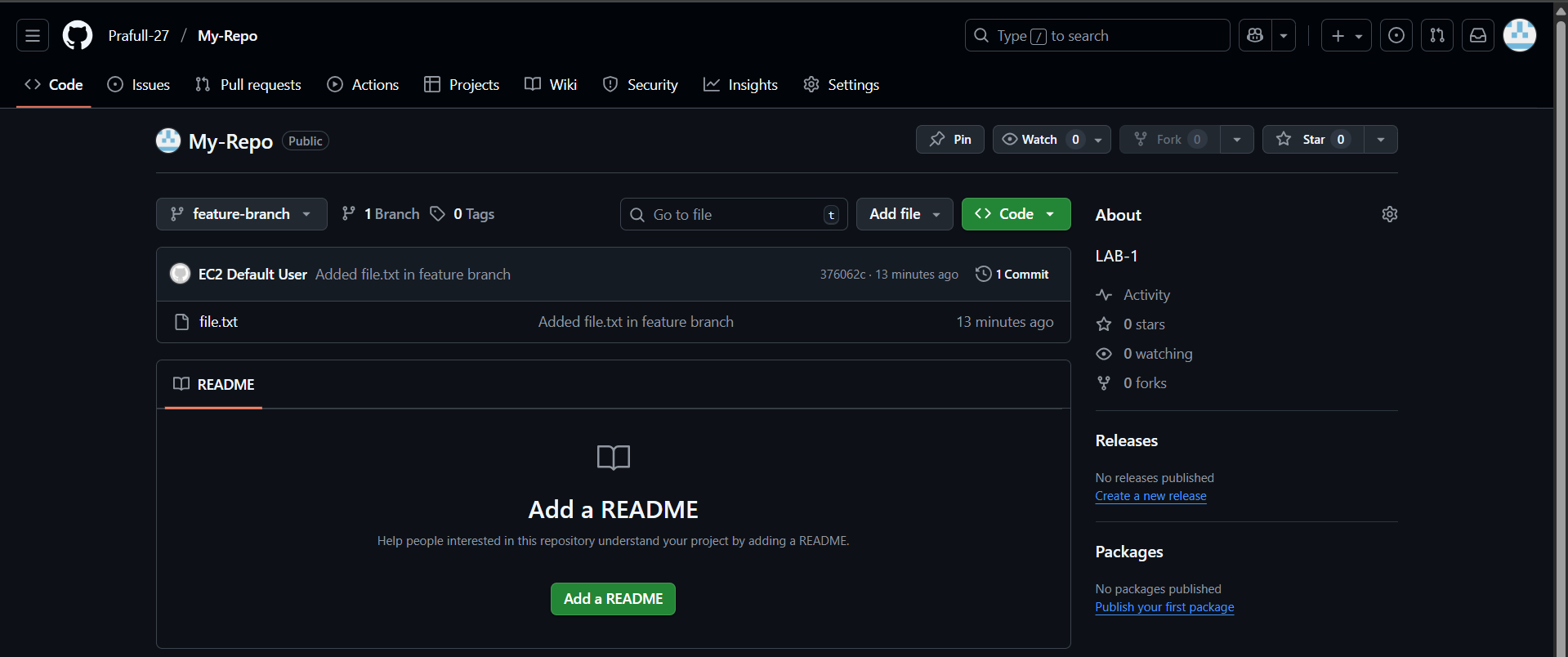


1. Push your branch to GitHub

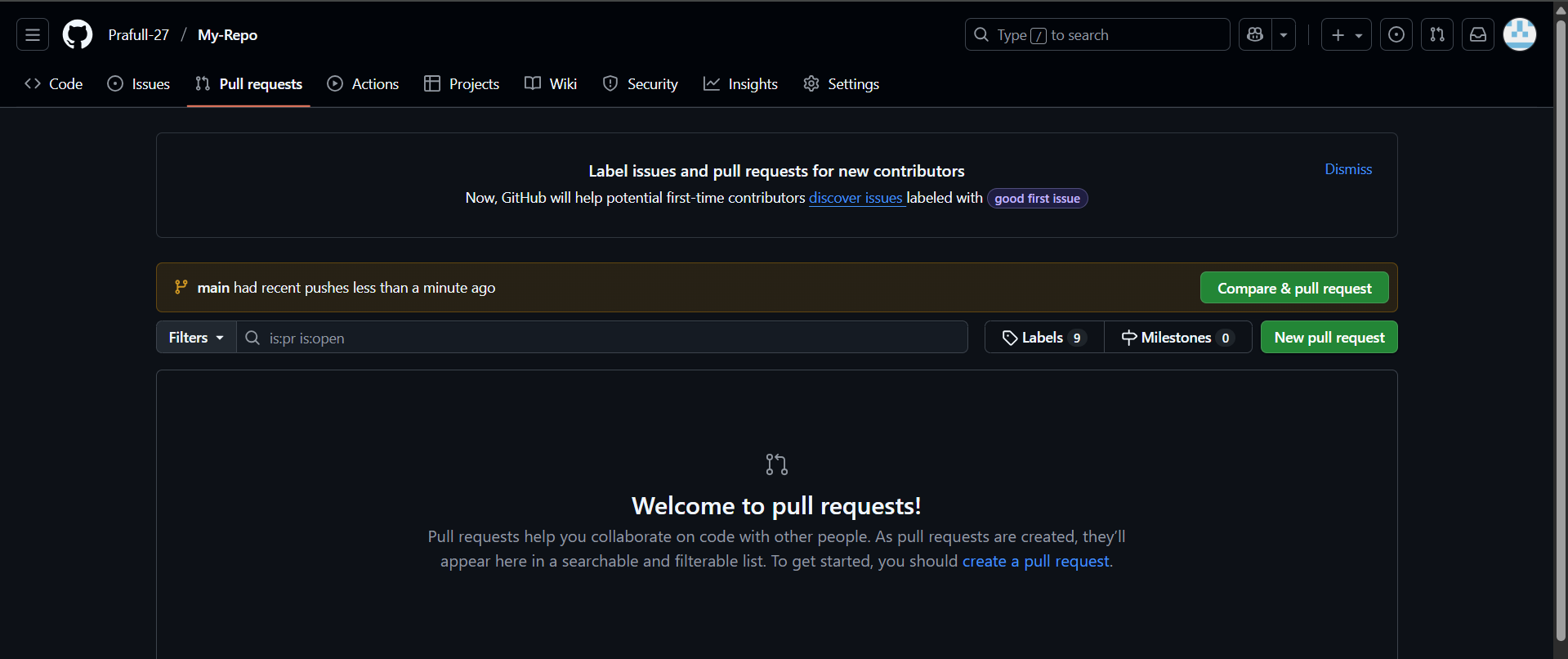


1. Create Pull Request (PR)

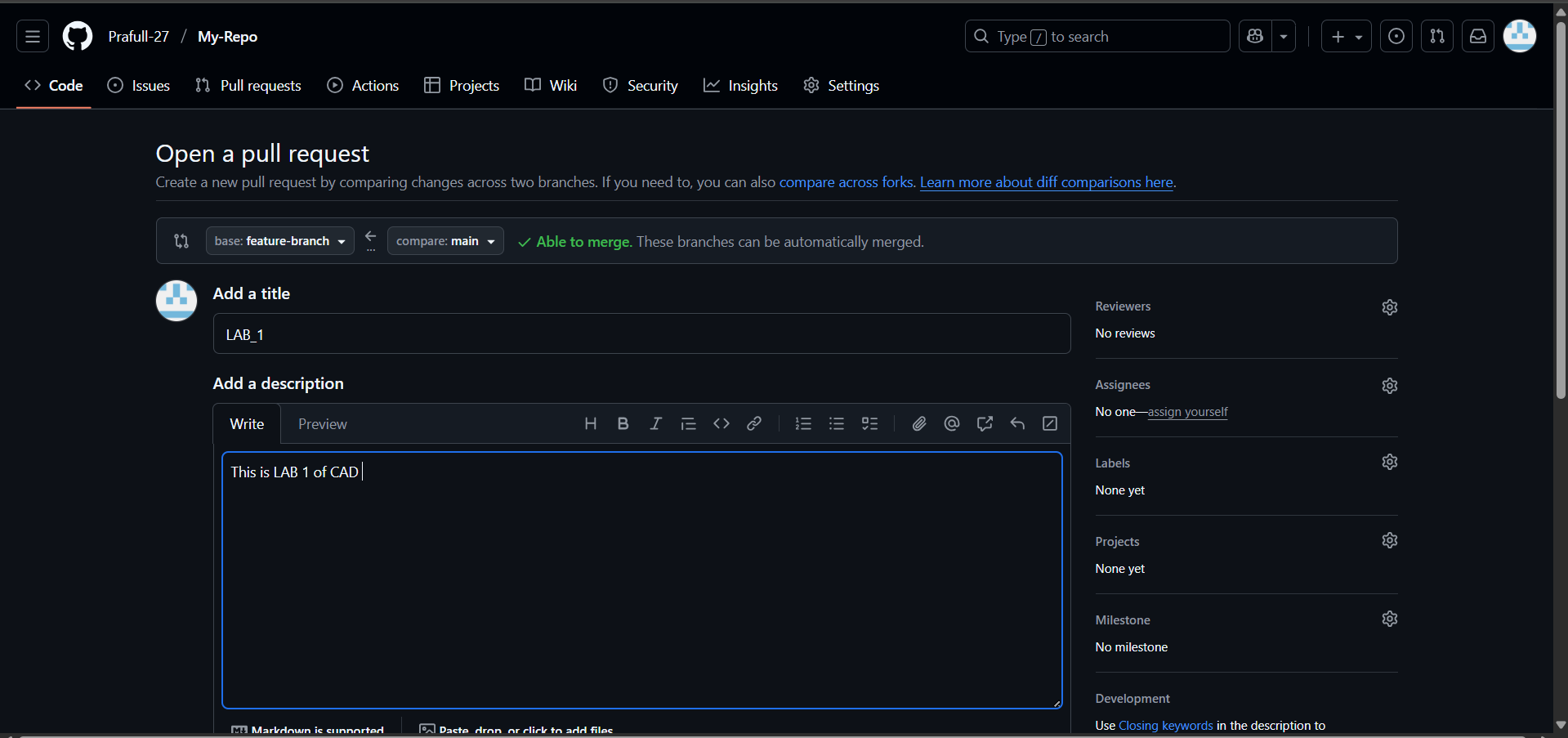
Go to your GitHub repo



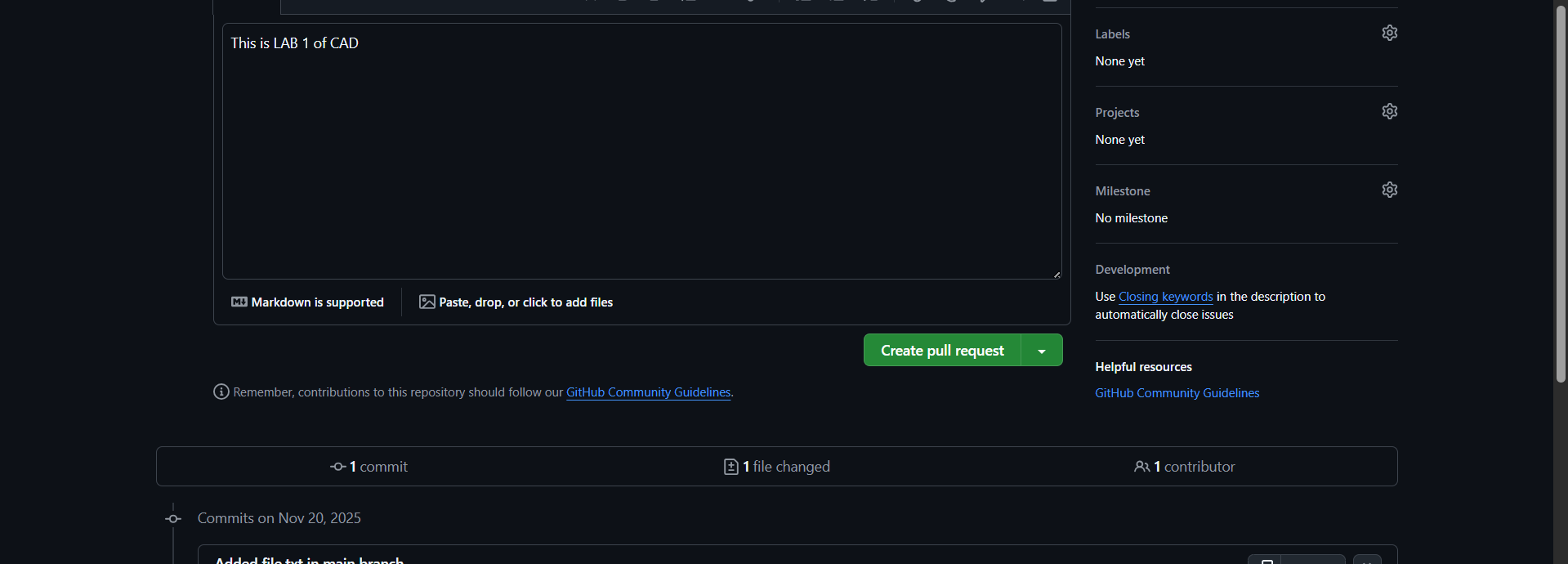
You will see a **Compare & Pull Request** button → click it



Add a title and description



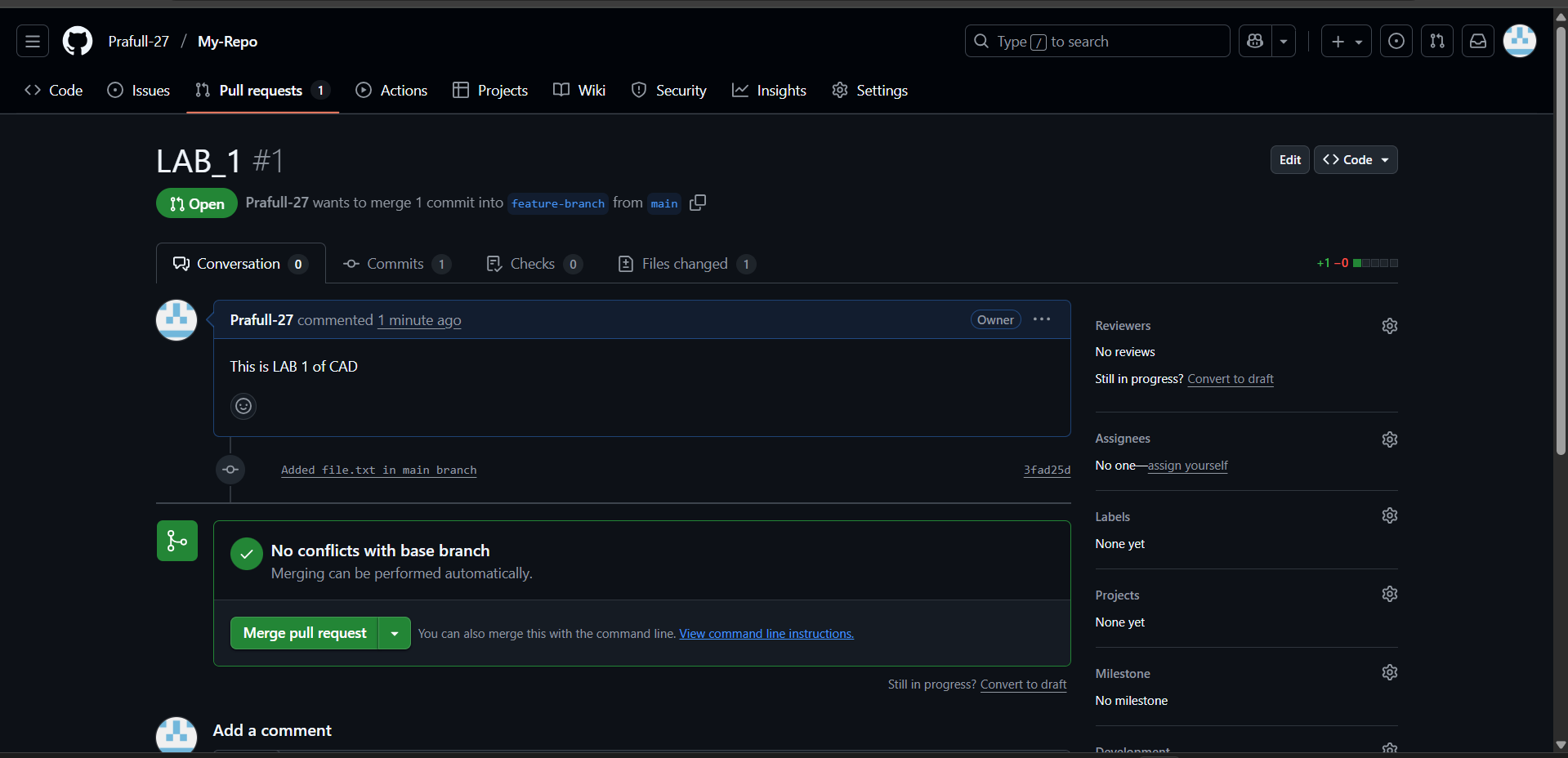
Click Create Pull Request

****

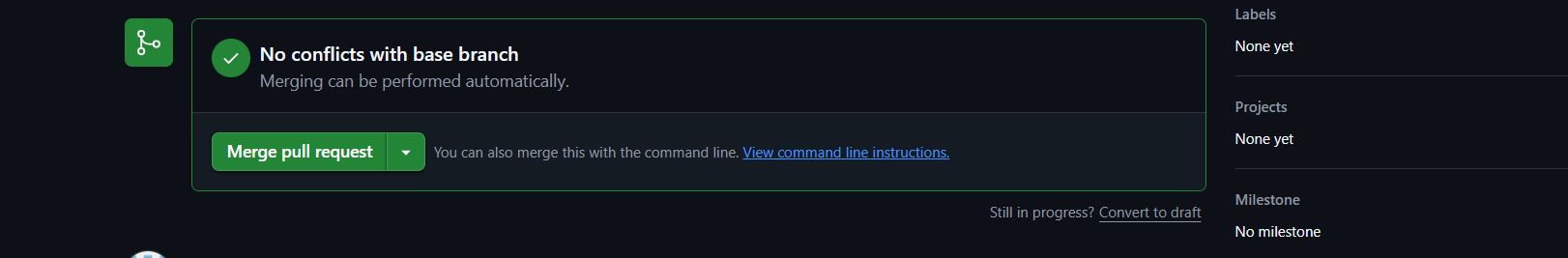
10.Review and Merge PR

On GitHub:

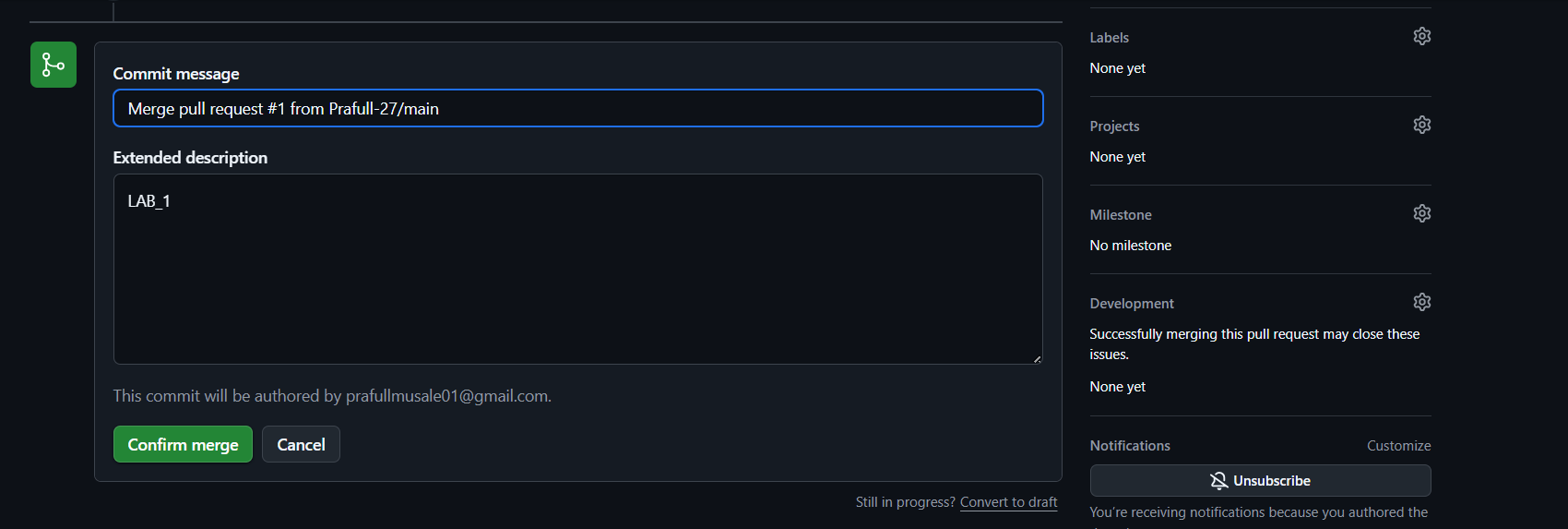
1. Open the PR



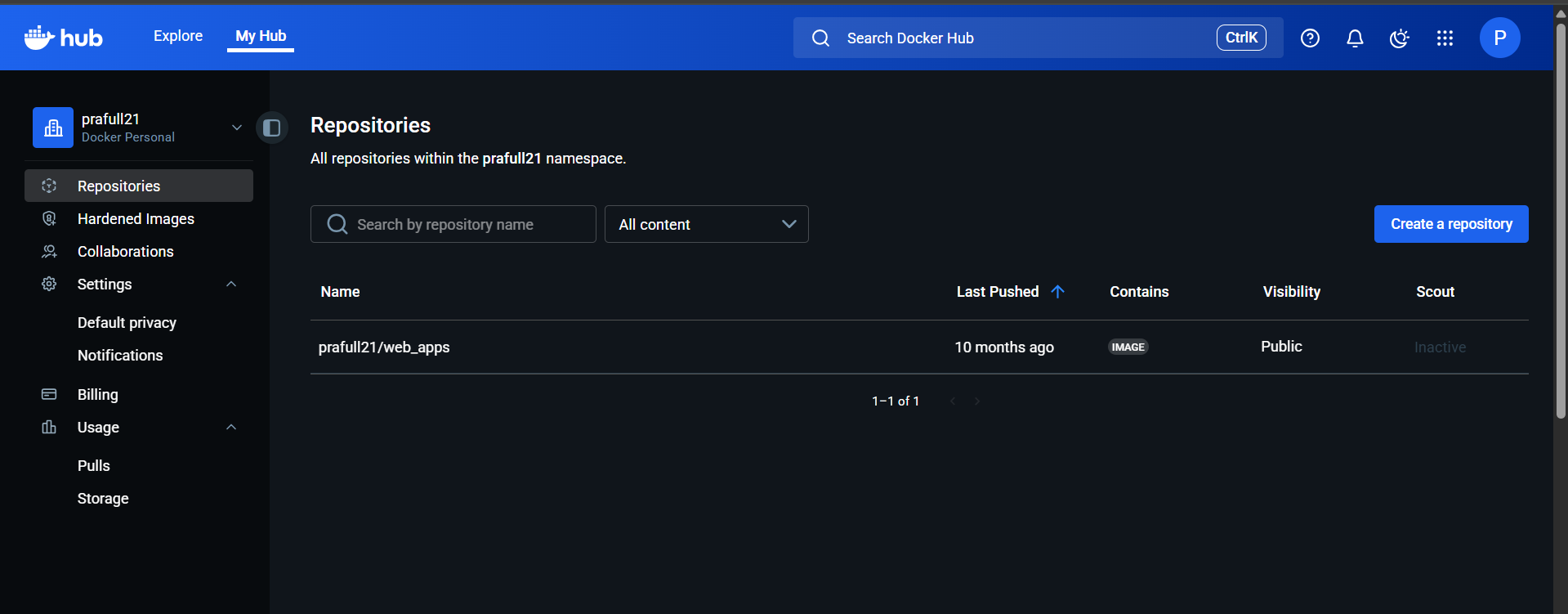
1. Click **Merge Pull Request**



1. Then click **Confirm Merge**

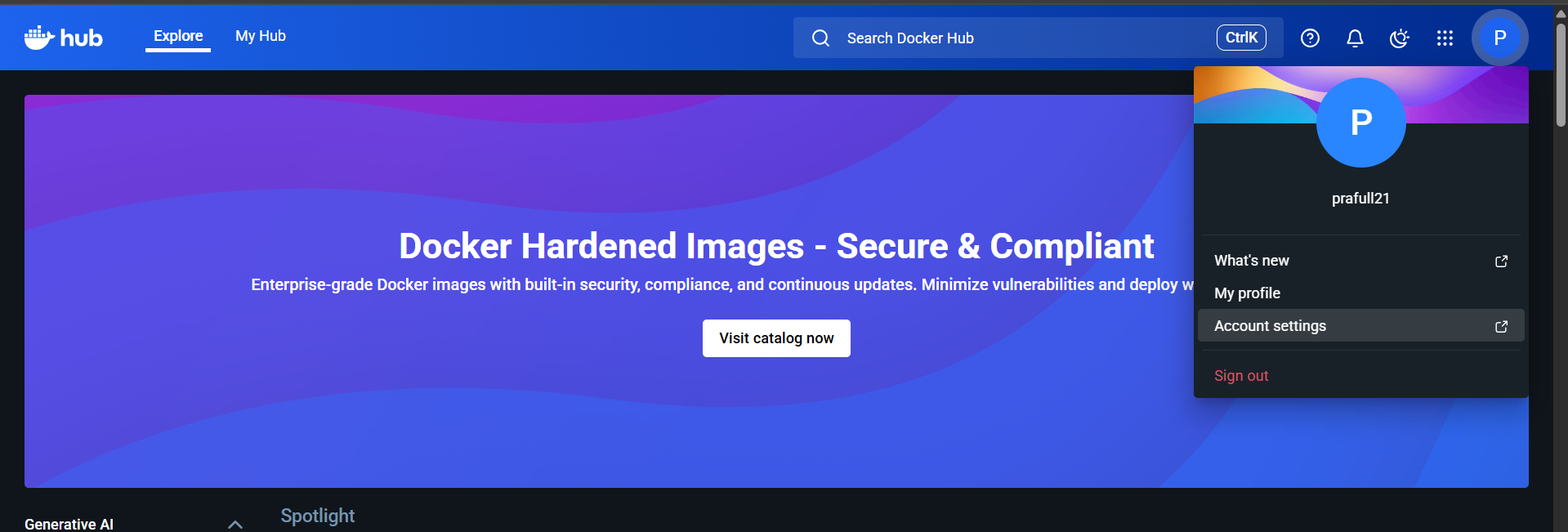


Step 1 — Create Docker Hub account.

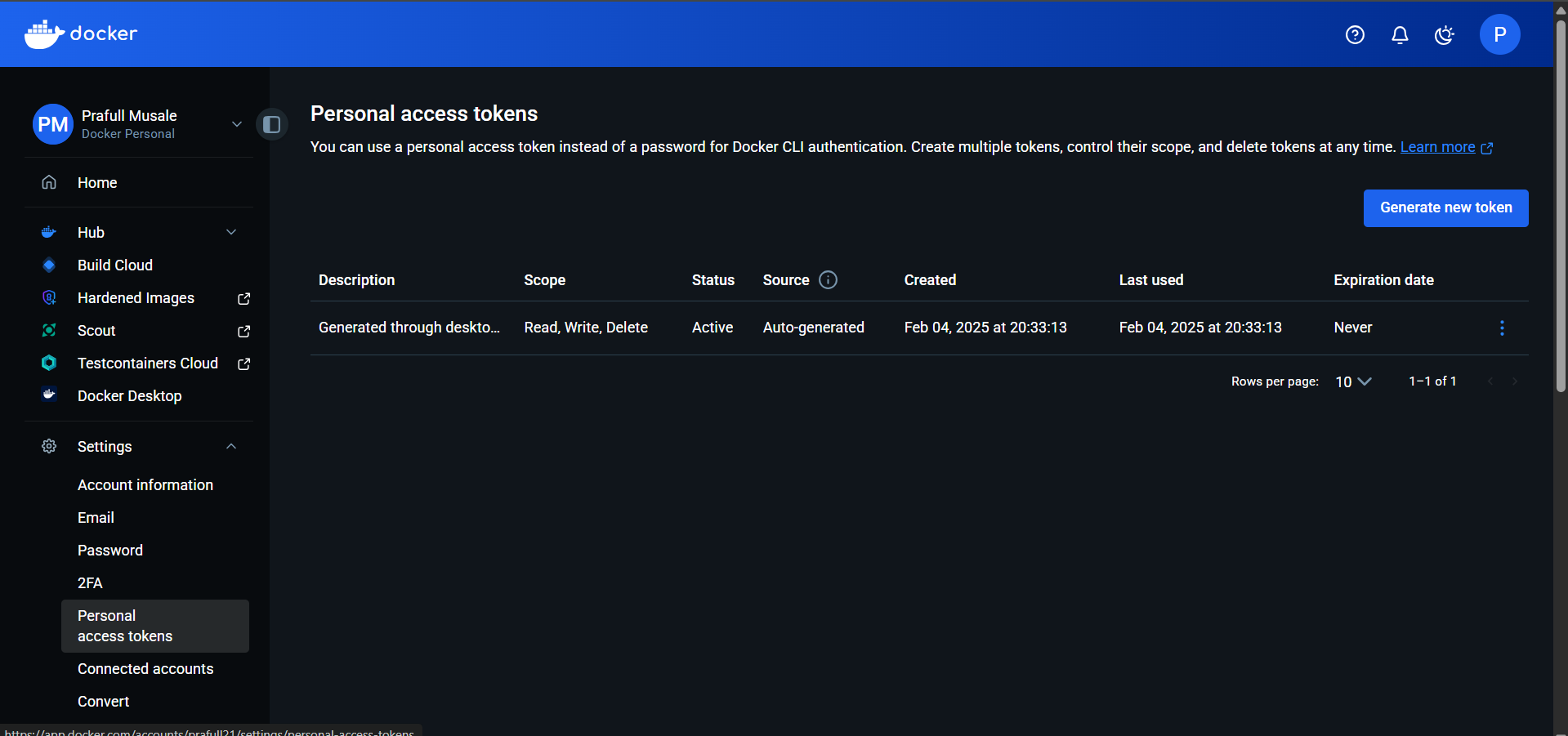


Step 2 — Generate Docker Hub Access Token

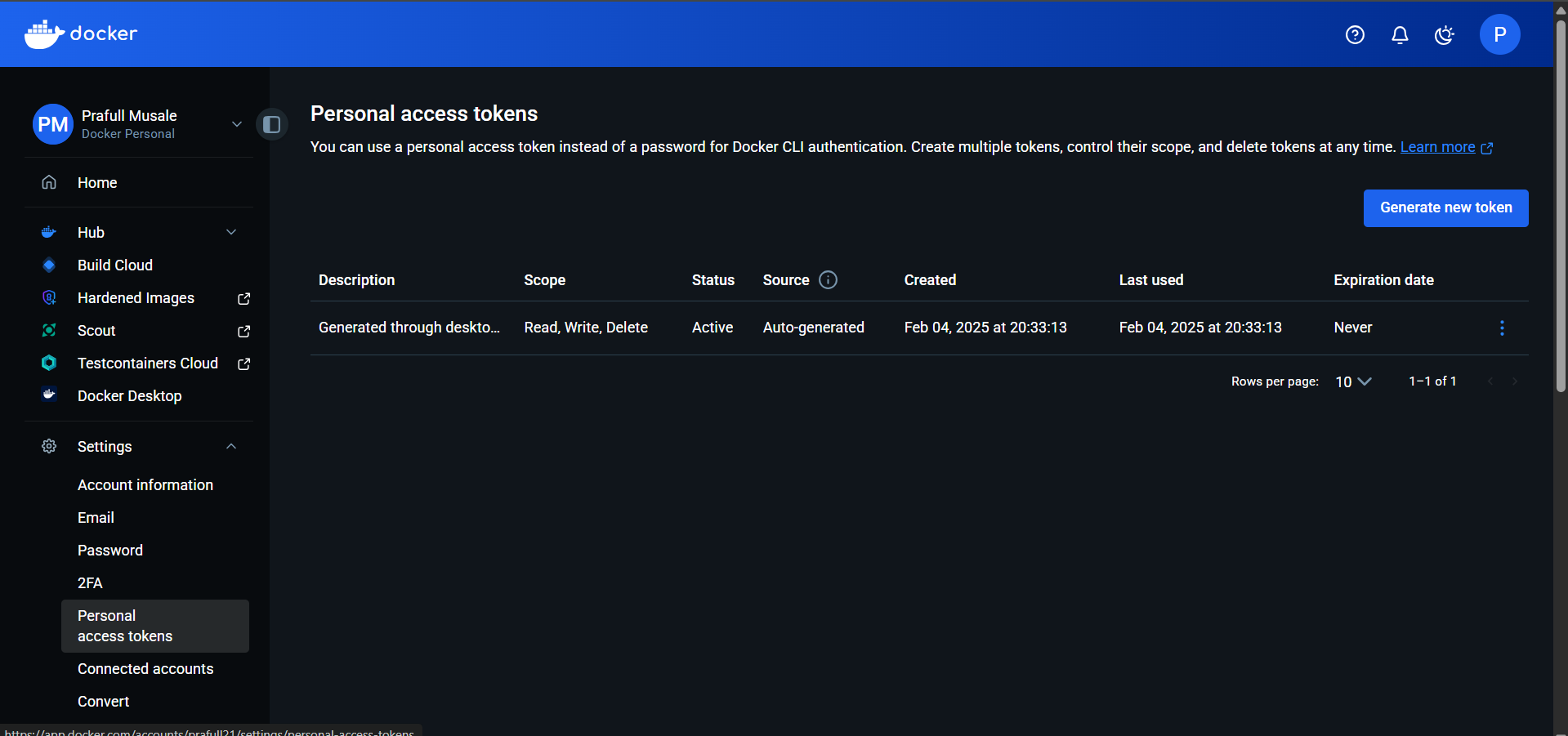
1. Go to **Account Settings**



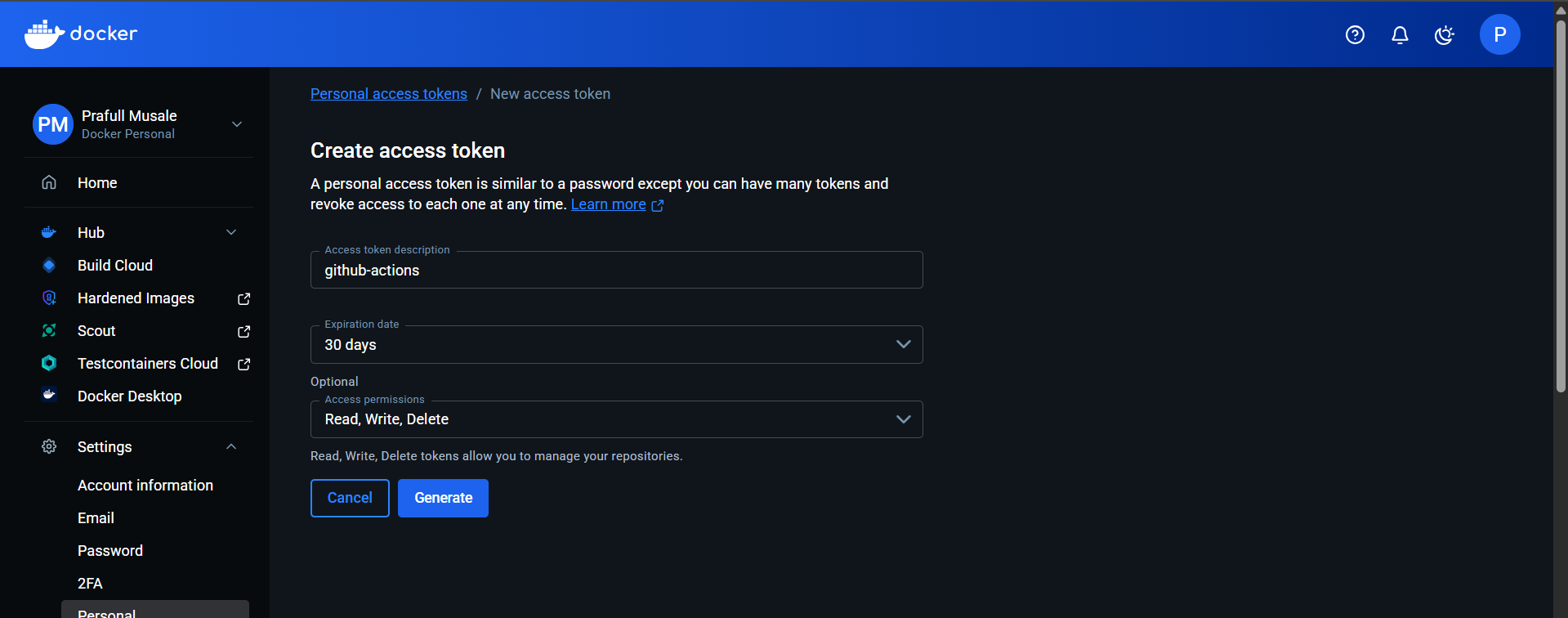
1. Click **Person Access Token**



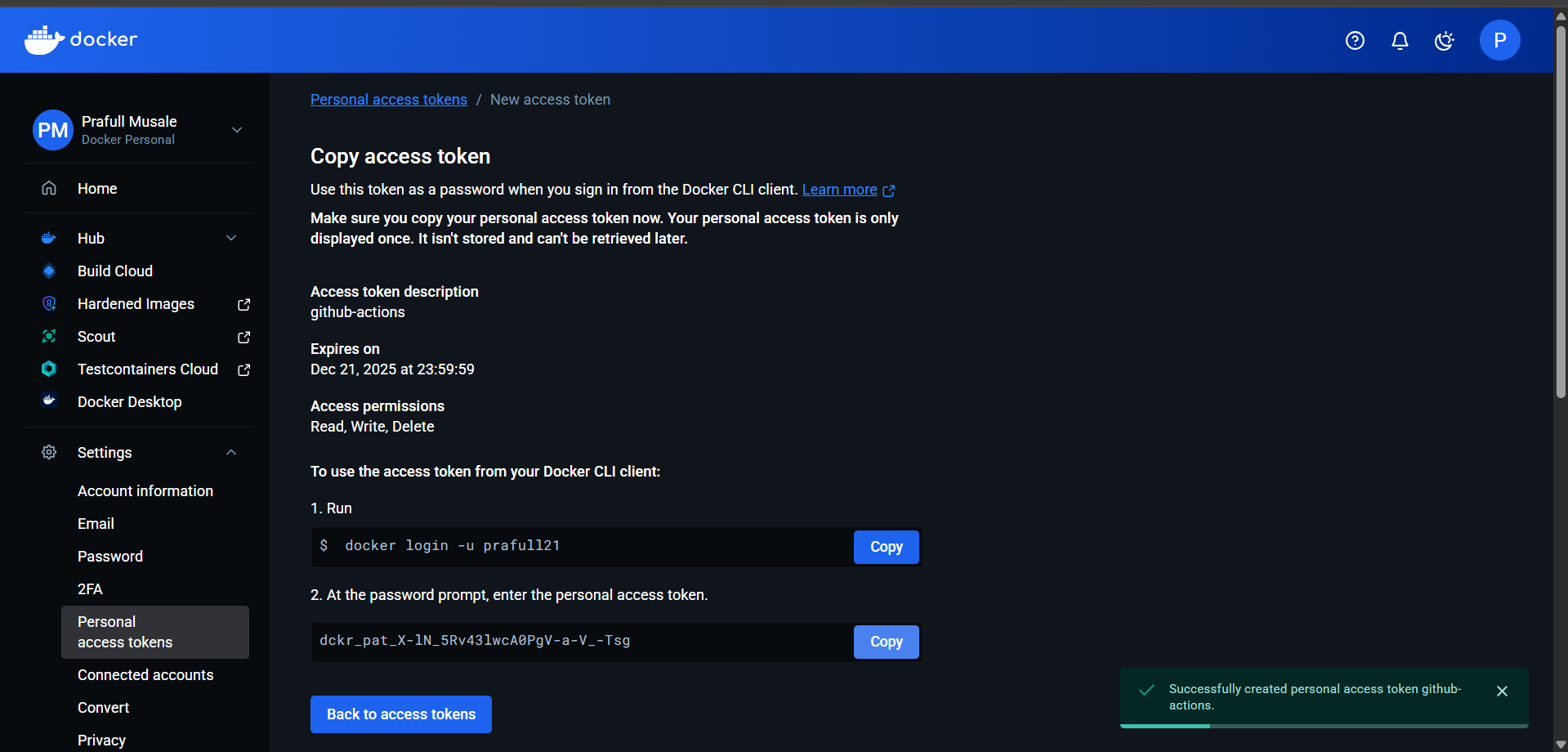
1. Click **New Access Token**



1. Name: github-actions

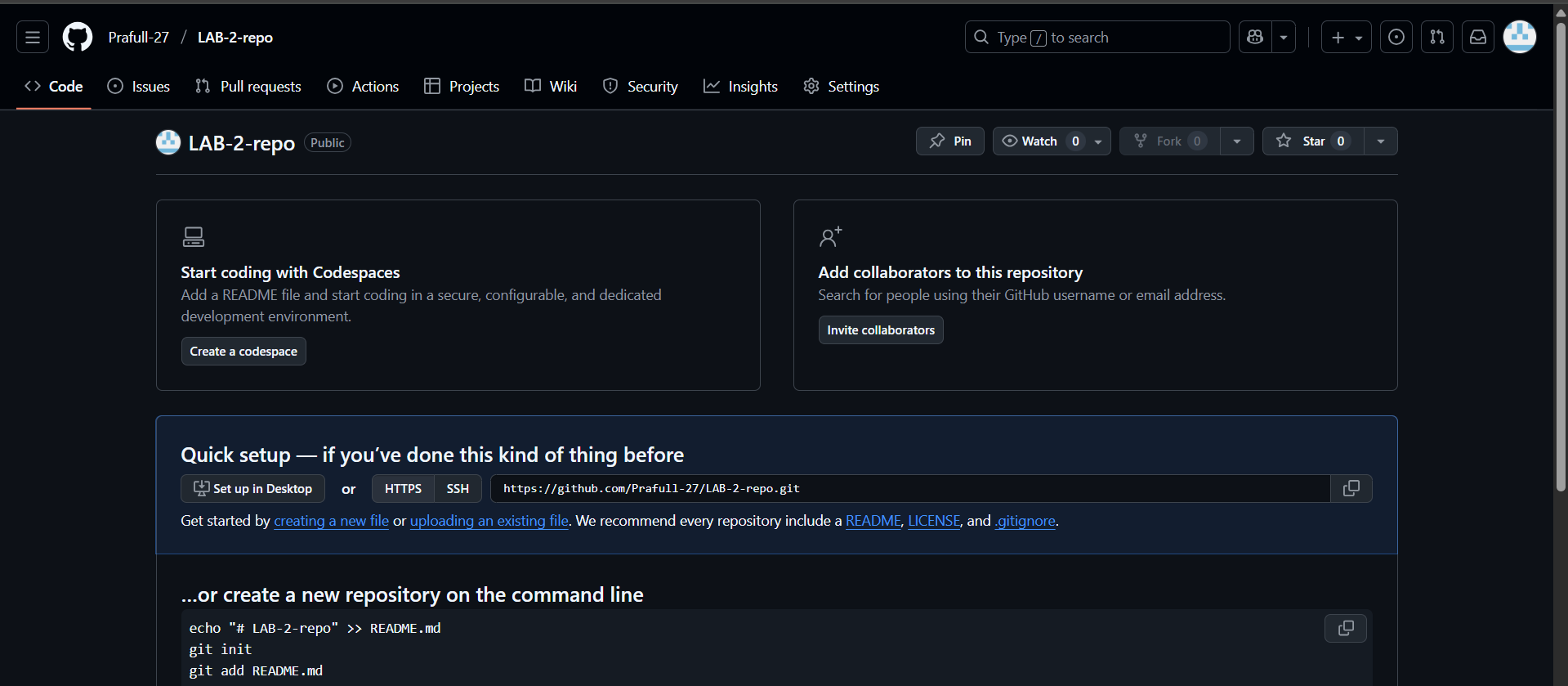


1. Copy the token (you will use it soon)



Step 3 — Add Secrets in GitHub Repository

1. Open your repository



1. Go to:  
   **Settings → Secrets and variables → Actions → New repository secret**

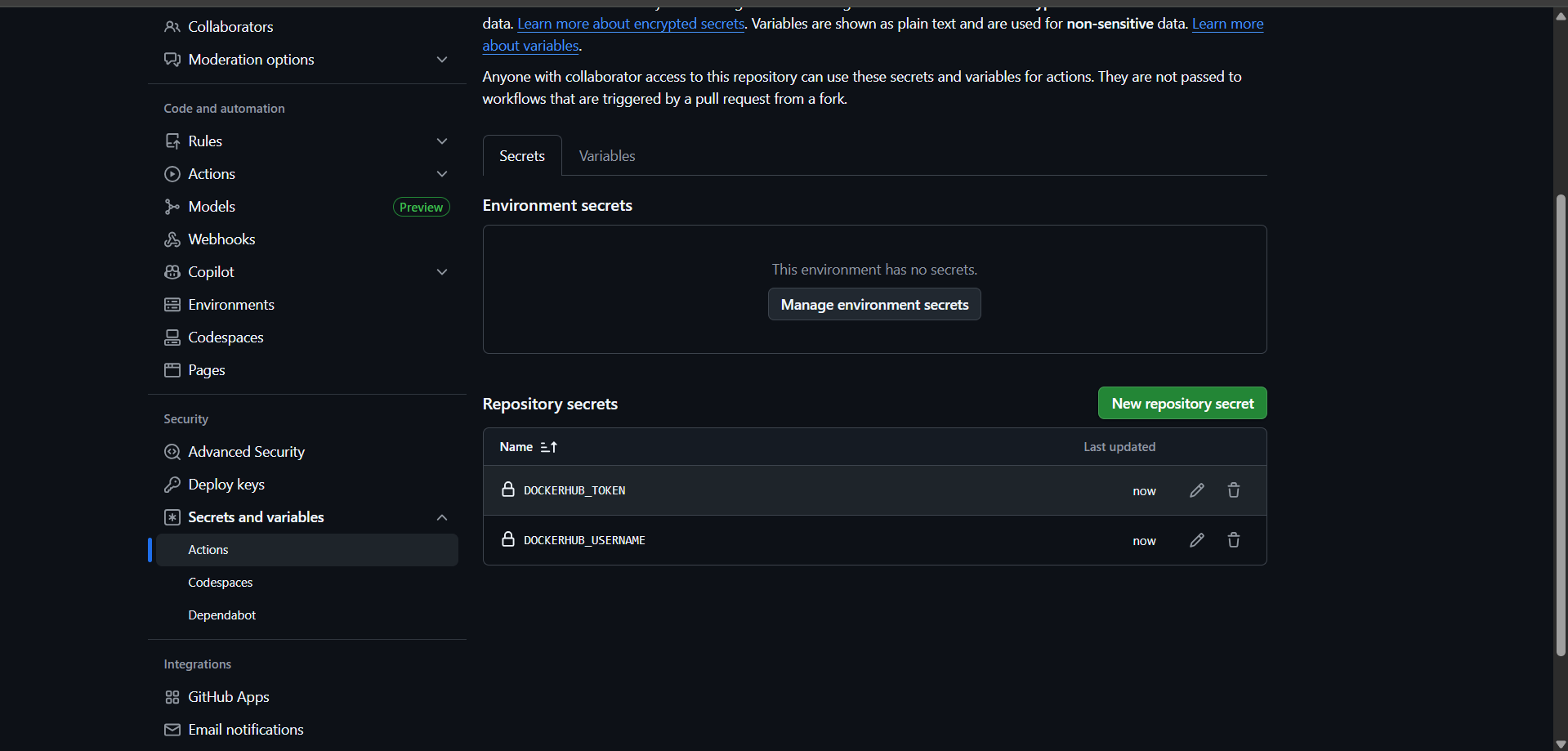
Create two secrets:

**Secret 1:**

* Name: DOCKERHUB\_USERNAME
* Value: your Docker Hub username

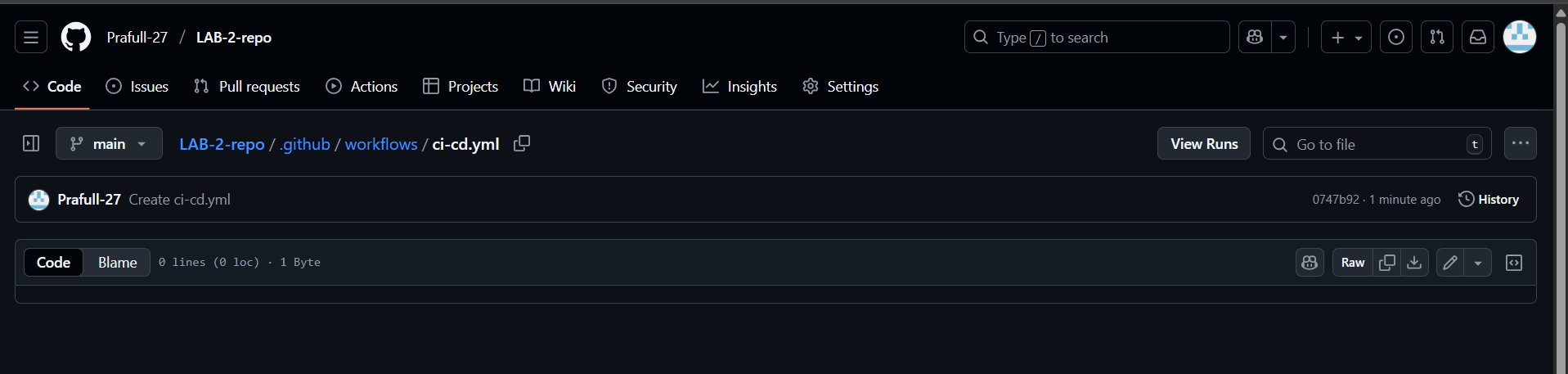
**Secret 2:**

* Name: DOCKERHUB\_TOKEN
* Value: the token you generated in Step 2



Step 4 — Create Workflow Folder

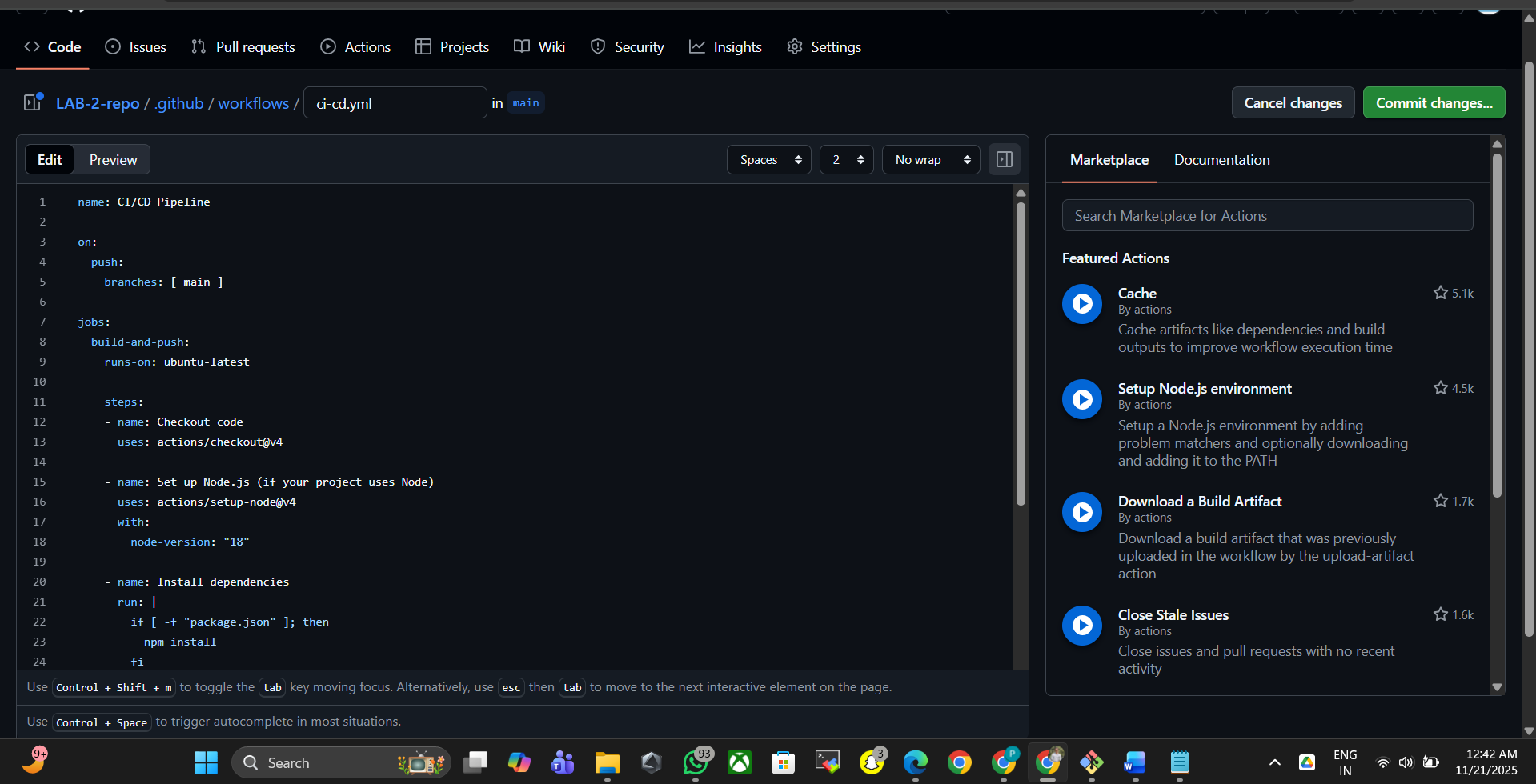
In your repository, create this folder: .github/workflows/



Step 5 — Create Workflow File (CI/CD File)

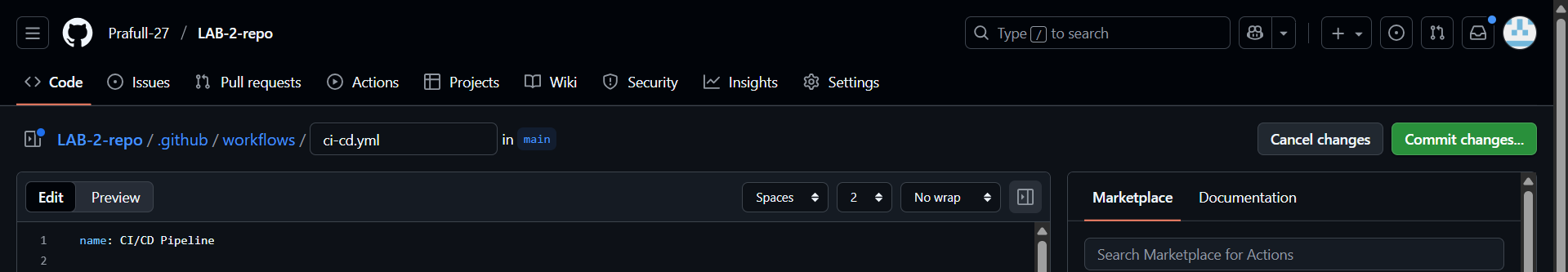
Inside .github/workflows/, create a file

Paste this code inside (simple CI/CD pipeline)



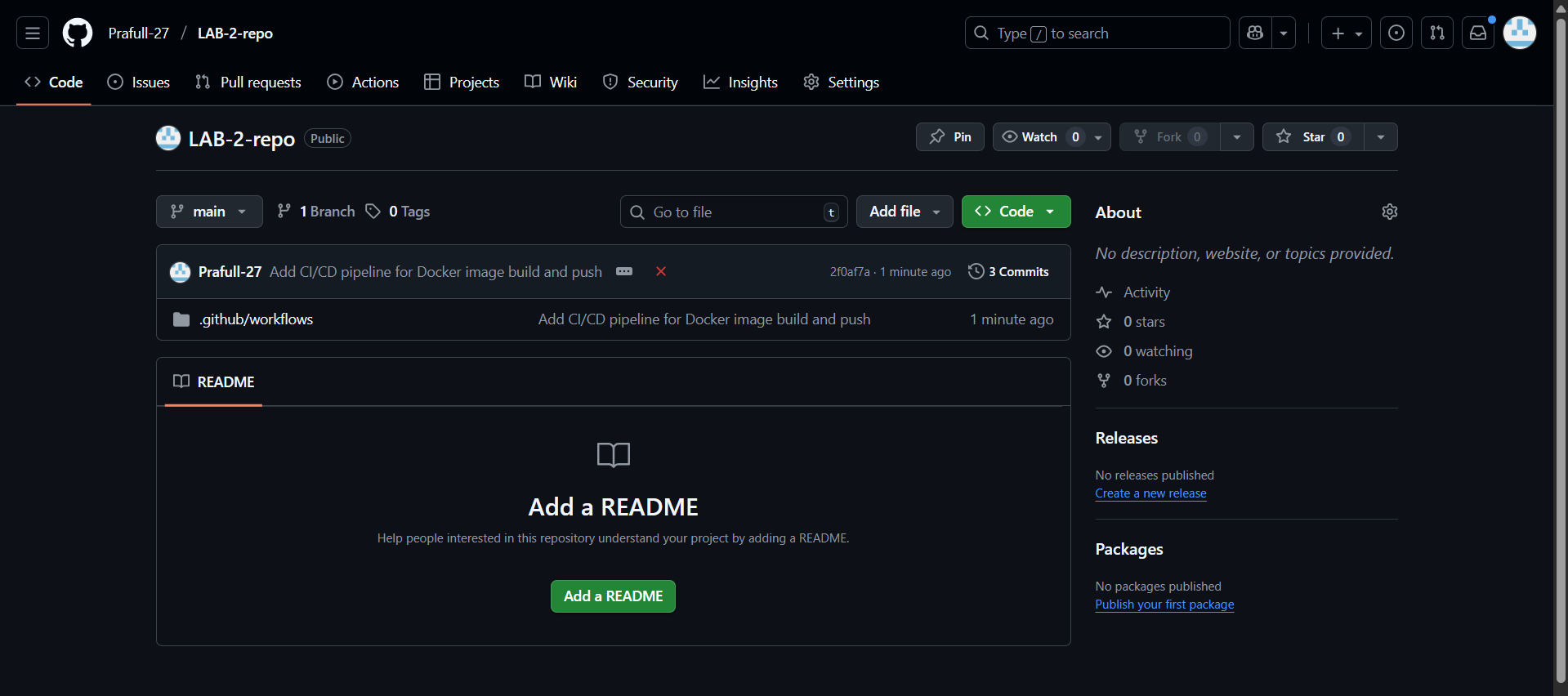
**Step 6 — Commit & Push**

If you push to the **main branch**, the pipeline runs automatically.

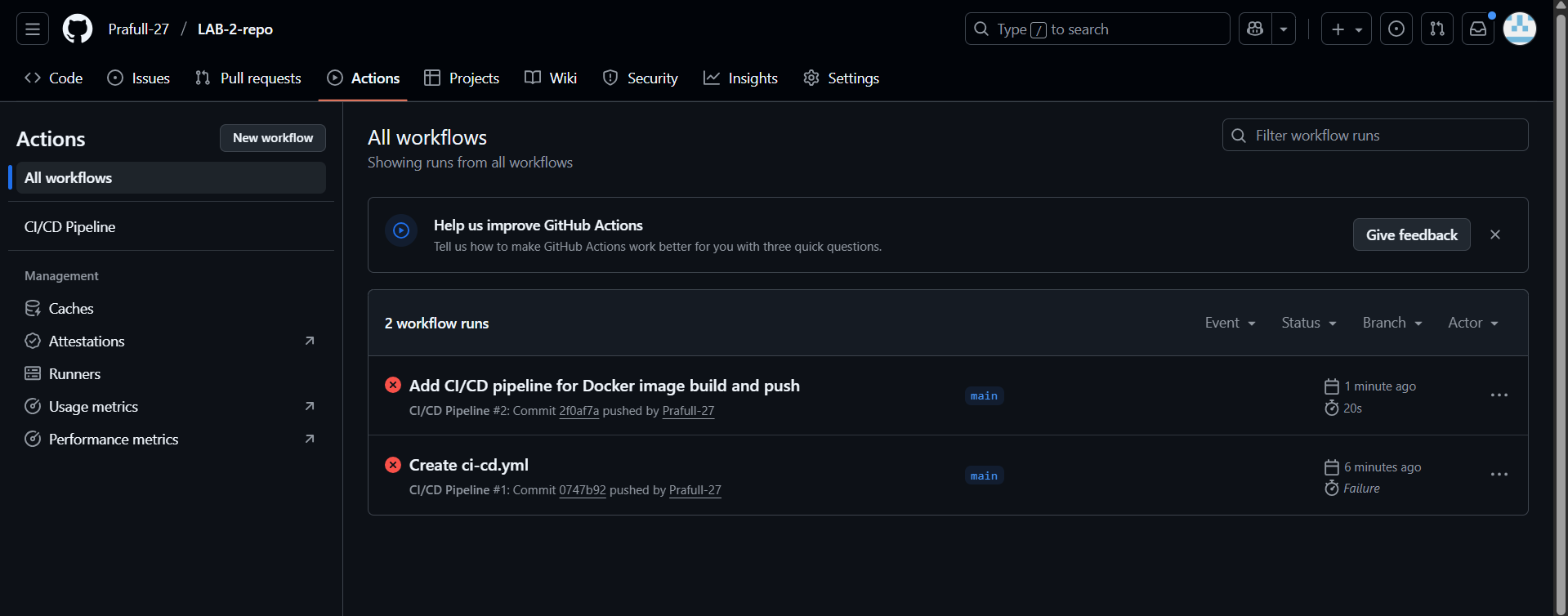


**Step 7 — Check the GitHub Actions Pipeline**

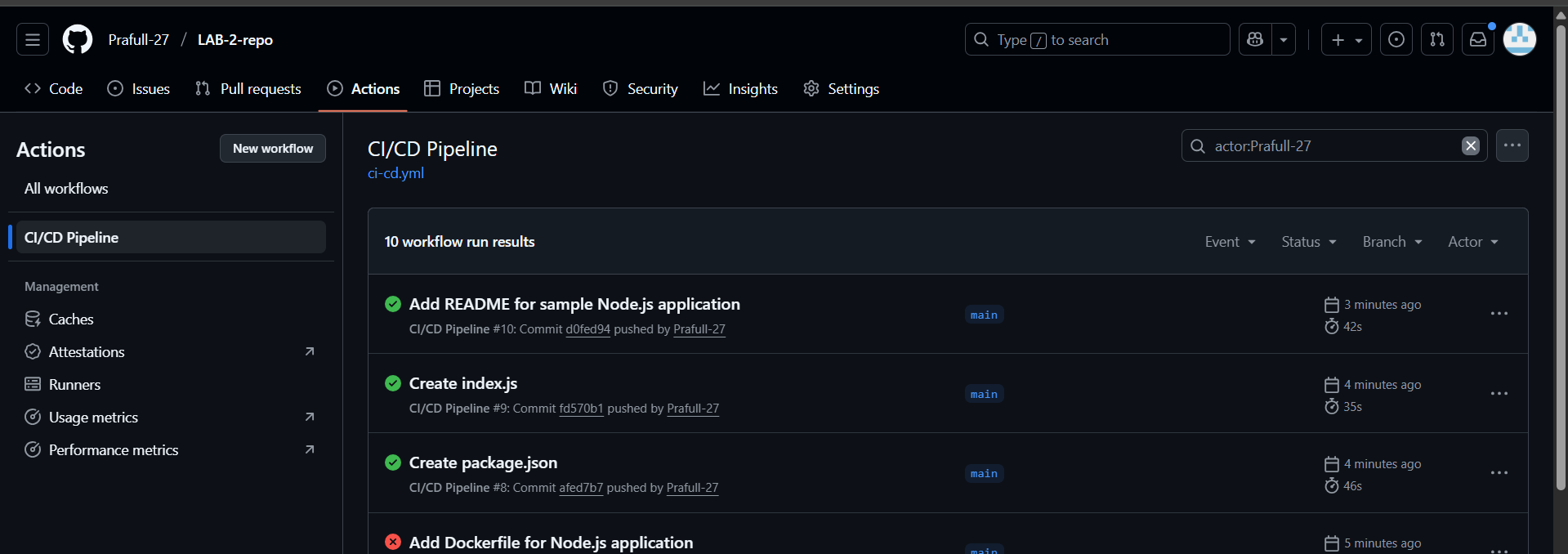
1. Go to your repository



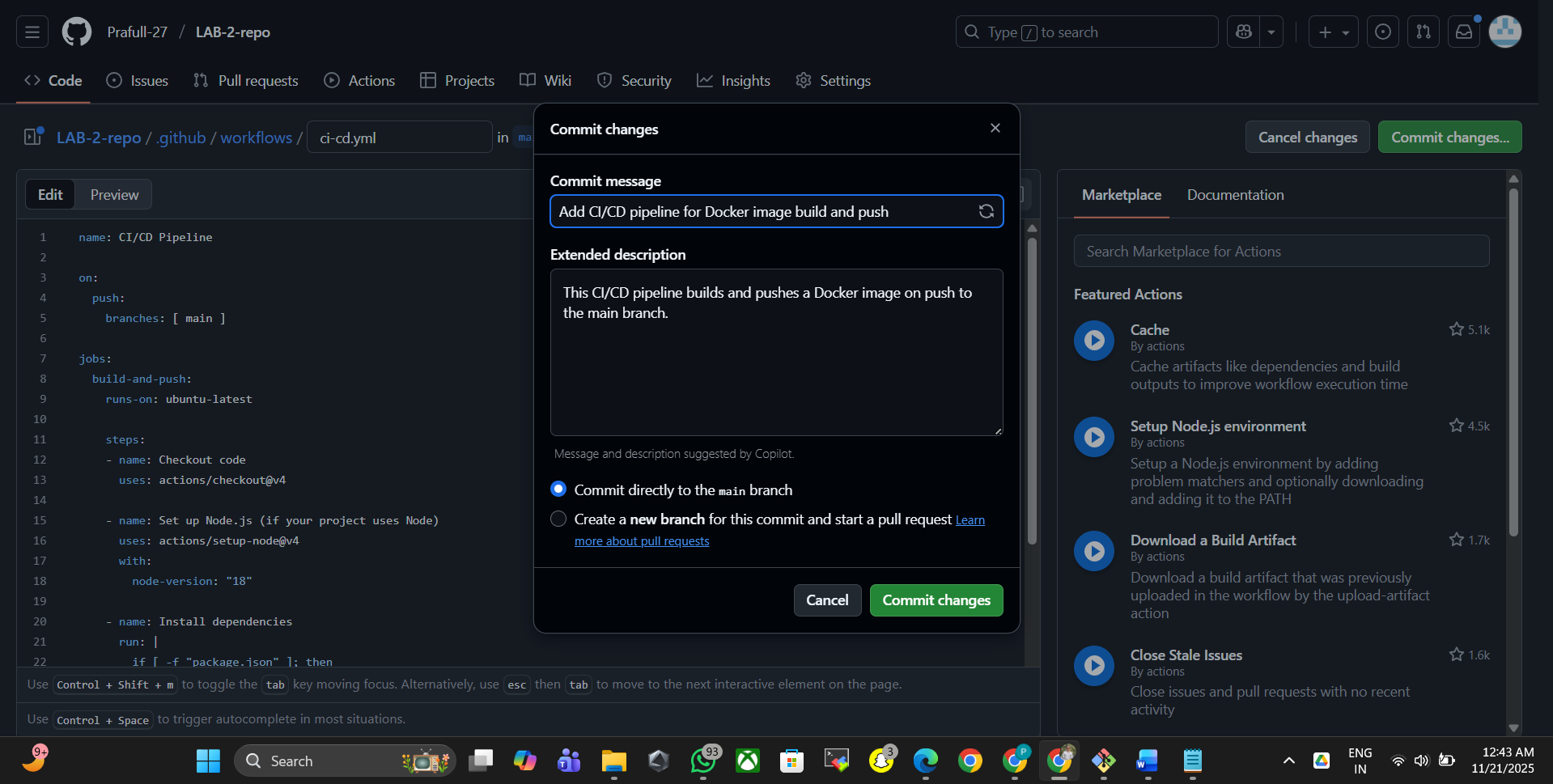
1. Click **Actions**



1. You will see a new workflow running



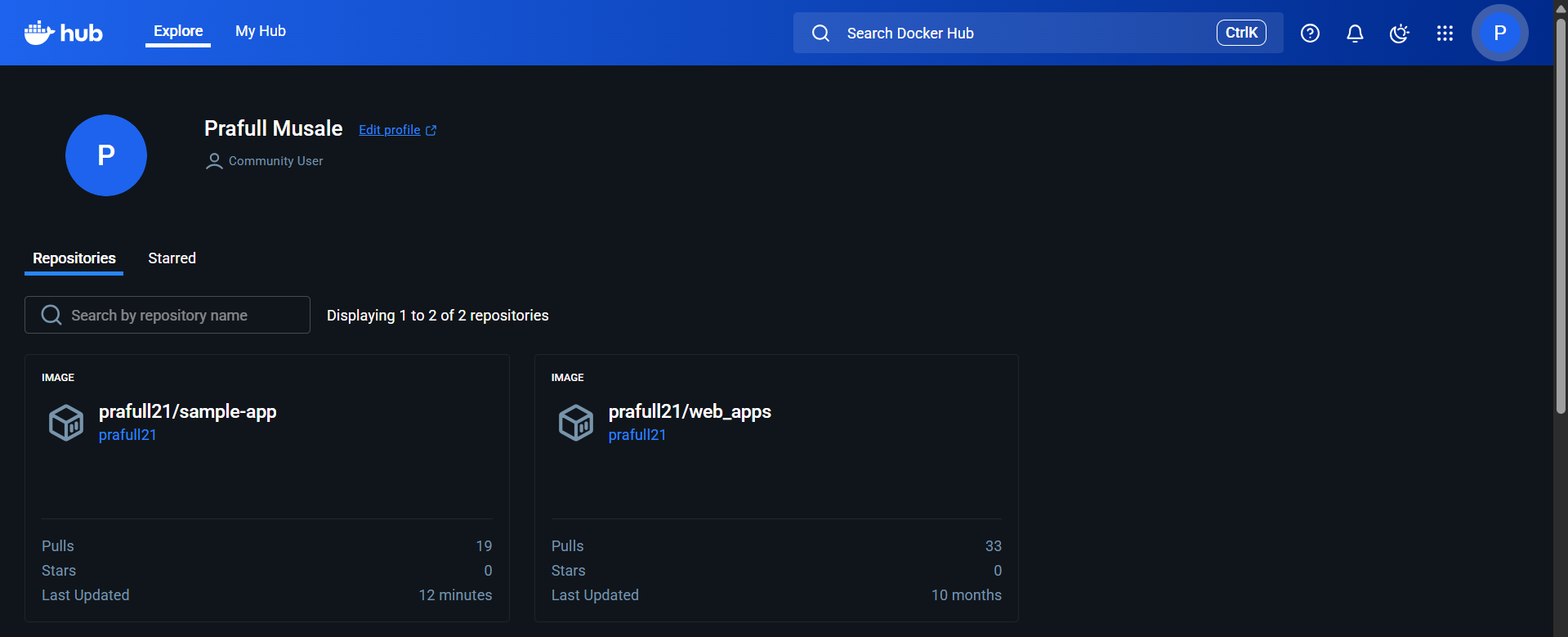
1. Click it → you can see all logs: build, test, docker push, etc.



**Step 8 — Confirm Docker Image in Docker Hub**

Go to Docker Hub → Repositories →  
You will see:

* sample-app:latest
* sample-app:<build-number>

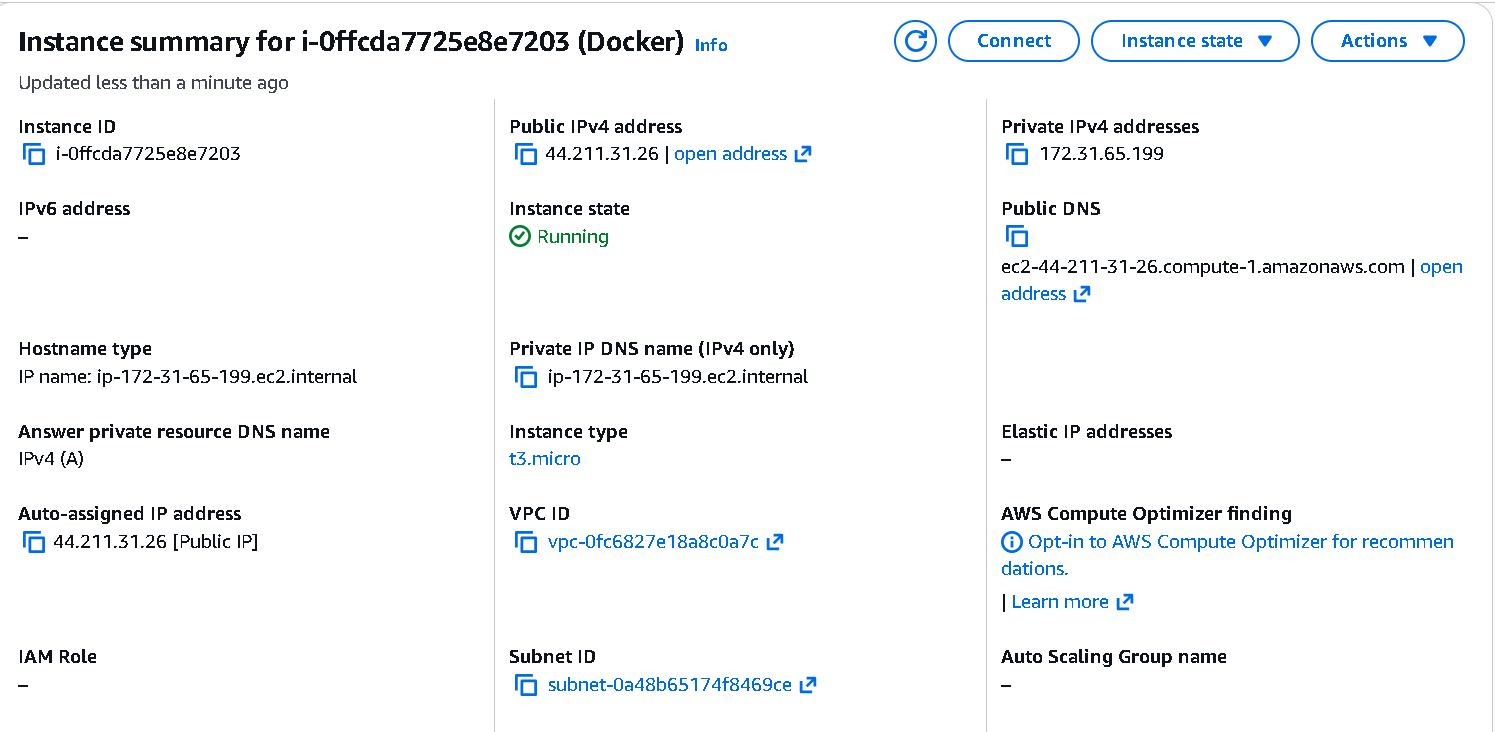


1. **Launch the instance.**

Name: Docker

AMI: Amazon Linux2023

Instance Type: t3.micro



1. **Install the Docker & Docker Compose.**

sudo yum update -y

sudo dnf install docker -y

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker ec2-user

sudo usermod -aG docker $USER

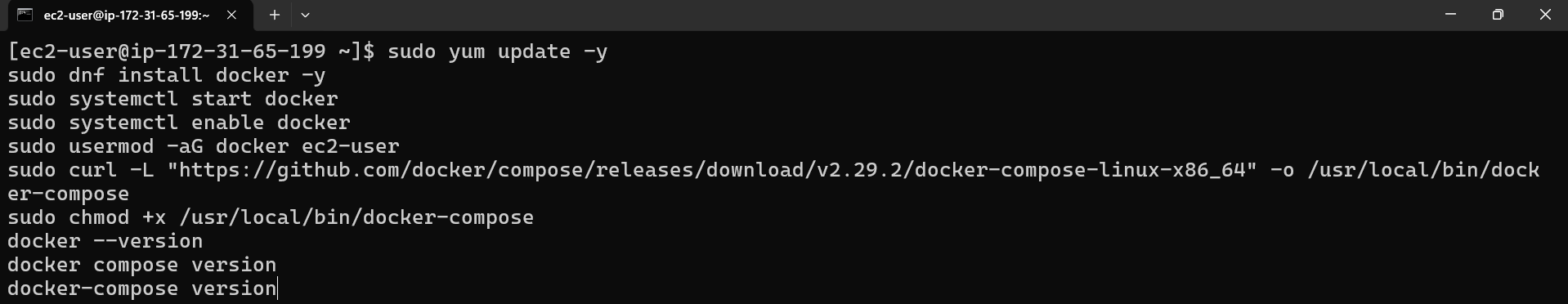
sudo curl -L "https://github.com/docker/compose/releases/download/v2.29.2/docker-compose-linux-x86\_64" -o /usr/local/bin/docker-compose

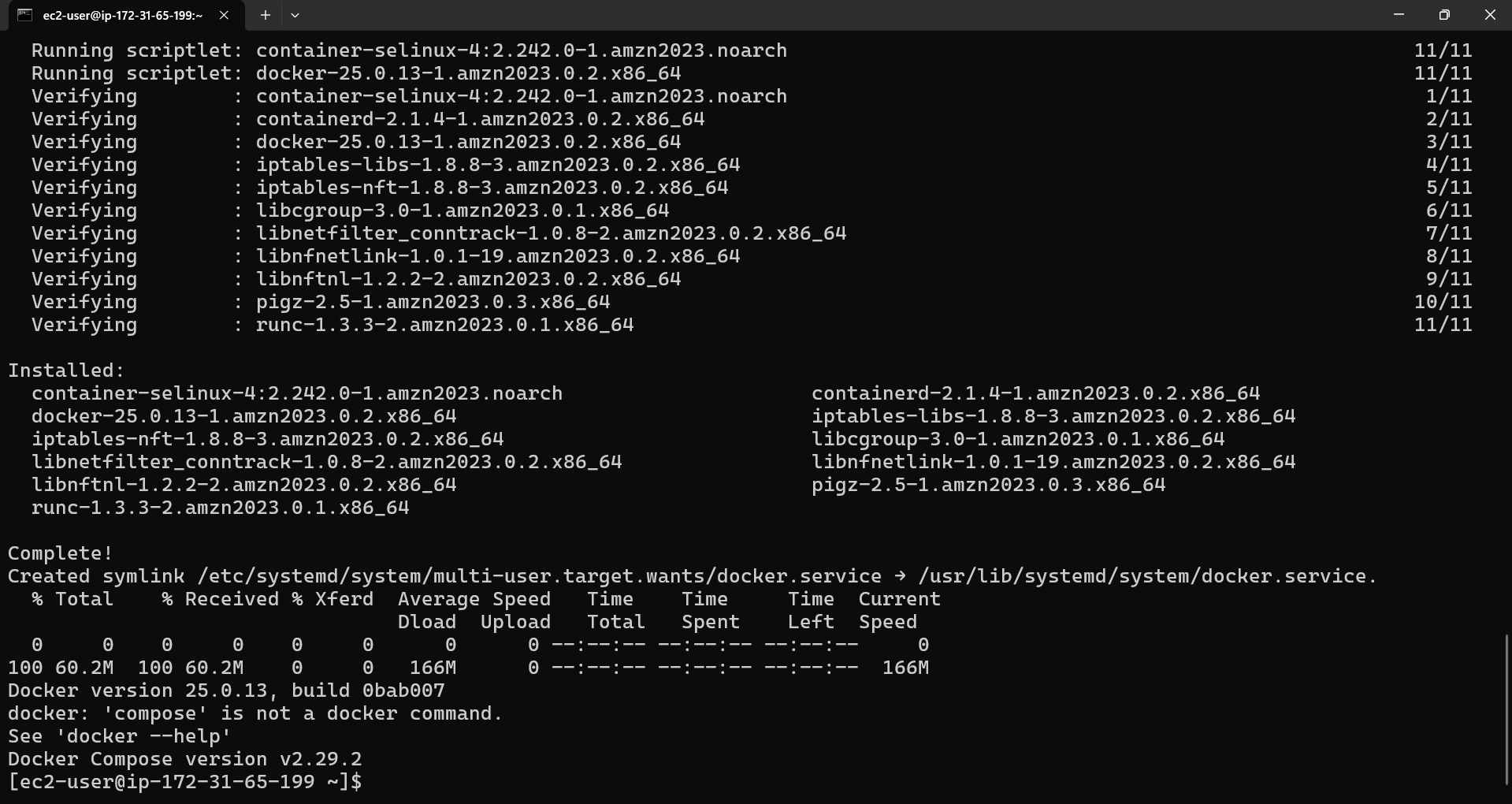
sudo chmod +x /usr/local/bin/docker-compose

docker --version

docker compose version

docker-compose version

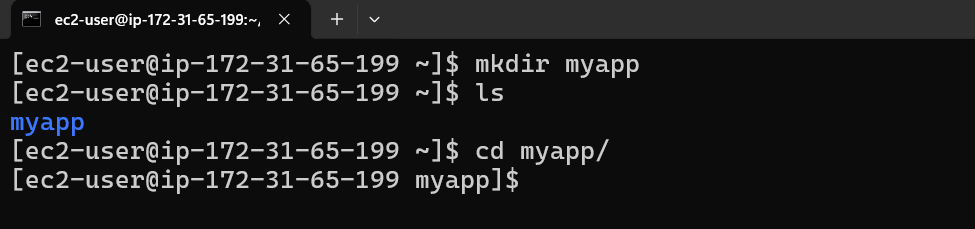




1. **Create Your Project Folder**

mkdir myapp

cd myapp



1. **Add Your Application Code**

Example: Node.js Express app

**Create app.js**

**$ vi app.js**

const express = require("express");

const app = express();

app.get("/", (req, res) => res.send("Hello from Docker!"));

app.listen(3000, () => console.log("Server running on port 3000"));



Create package.json

**$ package.json**

{

"name": "docker-app",

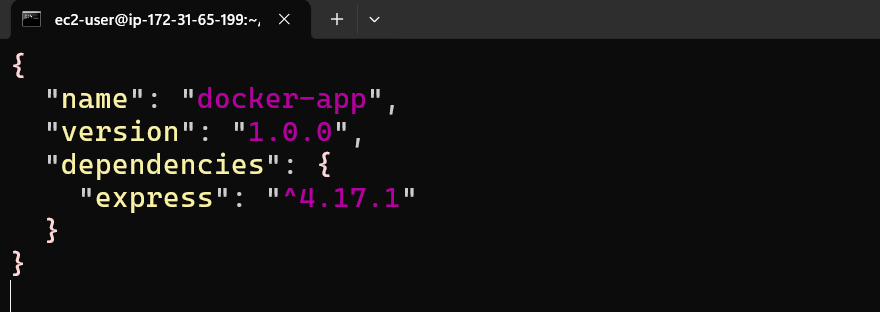
"version": "1.0.0",

"dependencies": {

"express": "^4.17.1"

}

}



1. **Write the Dockerfile**

Create Dockerfile

**$ vi dockerfile**

FROM node:18

WORKDIR /app

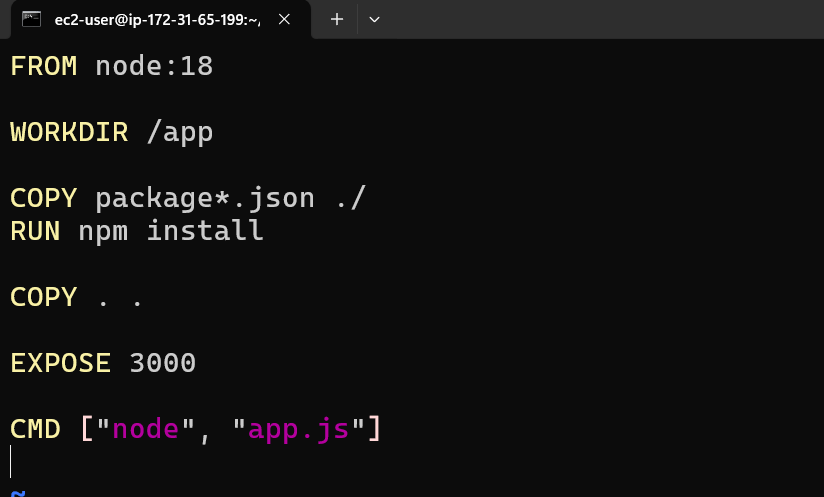
COPY package\*.json ./

RUN npm install

COPY . .

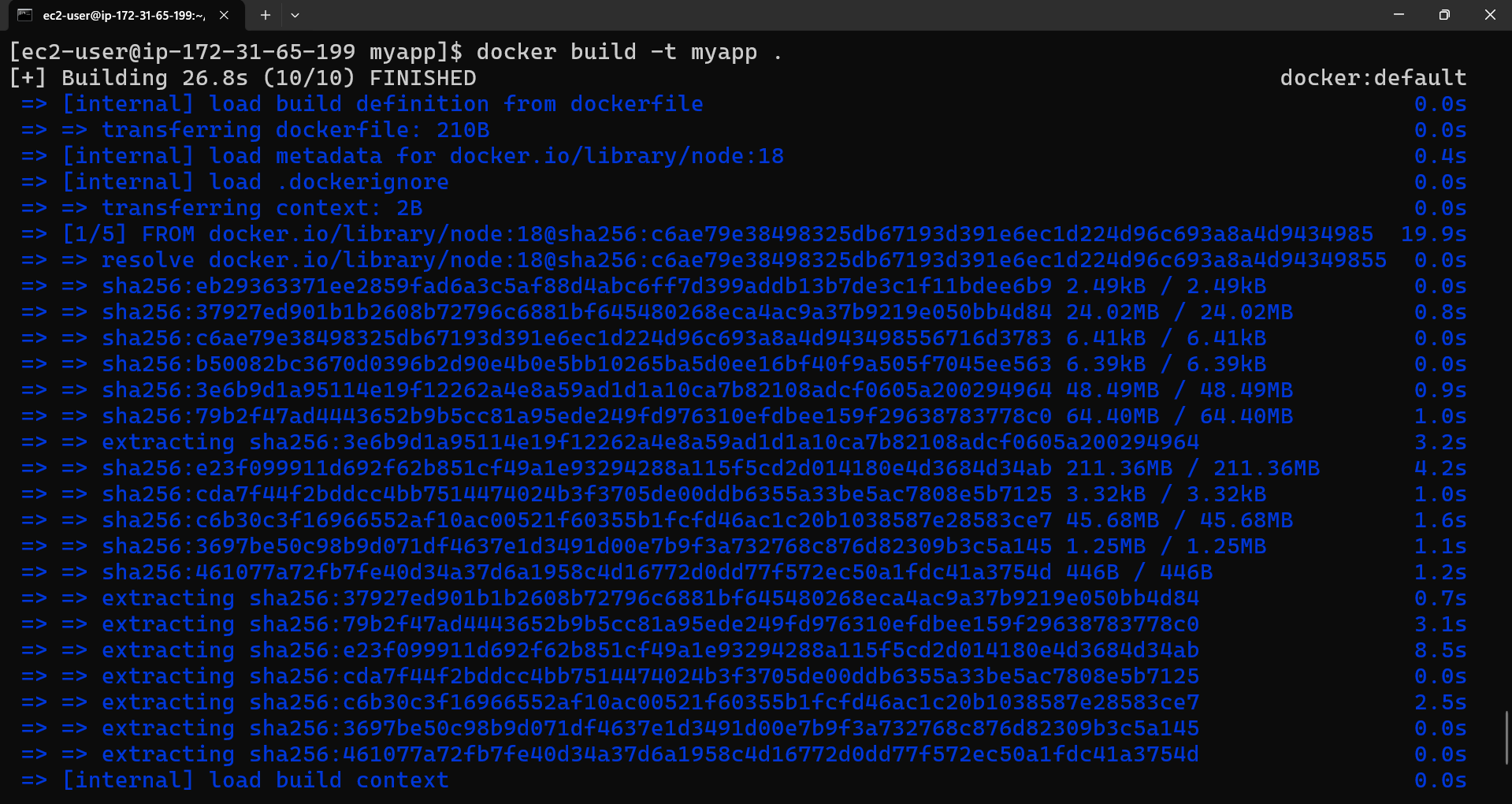
EXPOSE 3000

CMD ["node", "app.js"]



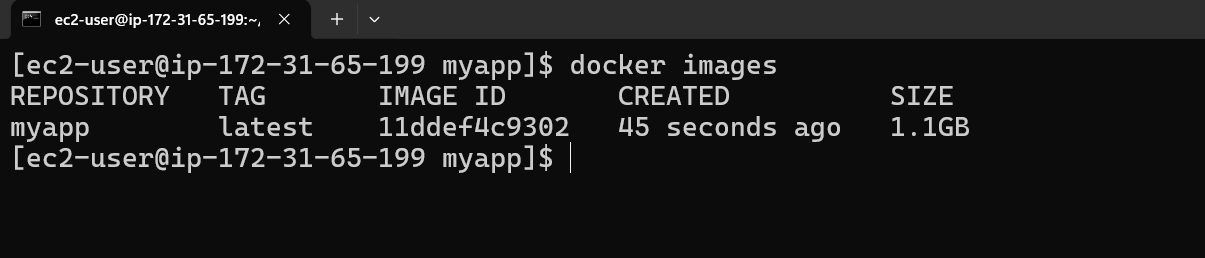
1. **Build Docker Image**

$ docker build -t myapp .



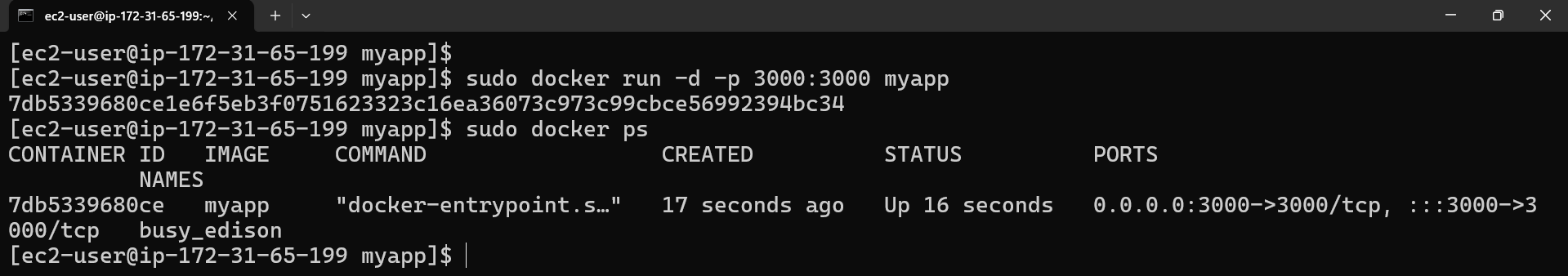
**Check image:**

$ docker images



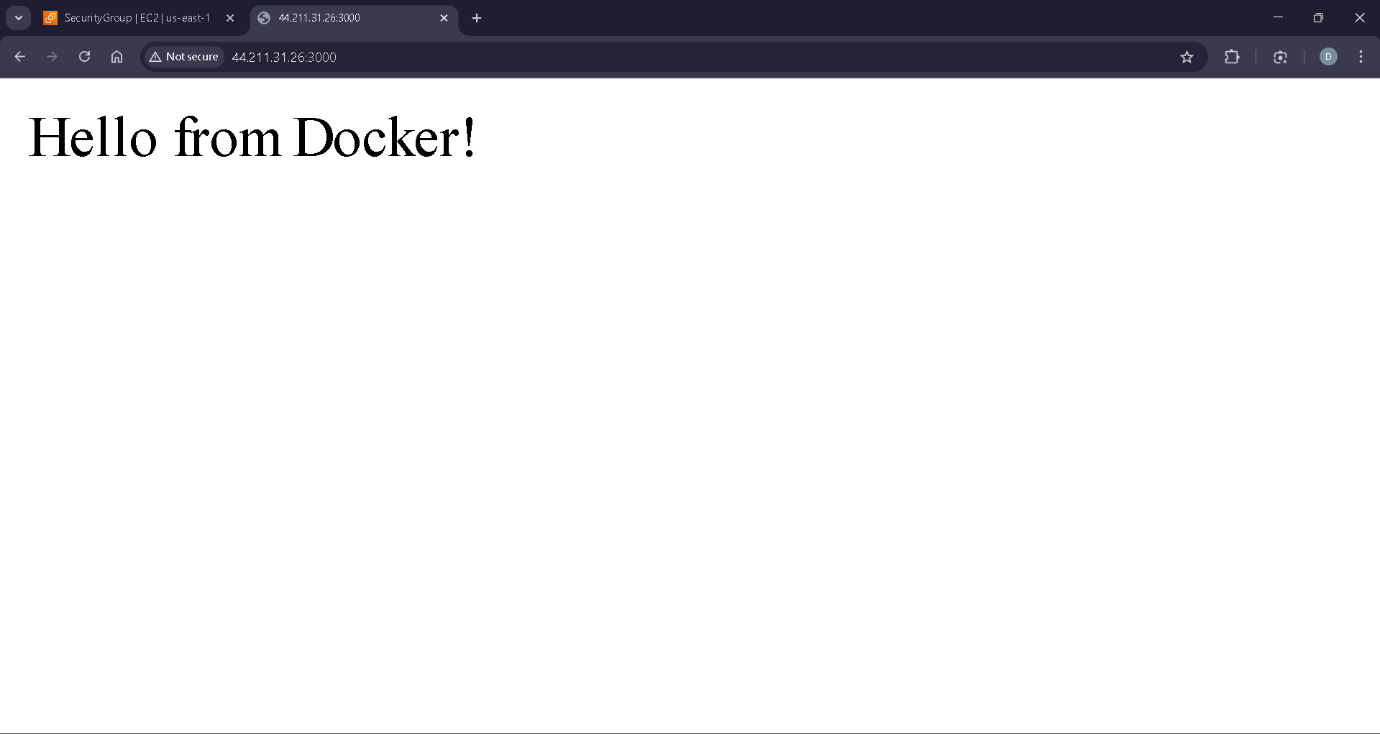
1. **Run Container (Without Compose)**

$ docker run -d -p 3000:3000 --name myapp\_container myapp



Test:

http://<ip>:3000



1. **Create docker-compose.yml**

$ docker-compose.yml

version: "3.8"

services:

web:

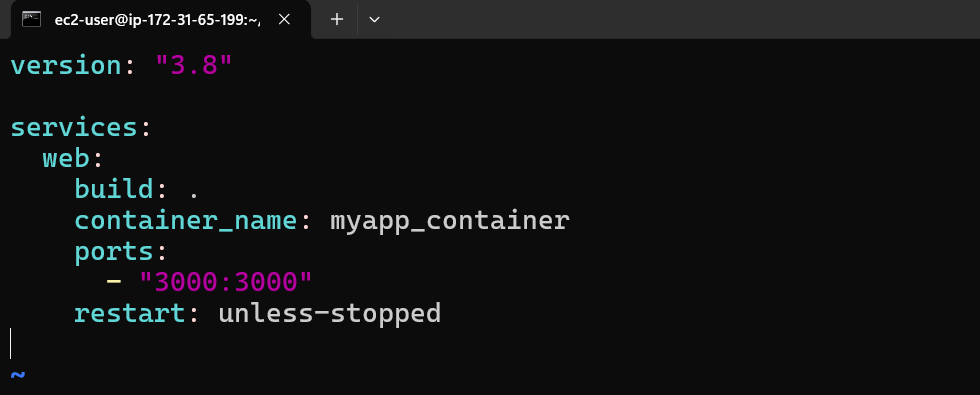
build: .

container\_name: myapp\_container

ports:

- "3000:3000"

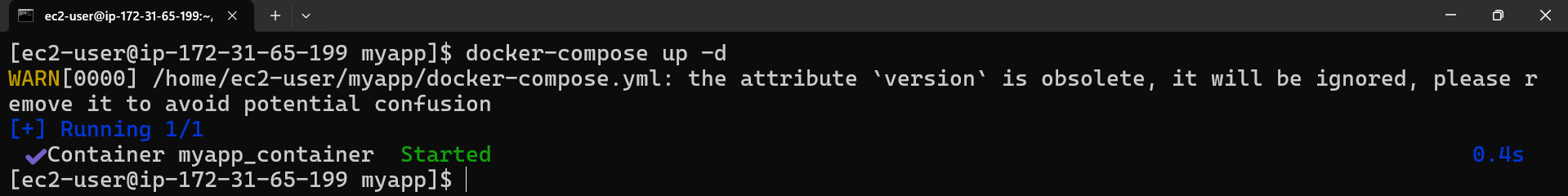
restart: unless-stopped



1. **Run Using Docker Compose**

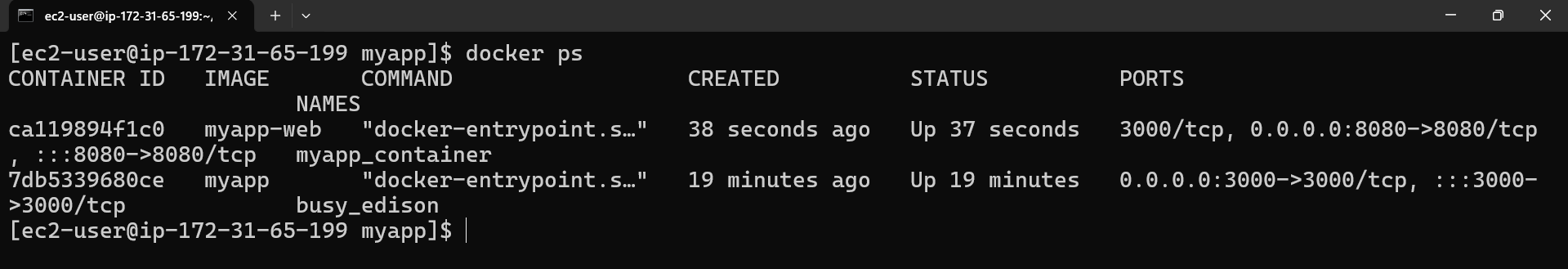
**Start:**

$ docker-compose up -d

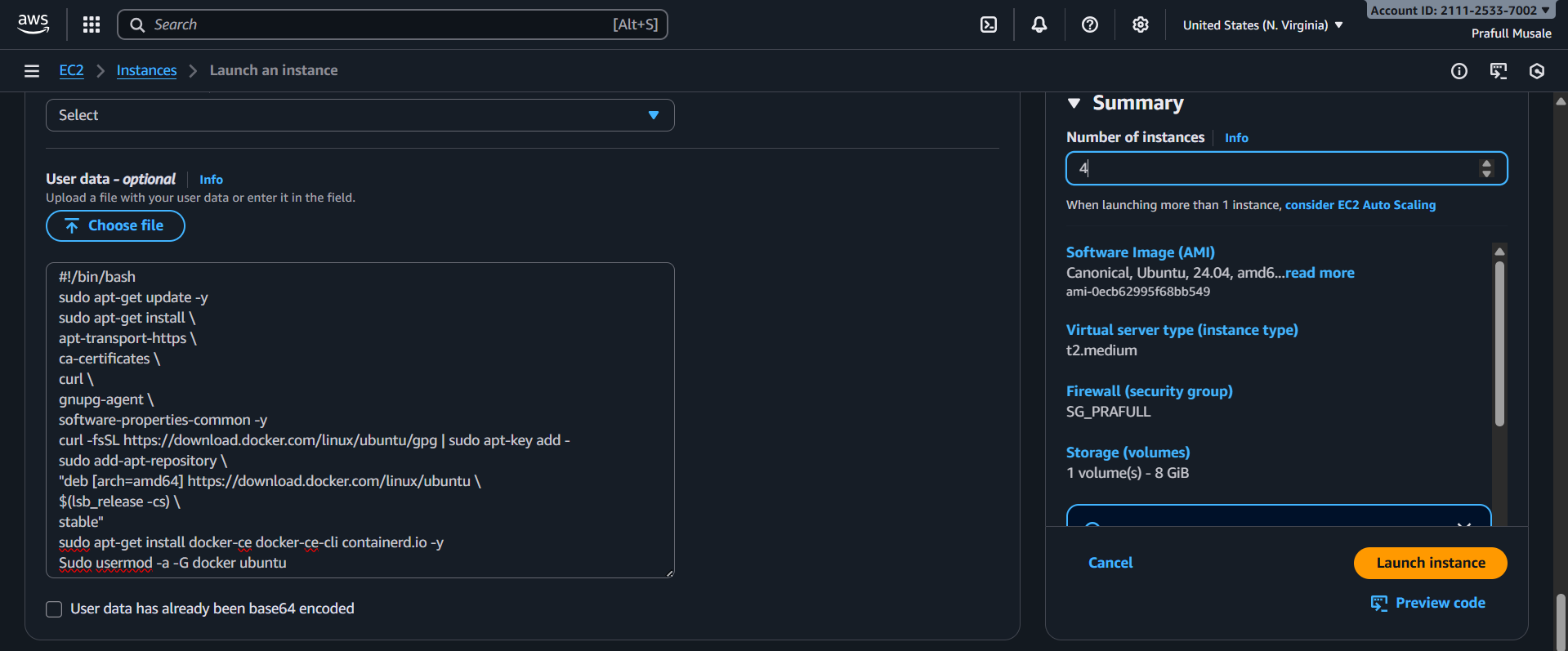


**Check running:**

$ docker ps



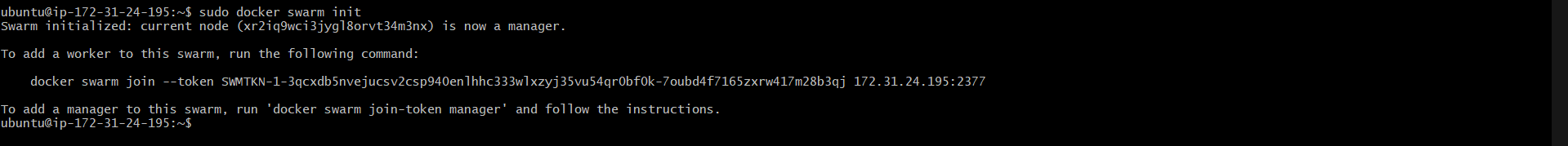
* 1. Launch 4 instance with docker installation



* 1. Connect all instance to terminal



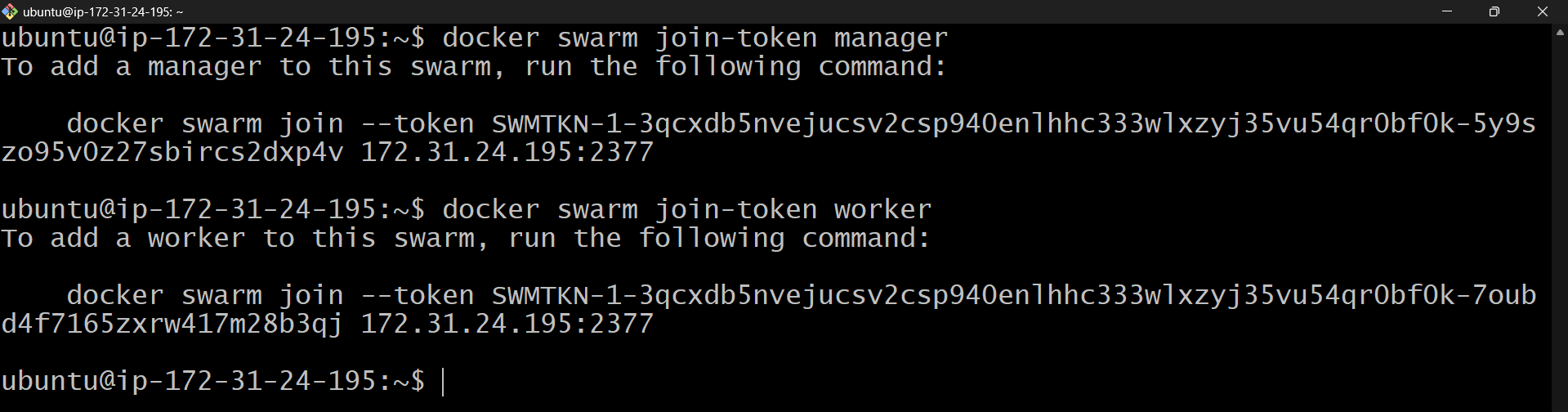
* 1. Now, Fire $ sudo docker swarm init at 1st instance for give him leadership



* 1. Now generate token for manager and worker with following commands

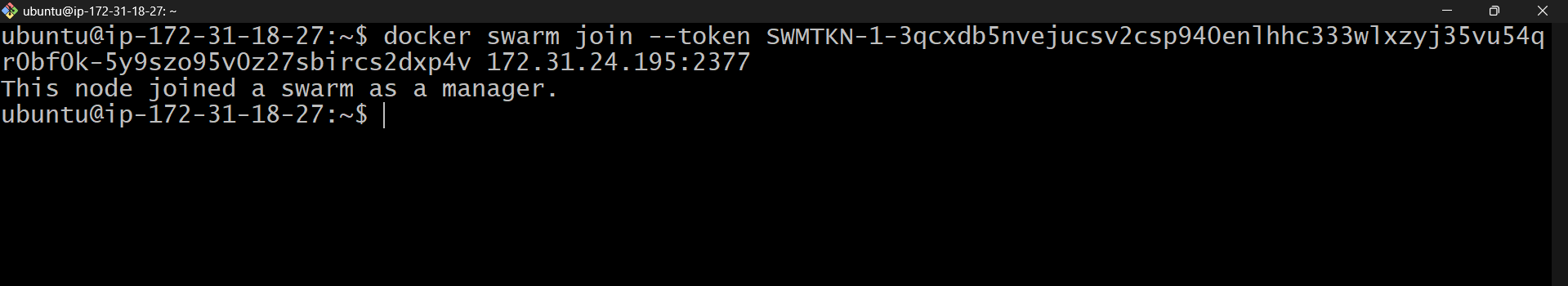
$ docker swarm join-token manager

$ docker swarm join-token worker



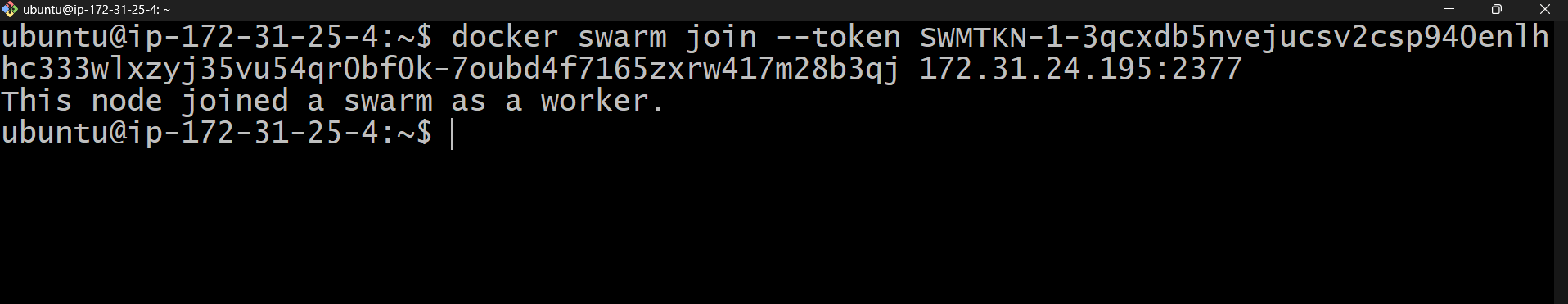
* 1. Now join 2nd instance as a manger with the help of token

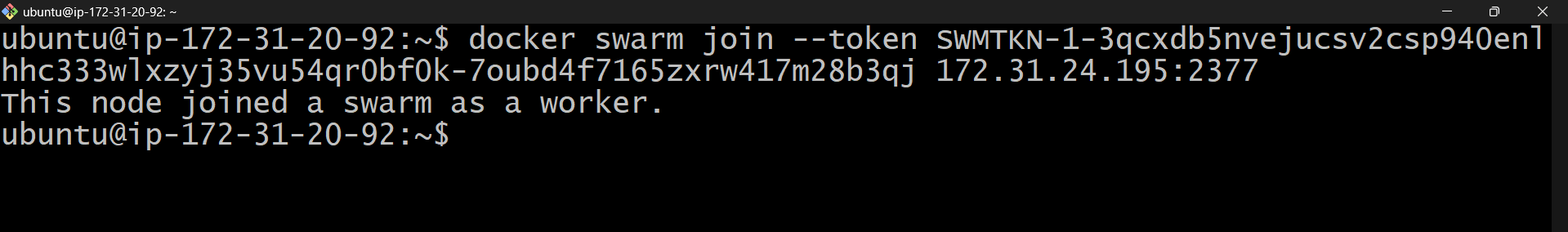
Copy manager token and paste into 2nd instance



* 1. Join 3rd and 4th instance as a worker with the help of token

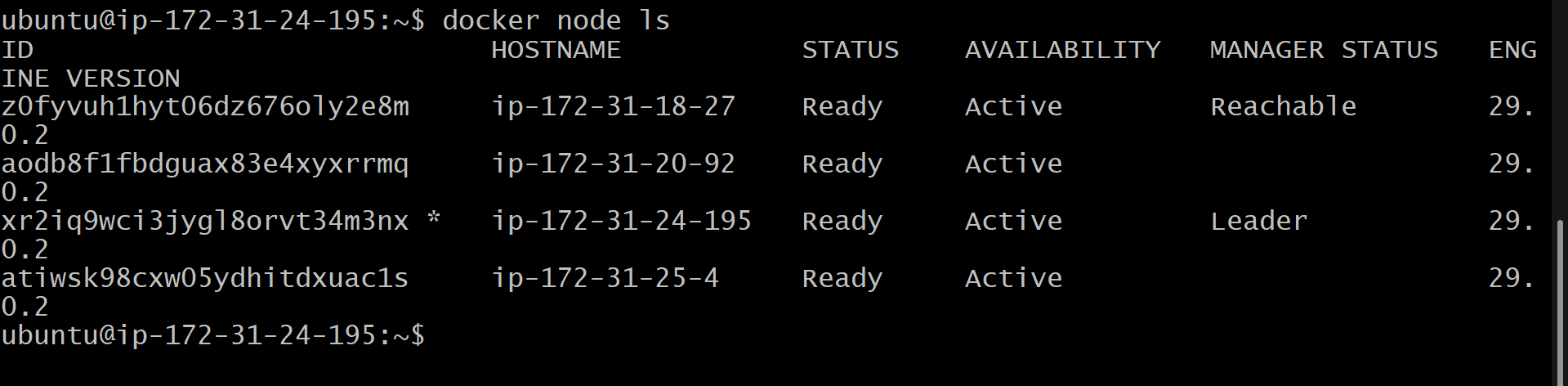
Copy worker token and paste into 3rd and 4th instance





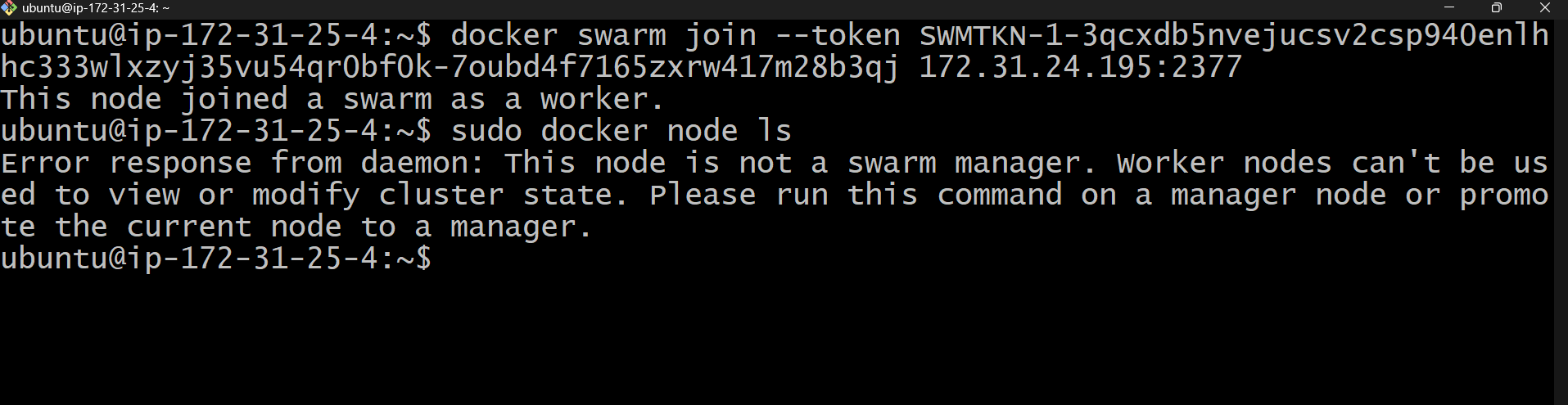
* 1. Now check which node is worker and which node is manager with the help of following command . This command fire on manager node because only manager or leader can show this .

$ docker node ls

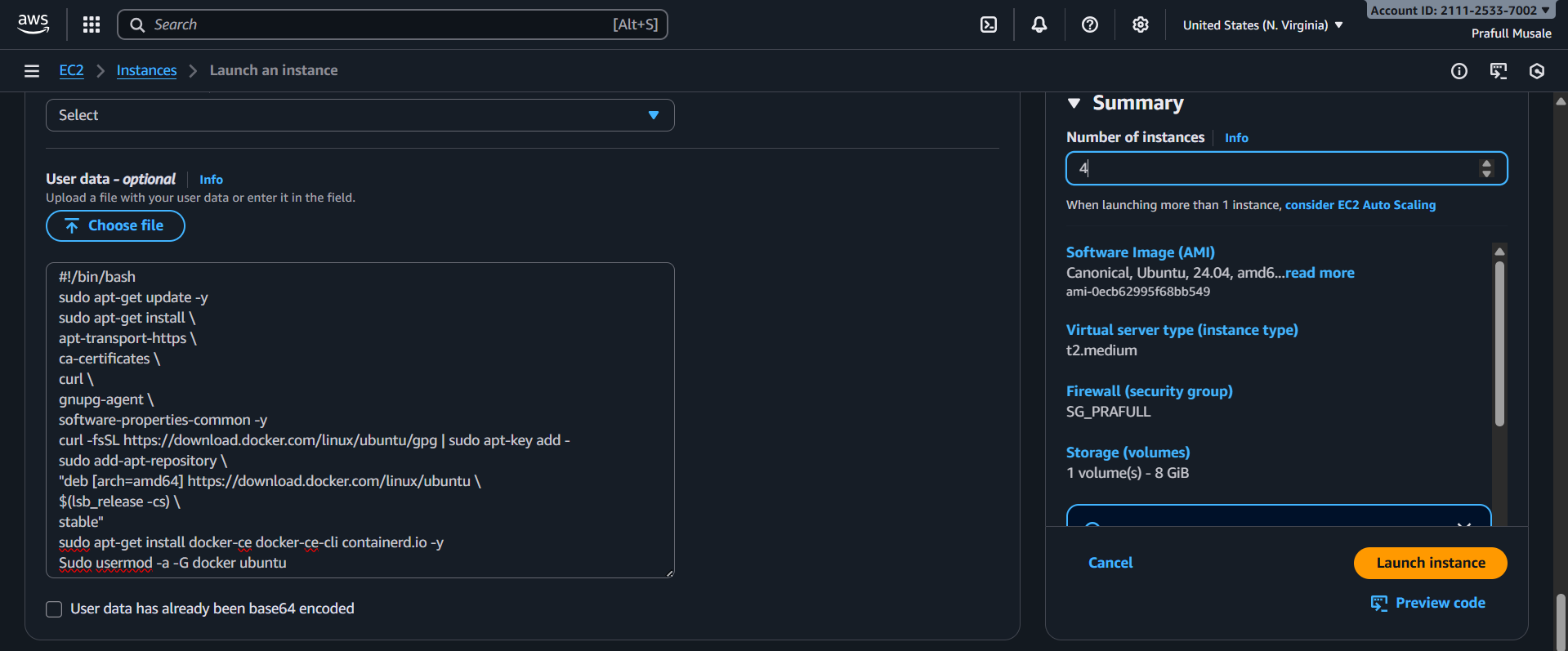


* 1. Now same command fire on worker node

Check worker has access or not



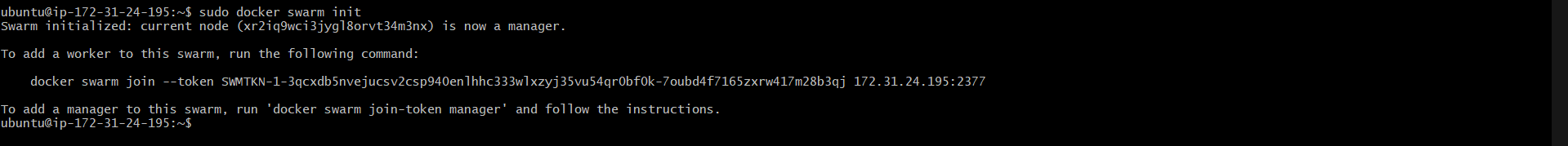
* 1. Launch 4 instance with docker installation



* 1. Connect all instance to terminal



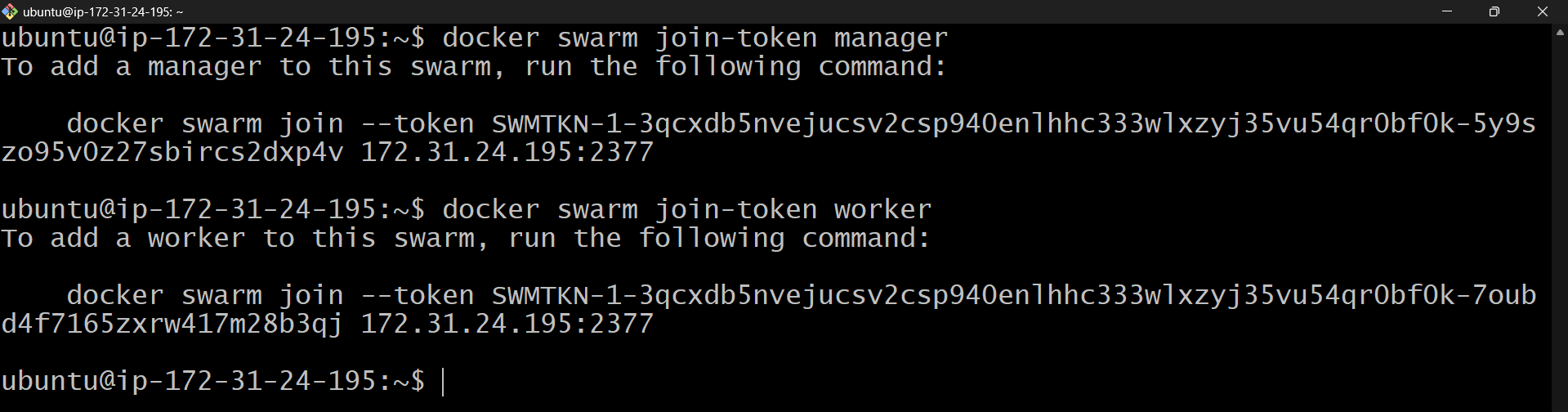
* 1. Now, Fire $ sudo docker swarm init at 1st instance for give him leadership



* 1. Now generate token for manager and worker with following commands

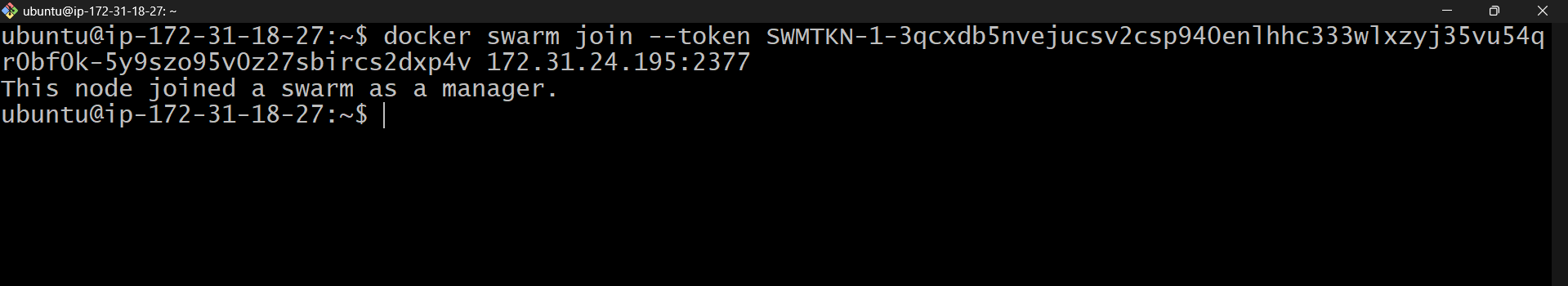
$ docker swarm join-token manager

$ docker swarm join-token worker



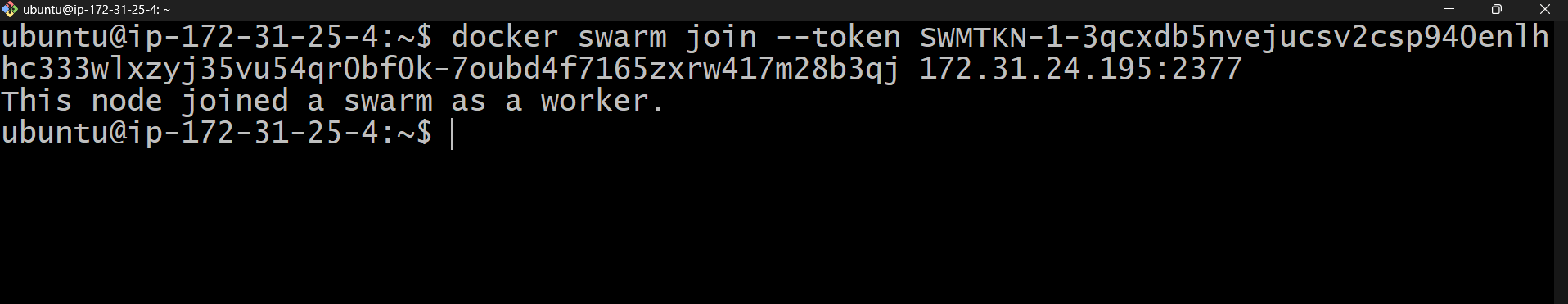
* 1. Now join 2nd instance as a manger with the help of token

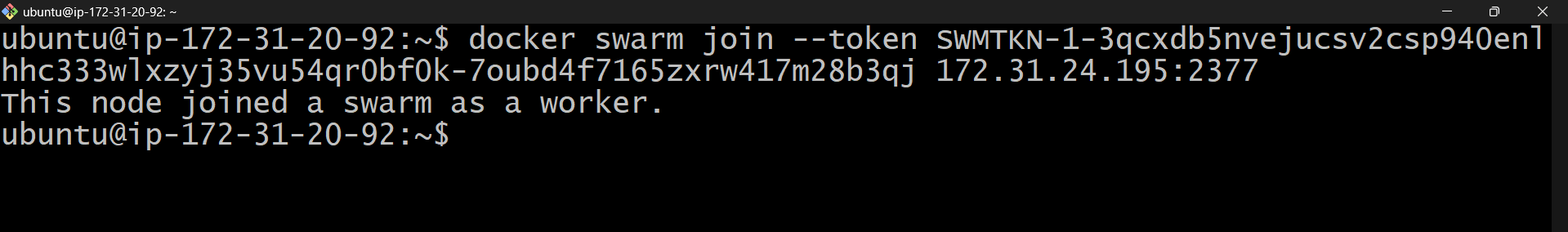
Copy manager token and paste into 2nd instance



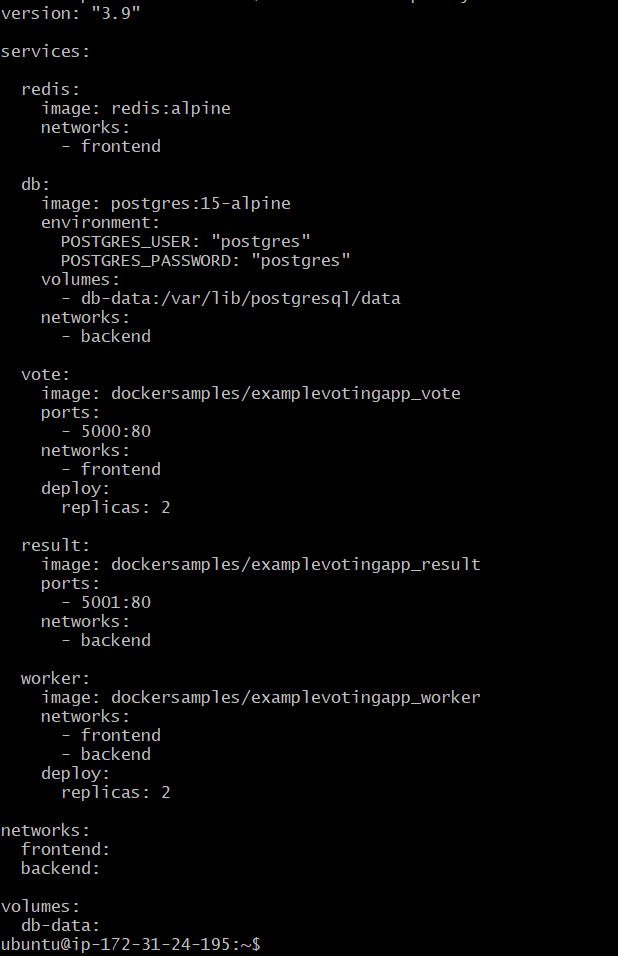
* 1. Join 3rd and 4th instance as a worker with the help of token

Copy worker token and paste into 3rd and 4th instance



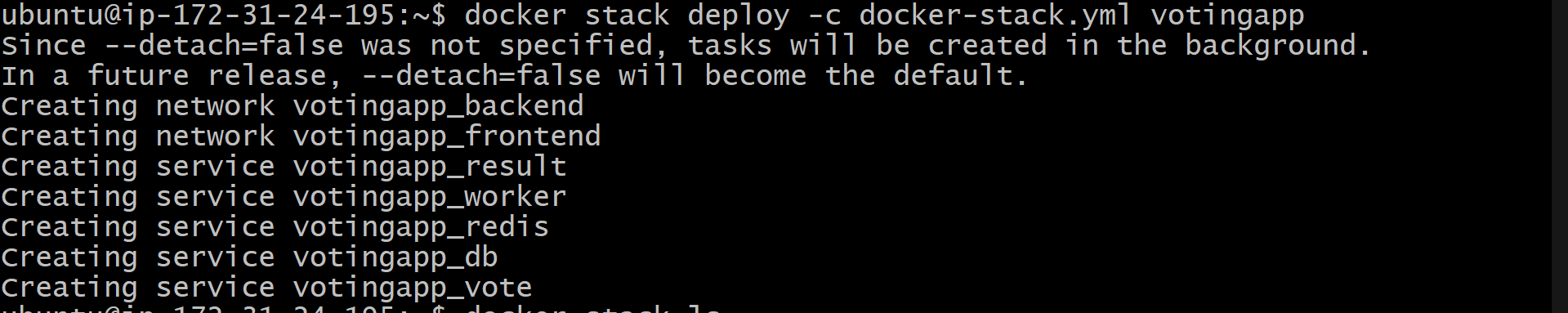


* 1. Now create docker-stack.yml to write the services



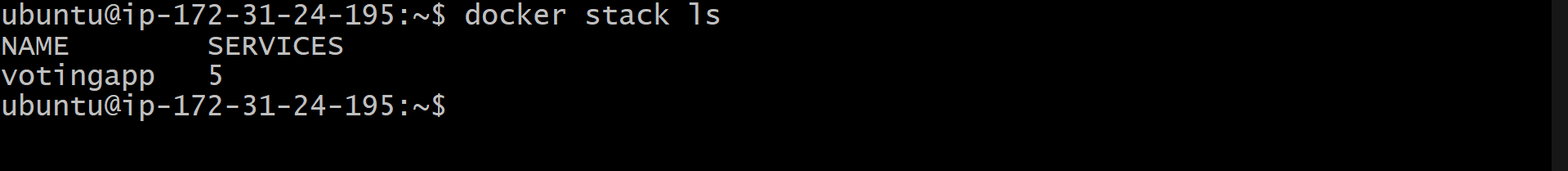
* 1. Now deploy with stack with the following command

$ docker stack deploy -c docker-stack.yml votingapp

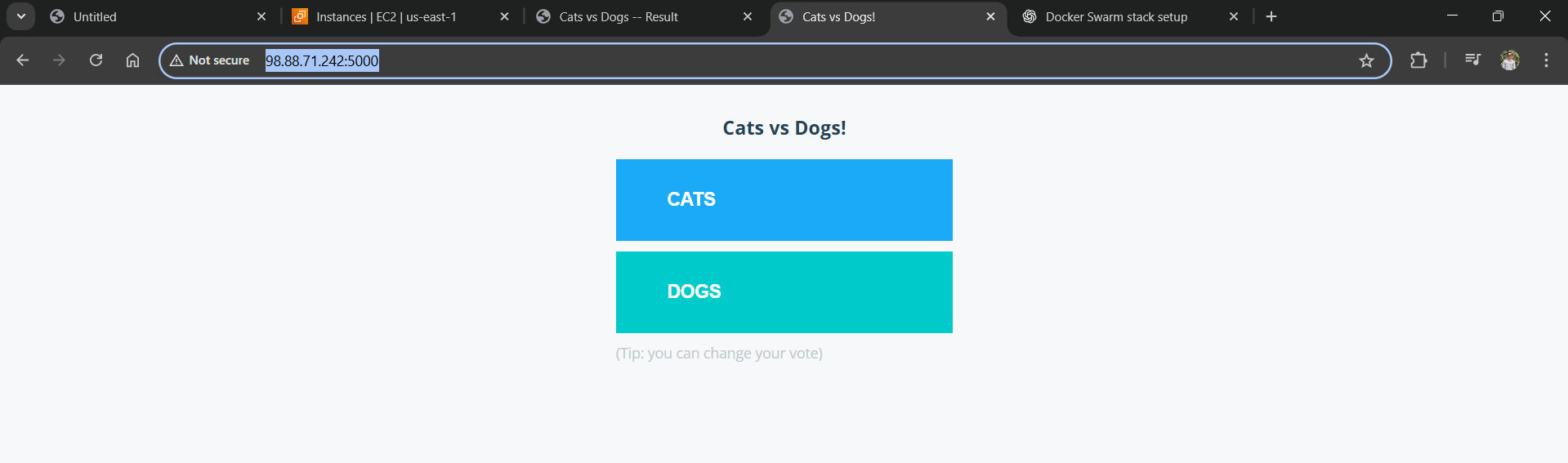


* 1. Now check stack is created or not with following command

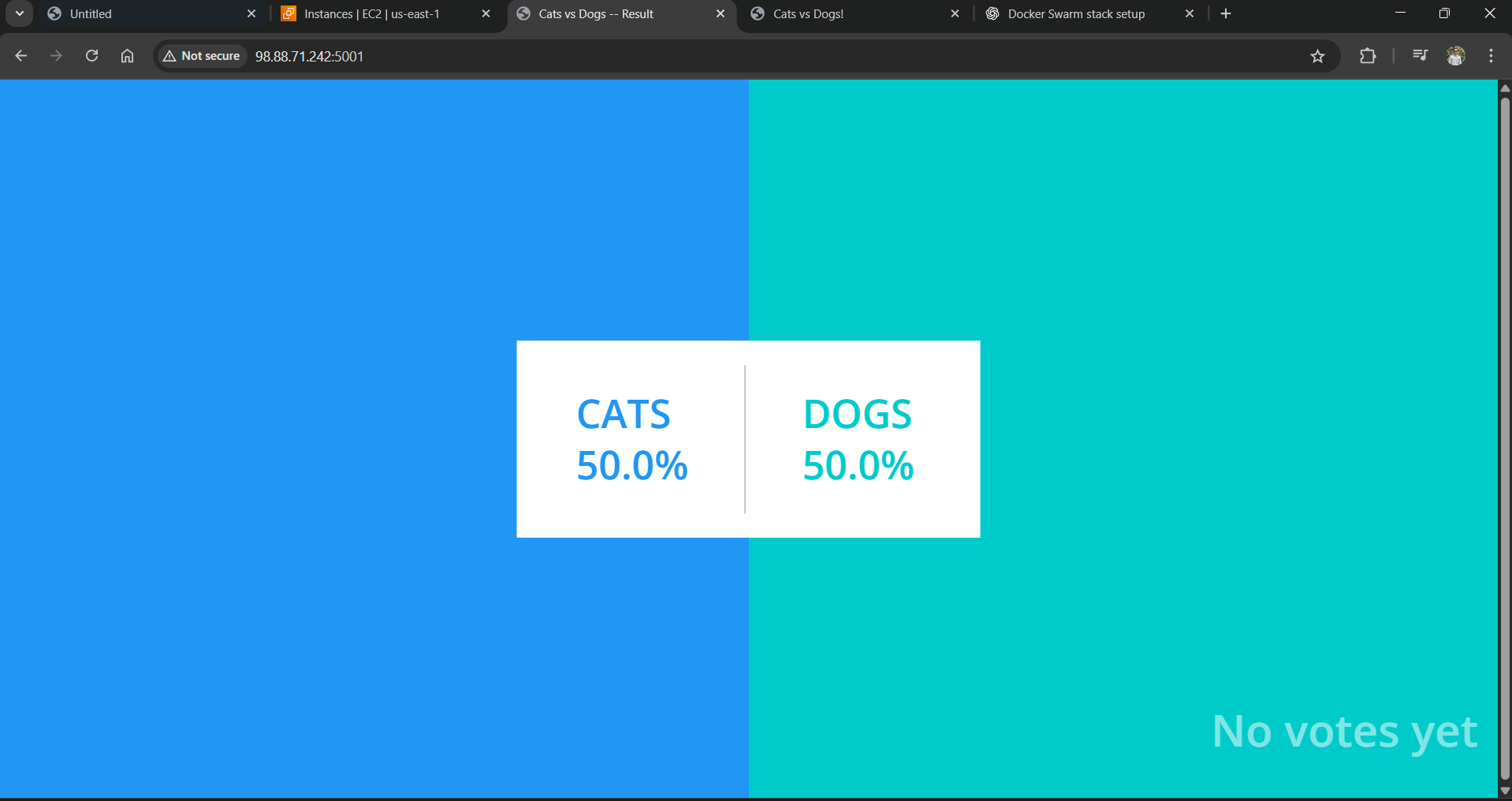
$docker stack ls



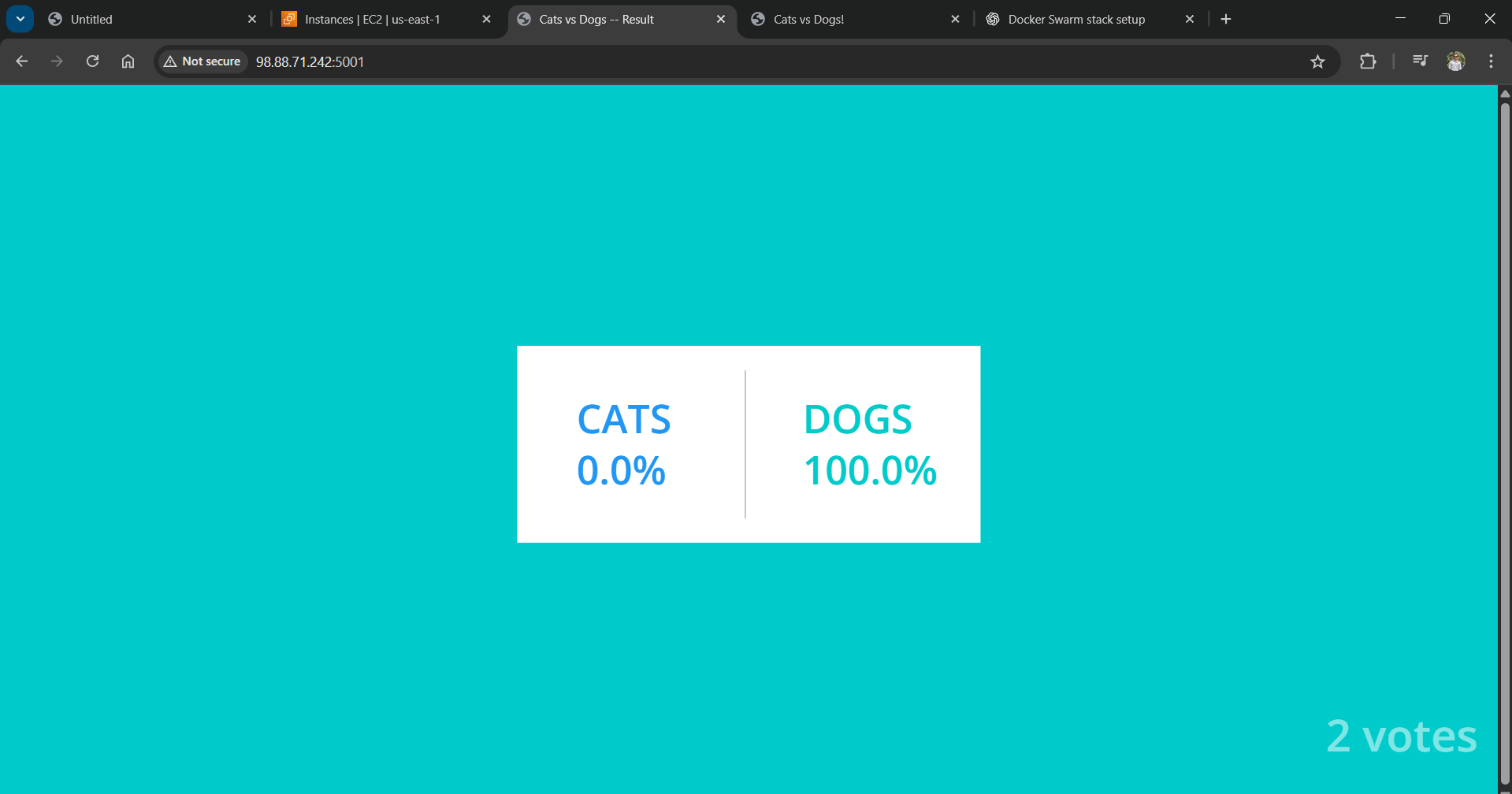
* 1. Now for voting enter on browser <Public-IP>:5000



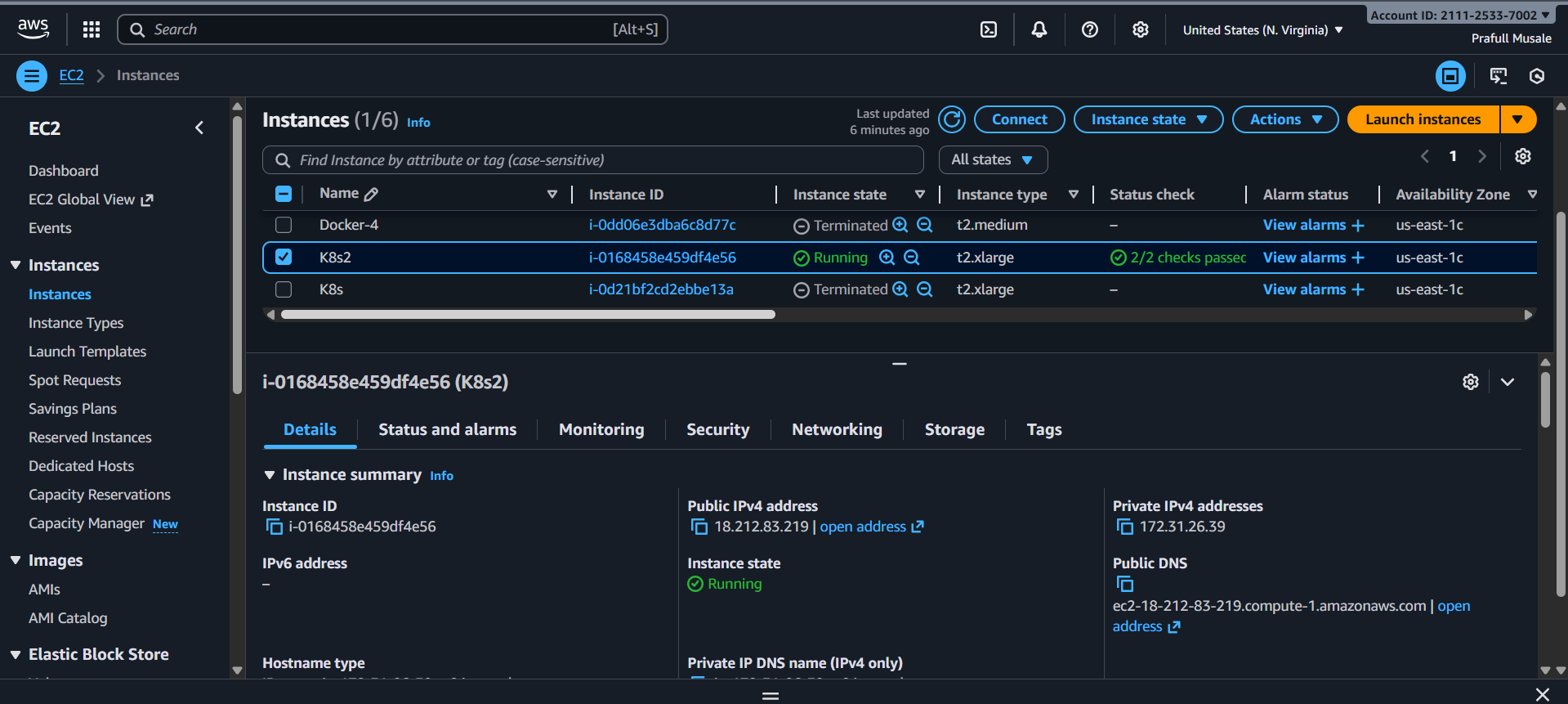
* 1. For checking Result enter on browser <Public-IP>:5001



* 1. Our Voting app is properly working



1. Launch an instance with installation of docker and Kubernetes



1. User data for installation

#!/bin/bash

sudo apt-get update -y

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common -y

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get install docker-ce docker-ce-cli containerd.io -y

sudo usermod -a -G docker ubuntu

# Initialize Kubernetes cluster

sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --ignore-preflight-errors=NumCPU || (echo 'Failed to initialize cluster' && exit 1)

# Setup kubectl

mkdir -p $HOME/.kube ; sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config || echo 'Failed to copy admin.conf' ; sudo chown $(id -u):$(id -g) $HOME/.kube/config || echo 'Failed to change ownership of config'

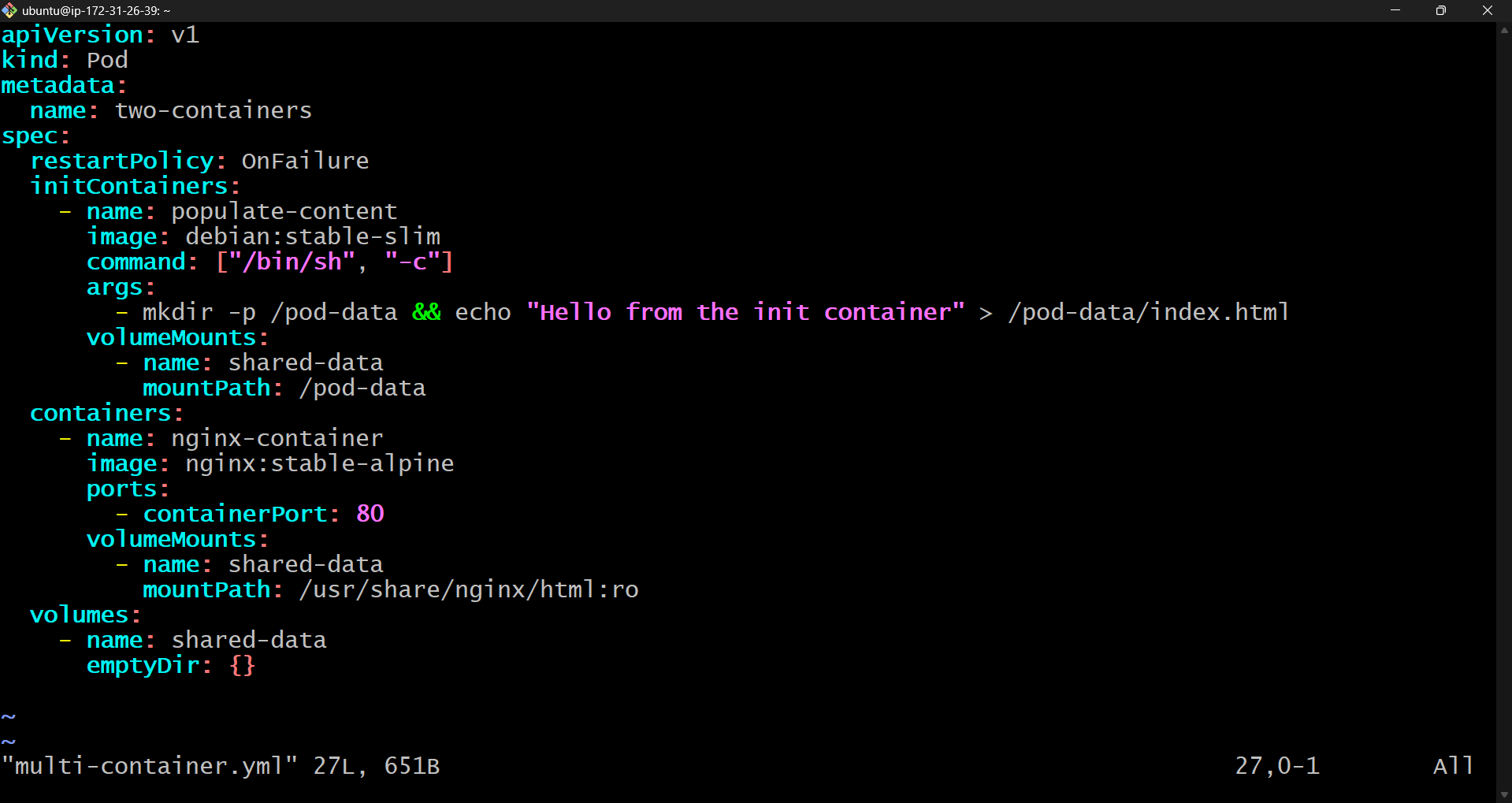
# Install Calico CNI

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.3/manifests/calico.yaml

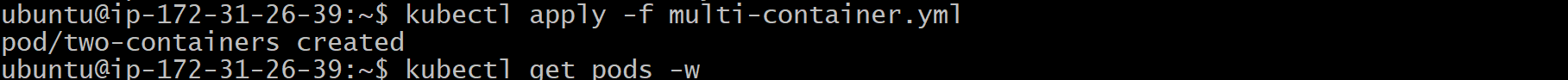
# Save join command with full permissions

sudo kubeadm token create --print-join-command

1. Now create multi-container.yml file



1. Then apply it



1. Now check the pods

