**Files:**

Dockerfile:

Dockerfile used for creating the image for the flask-app

docker-compse.yaml:

yaml file used to test deploying locally, uses both the flask-app image generated by us and mongo image from dockerhub

Files inside K8s Folder:

Files used for deploying the images in minikube

app-deployment.yaml:

YAML file for deploying the flask-app and the service

db-deployment.yaml:

YAML file used for deploying mongo db,

Files inside K8s\_EKS Folder:

Files used for deploying the images in minikube

app-deployment.yaml:

YAML file for deploying the flask-app and the service

pv.yaml:

YAML file used for configuring the Persistent Volume for db

db-deployment.yaml:

YAML file used for deploying mongo db

**Part 1:**

**Creating an Application:**  
The application was downloaded from the given source code and tested locally using ‘flask run’ and having mongodb run on port 27017 locally

**Part 2:**

**Containerizing the Application on Docker:**

Changed the db from ‘localhost’ to ‘db’ inside the app so that it can run in dockerized form.

Used “docker build -t <image\_name> .” to build the docker image

Used “docker-compose up” to run both app and db images locally and tested

Pushed the docker image to dockerhub using “docker push <image\_name>”

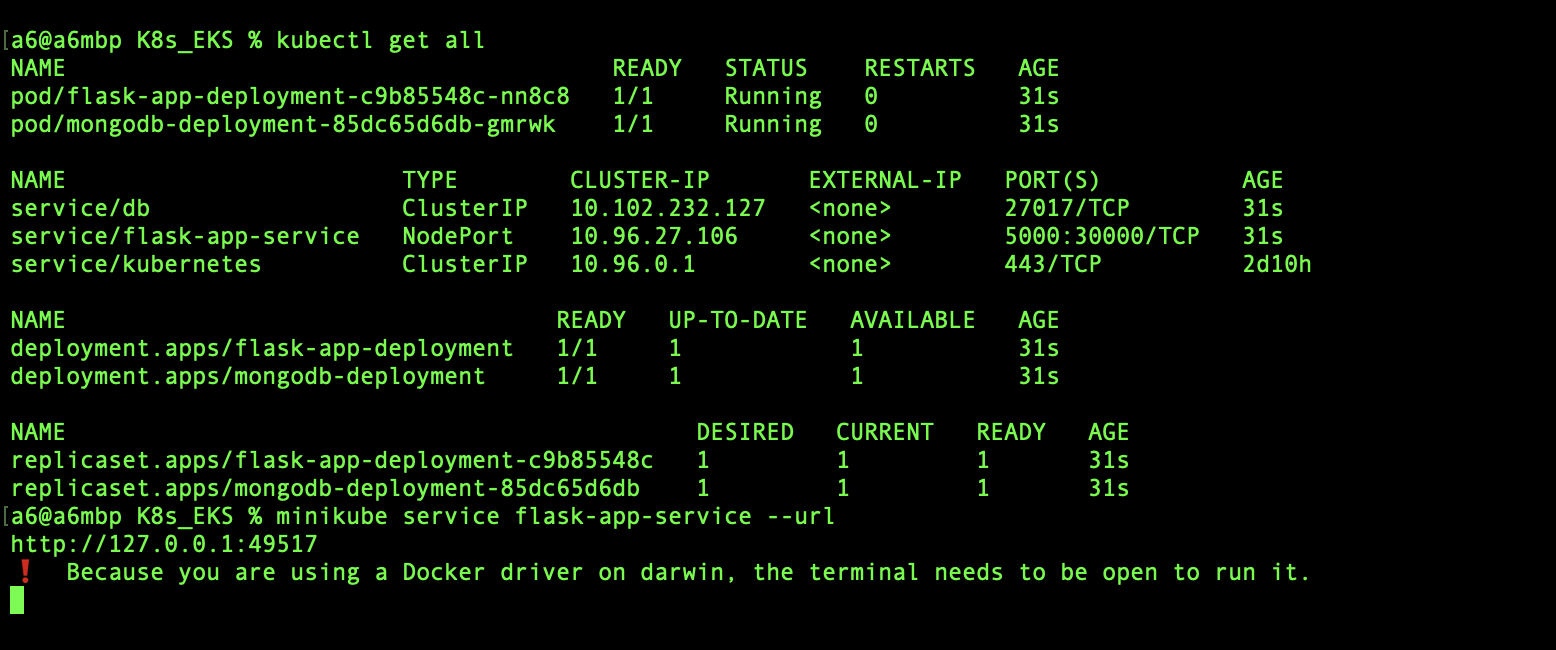
**Part 3:**

**Deploying the Application on Minikube:**

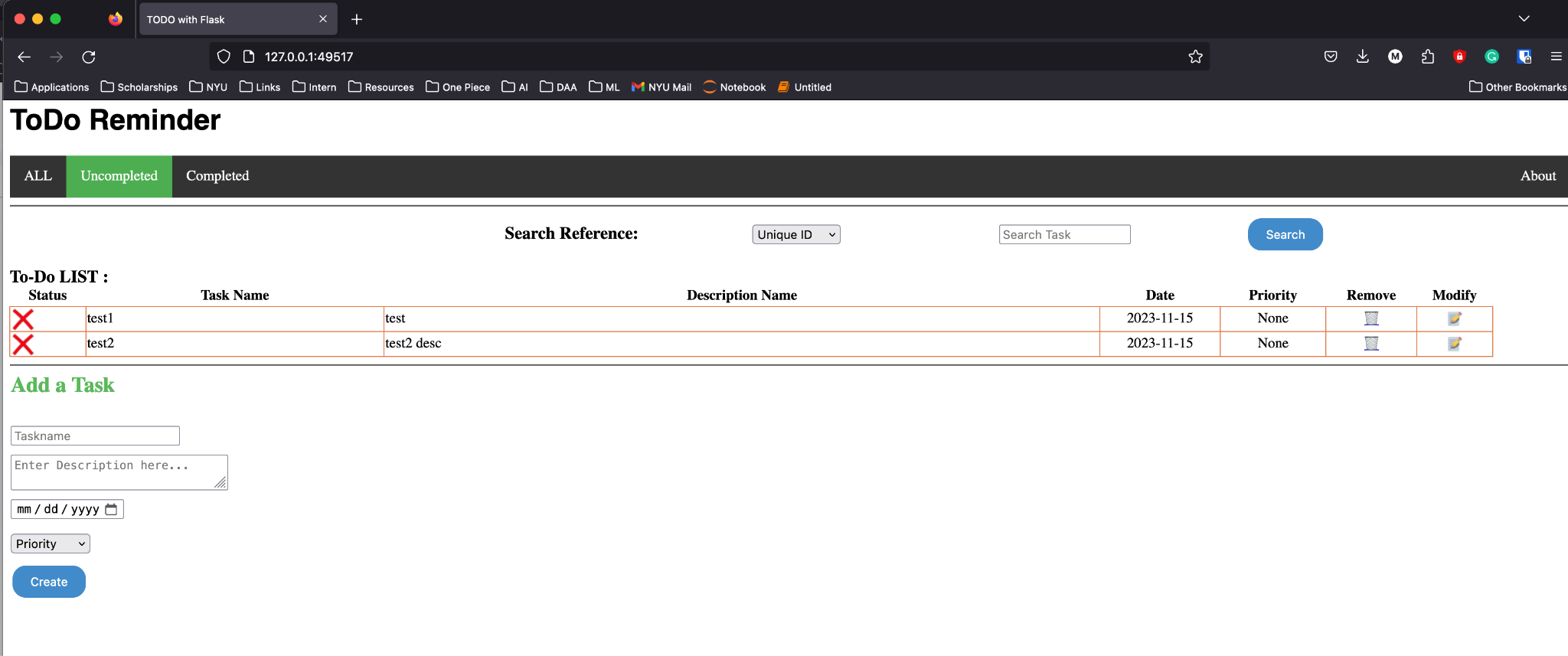
Installed minikube on the local system

Used ‘minikube start’ to start the system and used configured kubectl to use the context of minikube

Used ‘kubectl apply -f <yaml\_file\_name>’ we deployed the images on minikube.   
(Note Files used were K8s/app-deployment.yaml and K8s/db-deployment.yaml)



Using ‘minikube service flask-app-service --url’ we will get the URL which is used in the browser to get the app



**Part 4:**

**Deploying the Application on AWS EKS:**

Configured aws cli with the correct credentials and installed eksctl to use eks on aws

Created a cluster in AWS using the command

“eksctl create cluster --name app-cluster2 --version 1.28 --region us-east-1”

Also changed the nodegroup to use instances with lesser size because by default it was spawning 80 GB instances.

Configured kubectl to use the ekscluster using

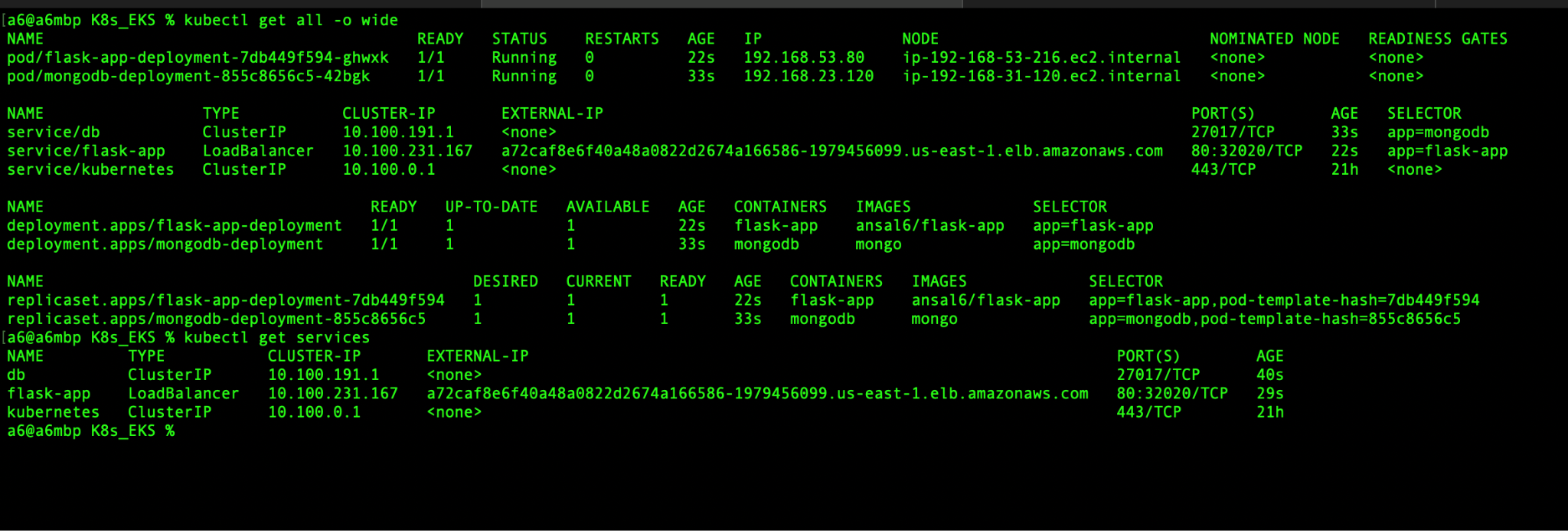
‘kubectl config set current-context <context-name>’

(we can list the contexts using kubectl config get-contexts’)

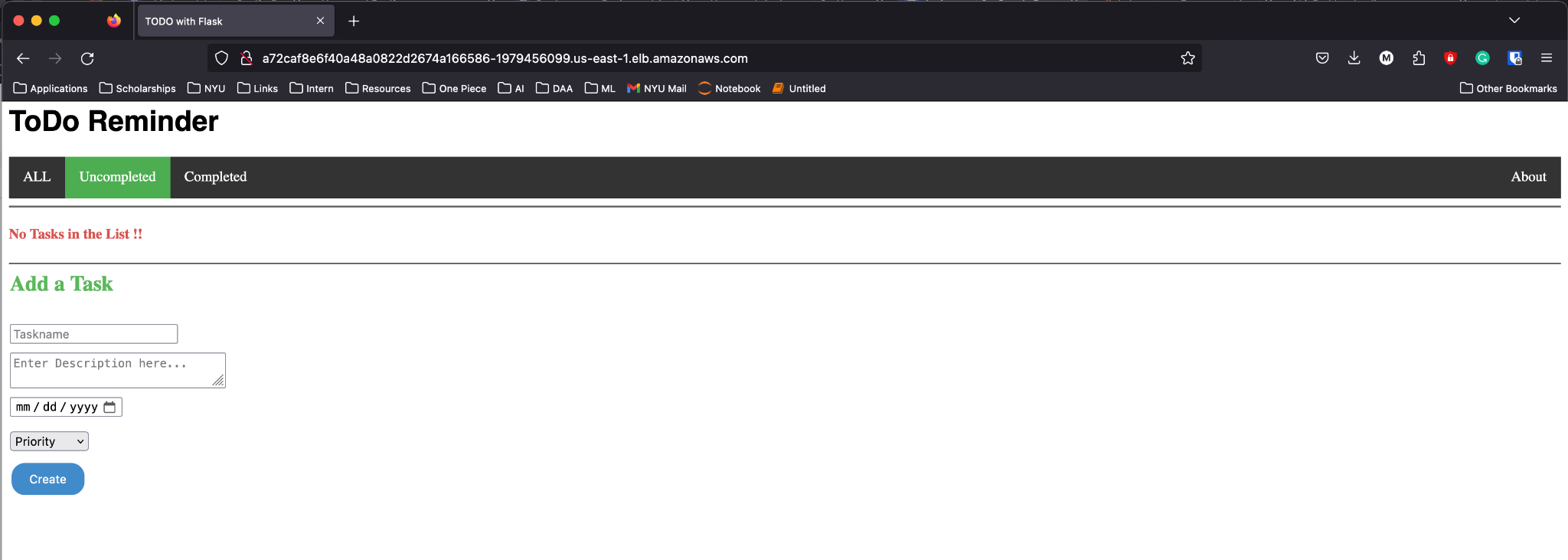
Deployed the yaml file for creating a PV and PVC(K8s\_EKS/pv.yaml)

Deployed the yaml file for creating mongo container(K8s\_EKS/db-deployment.yaml)

Deployed the yaml file for flask-app container(K8s\_EKS/app-deployment.yaml)



In the app-deployment.yaml we mentioned to use load balancer, so we wait for the external IP of the service to be available and we can access the app through the browser



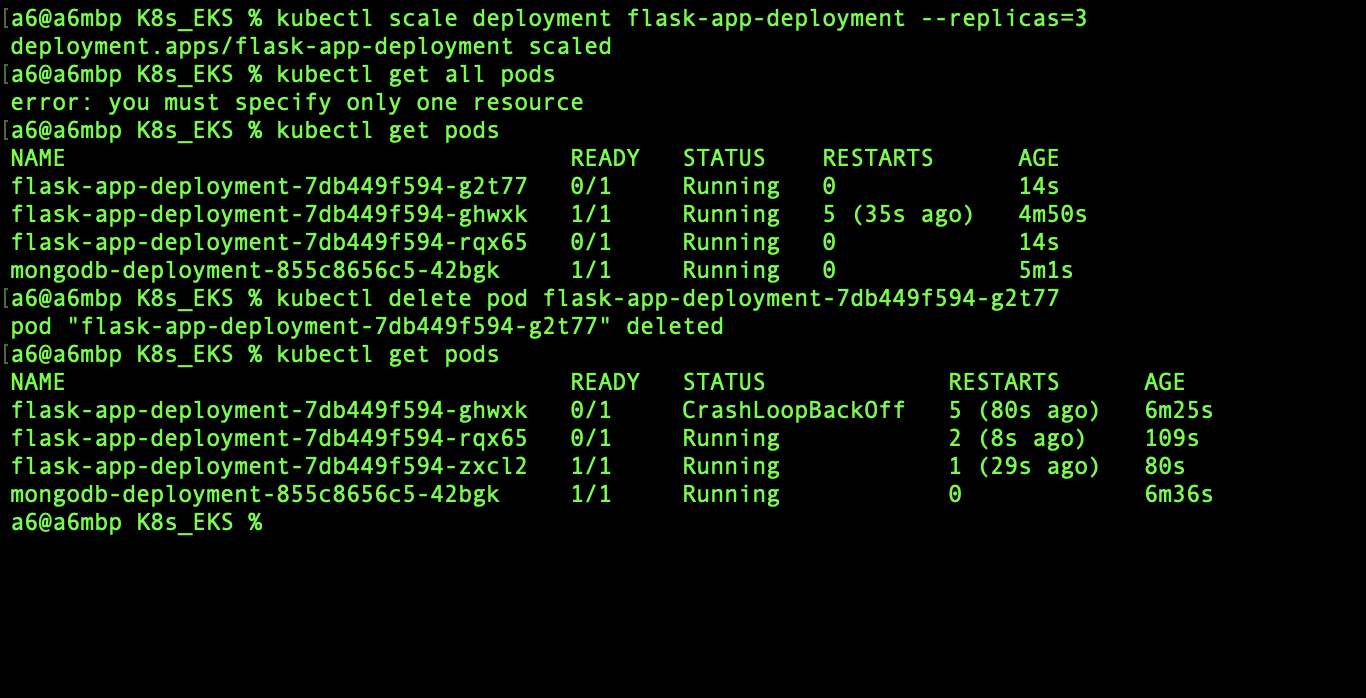
**Part 5:**

**Replication controller feature:**

During deployment, we specify the number of replicas in the yaml file, it can also be modified using the command

“kubectl scale deployment <deployment-name> --replicas=3”

Once scaled, we can see that there are three pods running the same image, if we delete one of the pod, another will be spawned and always the number of replicas are maintained



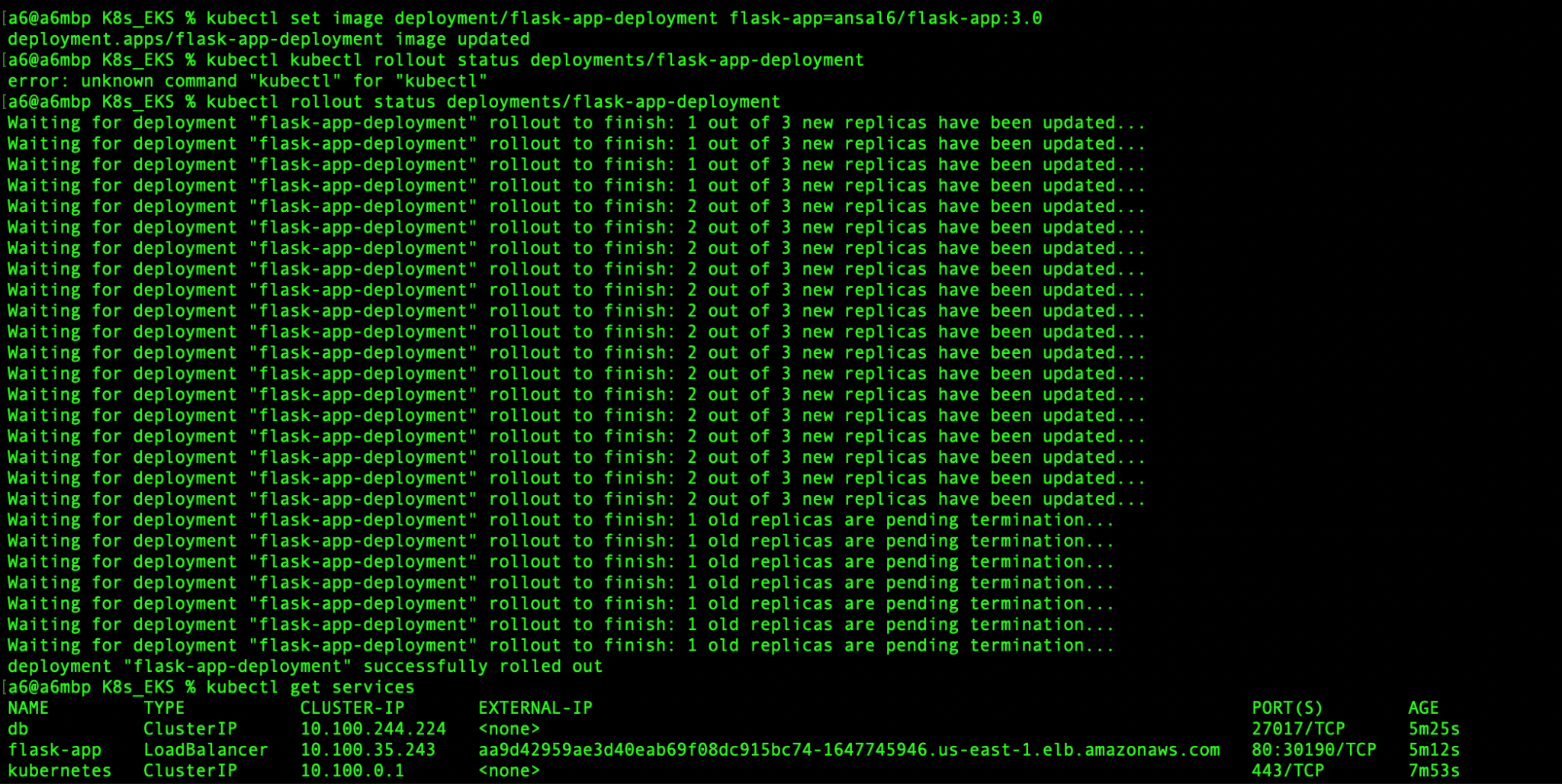
**Part 6:**

**Rolling update strategy:**

For testing the rolling update we made a change in the title so that the changes can be seen from the UI

Kubernetes uses rolling update strategy by default. So to verify that we will update the image for the flask-app deployment

We have added the rollingUpdate to use maxUnavailable as 0 and maxSurge as 1 in the yaml file(K8s\_EKS/app-deployment.yaml)



**Part 7:**

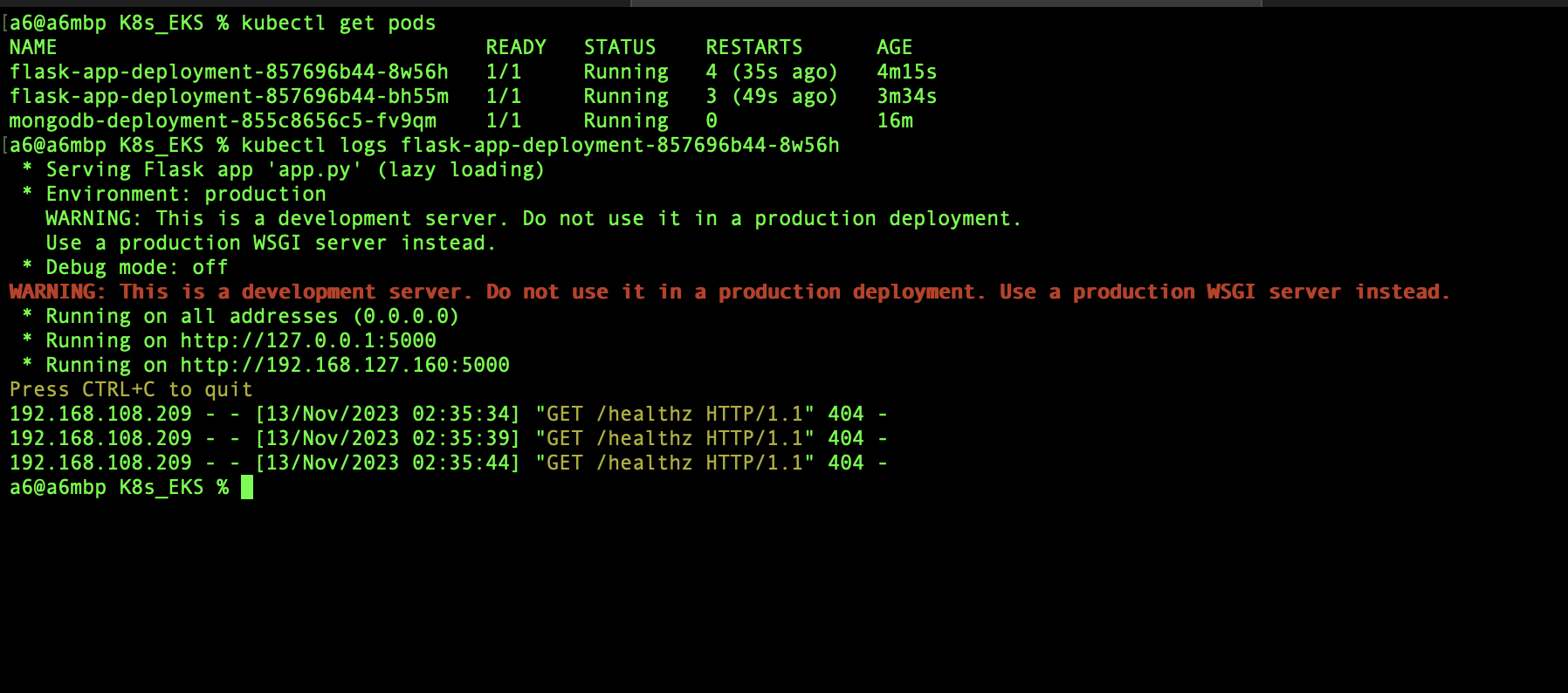
**Health monitoring:**

For health monitoring, we are using liveness and readiness probes in the yaml files (K8s\_EKS/app-deployment.yaml)

For liveness probe, we created a new endpoint in our application called /healthz which will respond ‘OK’, if that endpoint is working the app is live.

For readiness probe we are not using a specific endpoint rather checking if the instance is up.

For testing, we will deliberately remove the /healthz endpoint from the application

****

We can see that the pod is getting restarted from the response of ‘kubectl get pods’, We can also see that the endpoint is getting 404 from the logs.(we are running two replicas for testing this step)

So when our probe is failing, the pod is getting restarted. This verifies that Kubernetes is taking appropriate action when the probe fails