Tejaswini Pradip Srivastava - tps7866
Komal Bagwe - knb4003

Cloud Computing and Big Data Systems
Fall 2025 Assignment 2

Part 1: Creating the Application:

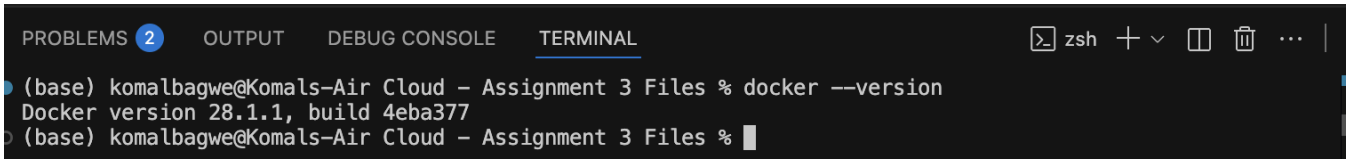
We have created the application using the source code and uploaded the repository on Github - https://github.com/komal-b/ToDoApp

Part 2: Containerizing the Application on Docker:

Step 1: Install Docker

1. Download and install Docker from the official Docker website.

2. Verify installation:
   docker --version



Step 2: Create a Dockerfile for Flask App

1. We created the following docker file and stored in this path: **Path: /Cloud - Assignment 3 Files/Dockerfile**

Step 3: Build the Docker Image

1. To build the docker image. We ran the following command in the same directory as the Dockerfile:
   docker build -t flask-todo-app:latest .

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker build -t flask-todo-app:latest .

[+] Building 0.8s (11/11) FINISHED                                                docker:desktop-linux
 => [internal] load build definition from Dockerfile                                              0.0s
 => => transferring dockerfile: 488B                                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                               0.6s
 => [auth] library/python:pull token for registry-1.docker.io                                     0.0s
 => [internal] load .dockerignore                                                                 0.0s
 => => transferring context: 2B                                                                   0.0s
 => [1/5] FROM docker.io/library/python:3.10-slim@sha256:e0c4fae70d550834a40f6c3e0326e02cfe239c2351  0.0s
 => [internal] load build context                                                                 0.0s
 => => transferring context: 7.34kB                                                               0.0s
 => CACHED [2/5] WORKDIR /app                                                                     0.0s
 => CACHED [3/5] COPY requirements.txt .                                                          0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                               0.0s
 => [5/5] COPY . .                                                                                0.0s
 => exporting to image                                                                            0.0s
 => => exporting layers                                                                           0.0s
 => => writing image sha256:2cec1518c0d86808969bbb2e91f2cd7f3a853f29e46582919b7d68d94855e2d3      0.0s
 => => naming to docker.io/library/flask-todo-app:latest                                          0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/u4guzfpovrglob49kmdq0oqc3

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Step 4: Verify the image:

1. We ran the following command to verify the docker image:
   docker images

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker images

REPOSITORY                          TAG
              IMAGE ID       CREATED              SIZE
flask-todo-app                      latest
              2cec1518c0d8   About a minute ago   161MB
cloud-assignment3files-web          latest
              47152d8cbe06   9 hours ago          161MB
```

Step 5: Test Locally Using Docker Compose

1. To test the application locally with both containers (Flask and MongoDB), We have created docker-compose.yaml file.
   Path: /Cloud - Assignment 3 Files/docker-compose.yaml
2. To run the application: **docker-compose up**

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker-compose up
 [+] Running 2/2
  ✓ Container mongodb        Created                                                               0.
  ✓ Container flask-todo-app Created                                                               0.
 Attaching to flask-todo-app, mongodb
```

Step 6: Push the Docker Image to Docker Hub

1. Create a Docker Hub account at https://hub.docker.com

2. Log in using the Docker CLI:
   docker login

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker login
Authenticating with existing credentials... [Username: komalbagwe31]

i Info → To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded
```

3. Tag your local image to match your Docker Hub repository:
   docker tag flask-todo-app:latest <dockerhub-username>/flask-todo-app:latest

   a. For us : **docker tag flask-todo-app:latest komalbagwe31/flask-todo-app**

4. Push the image to Docker Hub:
   docker push <dockerhub-username>/flask-todo-app:latest

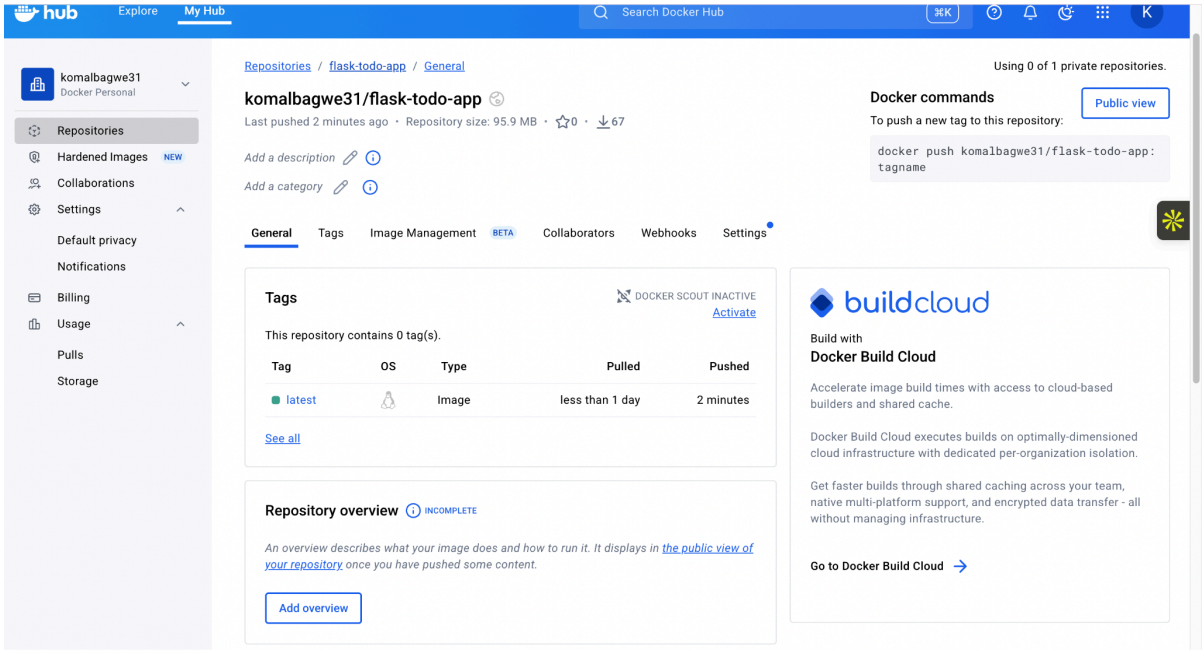
   a. For us : **docker push komalbagwe31/flask-todo-app:latest**

5. Image will be publicly available on Docker Hub for use in deployments.

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker tag flask-todo-app:latest komalbagwe31/fl
ask-todo-app

(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % docker push komalbagwe31/flask-todo-app:latest

The push refers to repository [docker.io/komalbagwe31/flask-todo-app]
e3b68d20237e: Pushed
59ba9a89551c: Layer already exists
736ee4c8d468: Layer already exists
8cc3bd02b194: Layer already exists
938c55db4ae0: Layer already exists
772dad6c1281: Layer already exists
2ace991c990b: Layer already exists
27d9a8093fcd: Layer already exists
latest: digest: sha256:9f238d5343c55b624856dae4b96d25ddf589267edf76b7e7a667be38d8ab2f5c size: 1993
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files %
```


hub
Explore
My Hub
Search Docker Hub
⌘K
K
komalbagwe31
Docker Personal
Repositories
Hardened Images
NEW
Collaborations
Settings
Default privacy
Notifications
Billing
Usage
Pulls
Storage
Repositories / flask-todo-app / General
Using 0 of 1 private repositories.
komalbagwe31/flask-todo-app
Last pushed 2 minutes ago • Repository size: 95.9 MB • 0 • 67
Add a description
Add a category
Docker commands
To push a new tag to this repository:
Public view
docker push komalbagwe31/flask-todo-app:tagname
General
Tags
Image Management
BETA
Collaborators
Webhooks
Settings
Tags
DOCKER SCOUT INACTIVE
Activate
This repository contains 0 tag(s).
Tag
OS
Type
Pulled
Pushed
latest
Image
less than 1 day
2 minutes
See all
Repository overview
INCOMPLETE
An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.
Add overview
buildcloud
Build with
Docker Build Cloud
Accelerate image build times with access to cloud-based builders and shared cache.
Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.
Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.
Go to Docker Build Cloud

Part 3: Deploying the Application on Minikube:

Step 1: Install Minikube

1. Download and install Minikube
2. Verify installation:
   minikube version

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % minikube version
minikube version: v1.36.0
commit: f8f52f5de11fc6ad8244afac475e1d0f96841df1
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files %
```

Step 2: Start Minikube

1. Start a local Kubernetes cluster:
   minikube start

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % minikube start

😄  minikube v1.36.0 on Darwin 14.1.2 (arm64)
🎉  minikube 1.37.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.37.0
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

✨  Automatically selected the docker driver
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.47 ...
💾  Downloading Kubernetes v1.33.1 preload ...
    > preloaded-images-k8s-v18-v1...:  327.15 MiB / 327.15 MiB  100.00% 11.63 M
    > gcr.io/k8s-minikube/kicbase...:  463.69 MiB / 463.69 MiB  100.00% 11.50 M
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

2. Verify the cluster is running:
   kubectl get nodes

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   5m8s   v1.33.1
```

Step 3a: MongoDB Deployment

1. A file **mongo-deployment.yaml** is created
   Path: **/Cloud - Assignment 3 Files/mongo-deployment.yaml**

2. Apply the MongoDB deployment:
   kubectl apply -f mongo-deployment.yaml

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl apply -f mongo-deployment.yaml

deployment.apps/mongo-deployment created
service/mongo created
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files %
```

Step 3b: Flask Deployment

1. A file **deployment.yaml** is created.
   Path: **/Cloud - Assignment 3 Files/deployment.yaml**

2. Apply the Flask deployment:
   kubectl apply -f deployment.yaml

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl apply -f deployment.yaml

deployment.apps/flask-todo-deployment created
service/flask-todo-service created
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files %
```

Step 4: Expose the Deployment

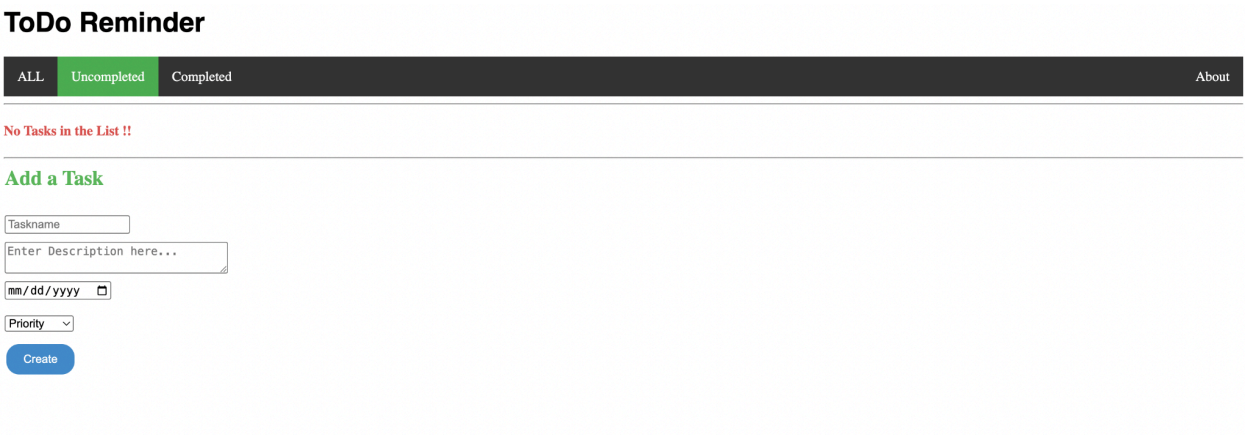
1. Minikube provides a URL to access the service:
   minikube service flask-todo-service --url

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % minikube service flask-todo-service  --url
http://127.0.0.1:54026
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Pods:

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl get pods
NAME                                 READY   STATUS              RESTARTS   AGE
flask-todo-deployment-8684d4f8d6-5h8gb   0/1     ContainerCreating   0          102s
flask-todo-deployment-8684d4f8d6-tx54b   0/1     ContainerCreating   0          102s
mongo-deployment-84c58f889-fqwmj         1/1     Running             0          3m35s
```

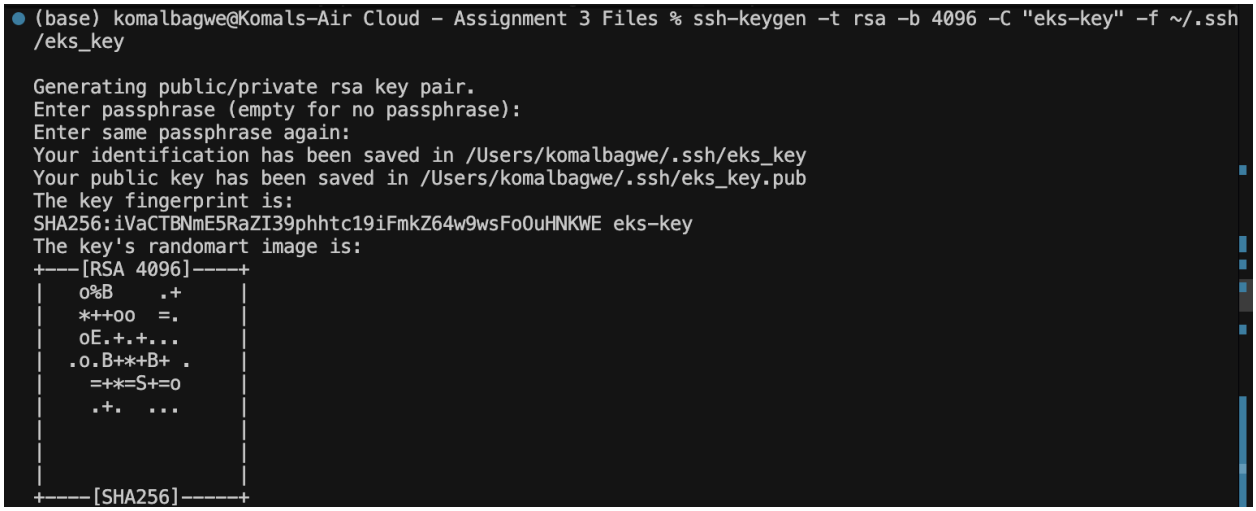
Output:



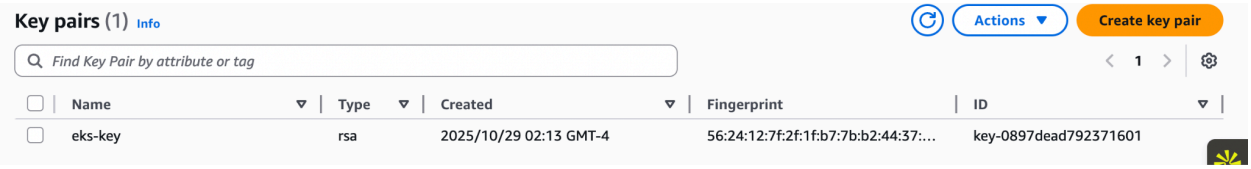
Part 4: Deploying the Application on AWS EKS:

Step 1: Create an SSH Keys if not present

1. Check if you have any SSH keys:
   ls ~/.ssh/
2. If not create one
   ssh-keygen -t rsa -b 4096 -C "eks-key" -f ~/.ssh/eks_key



3. Import the key in EC2
   a. Go to the **AWS Management Console → EC2 → Key Pairs**
   b. Click **Import key pair**
   c. We need to add name for key. I added eks-key as name



Step 2: Create an EKS Cluster

Command:

```
eksctl create cluster \
--name flask-eks-cluster \
--region us-east-1 \
--nodes 2 \
--node-type t3.medium \
--with-oidc \
--ssh-access \
--ssh-public-key eks-key \
--managed
```

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % eksctl create cluster \
--name flask-eks-cluster \
--region us-east-1 \
--nodes 2 \
--node-type t3.medium \
--with-oidc \
--ssh-access \
--ssh-public-key eks-key \
--managed

2025-10-29 02:15:21 [ℹ]  eksctl version 0.215.0
2025-10-29 02:15:21 [ℹ]  using region us-east-1
2025-10-29 02:15:22 [ℹ]  setting availability zones to [us-east-1f us-east-1d]
2025-10-29 02:15:22 [ℹ]  subnets for us-east-1f - public:192.168.0.0/19 private:192.168.64.0/19
2025-10-29 02:15:22 [ℹ]  subnets for us-east-1d - public:192.168.32.0/19 private:192.168.96.0/19
2025-10-29 02:15:22 [ℹ]  nodegroup "ng-a635343a" will use "" [AmazonLinux2023/1.32]
2025-10-29 02:15:22 [ℹ]  using EC2 key pair "eks-key"
2025-10-29 02:15:22 [!]  Auto Mode will be enabled by default in an upcoming release of eksctl. This means
 managed node groups and managed networking add-ons will no longer be created by default. To maintain curr
ent behavior, explicitly set 'autoModeConfig.enabled: false' in your cluster configuration. Learn more: ht
tps://eksctl.io/usage/auto-mode/
2025-10-29 02:15:22 [ℹ]  using Kubernetes version 1.32
2025-10-29 02:15:22 [ℹ]  creating EKS cluster "flask-eks-cluster" in "us-east-1" region with managed nodes
2025-10-29 02:15:22 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial m
```

Explanation:

1. **--name**: Cluster name
2. **--region**: AWS region
3. **--nodes**: Number of worker nodes

4. **--node-type**: EC2 instance type for nodes
5. **--managed**: Use managed node groups

This creates both the control plane and worker nodes.

Step 3: Create a deployment file

The **deployment.yaml** file we created had the service type set to **NodePort**, but in AWS EKS, we can't use **NodePort**. For AWS EKS, we need to use the service type **LoadBalancer** instead. So created new deployment file named **eks-deployment.yaml**

Apply the eks-deployment:

kubectl apply -f eks-deployment.yaml

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl apply -f eks-deployment.yaml

deployment.apps/flask-app-deployment unchanged
service/flask-app-service created
```

Apply the mongo-deployment.yaml

kubectl apply -f mongo-deployment.yaml

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl apply -f mongo-deployment.yaml
deployment.apps/mongo-deployment created
service/mongo created
```

Step 5: Test the Application

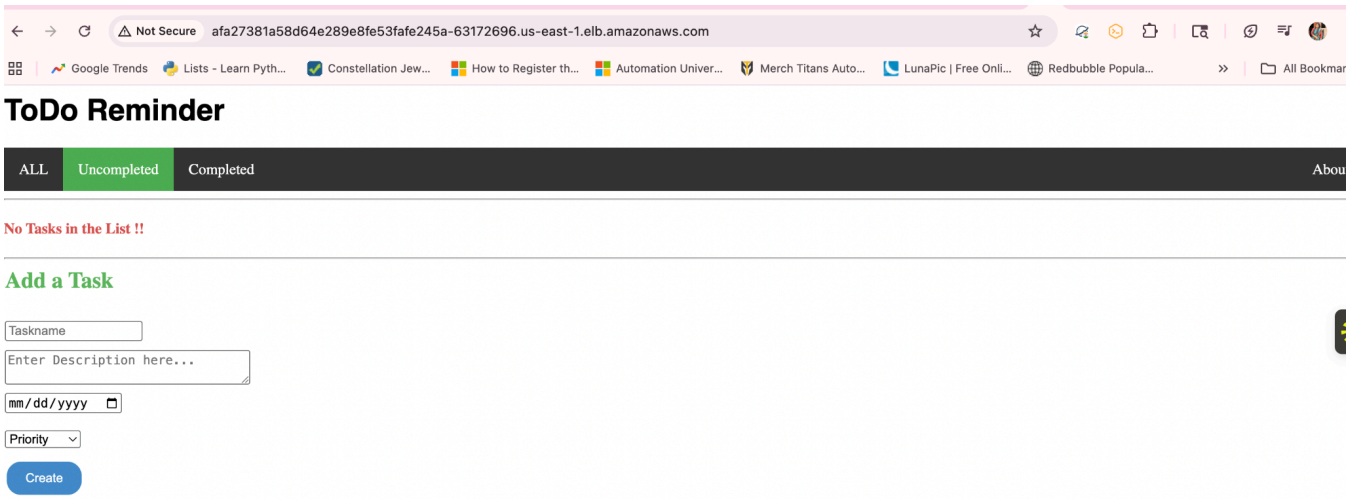
Get the external URL:

kubectl get service flask-app-service

```
(base) komalbagwe@Komals-Air Cloud - Assignment 3 Files % kubectl get service flask-app-service

NAME               TYPE           CLUSTER-IP     EXTERNAL-IP
                PORT(S)        AGE
flask-app-service   LoadBalancer   10.100.189.6   afa27381a58d64e289e8fe53fafe245a-63172696.us-east-1.
amazonaws.com   80:31001/TCP   11m
```

Output:


Not Secure
afa27381a58d64e289e8fe53fafe245a-63172696.us-east-1.elb.amazonaws.com
Google Trends
Lists - Learn Pyth...
Constellation Jew...
How to Register th...
Automation Univer...
Merch Titans Auto...
LunaPic | Free Onli...
Redbubble Popula...
All Bookmar
ToDo Reminder
ALL
Uncompleted
Completed
Abou
No Tasks in the List !!
Add a Task
Taskname
Enter Description here...
mm/dd/yyyy
Priority
Create

Part 5: Deployments and ReplicaSets:

1. Define the desired number of replicas in your deployment.yaml file under **spec.replicas.**
2. We have done this here: Line 6: **replicas: 2**

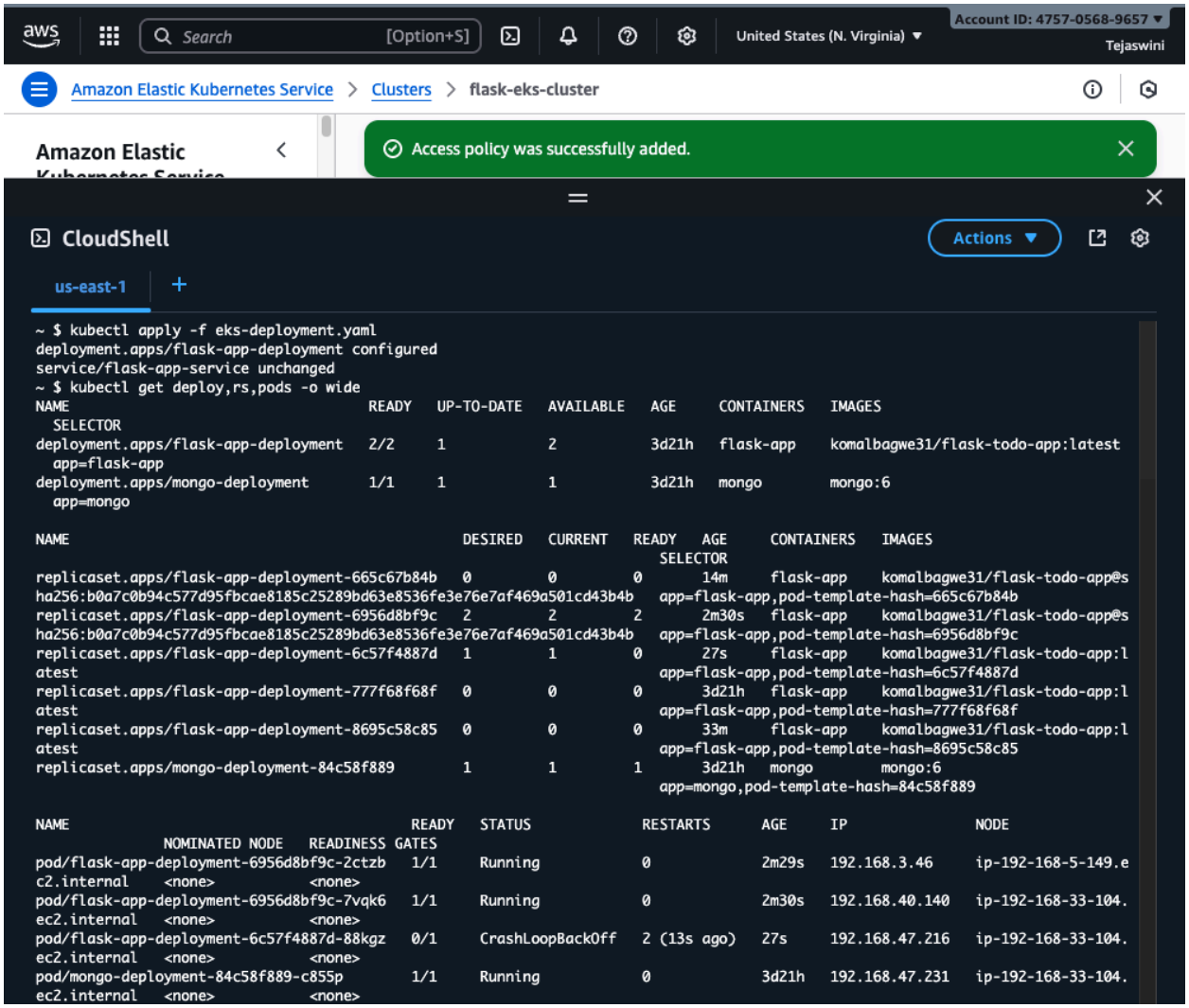
33 lines (32 loc) · 551 Bytes

Code | Blame | Raw

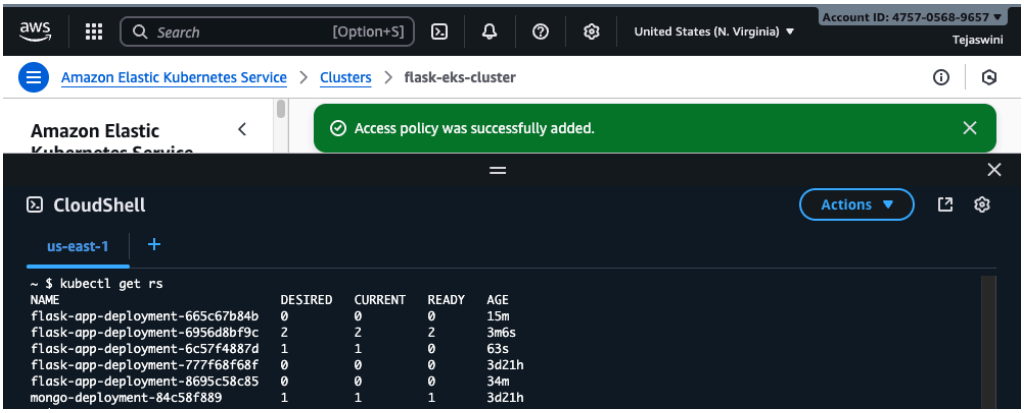
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: komalbagwe31/flask-todo-app:latest
        ports:
        - containerPort: 5000
---

apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  type: LoadBalancer
  selector:
    app: flask-app
  ports:
    - protocol: TCP
```

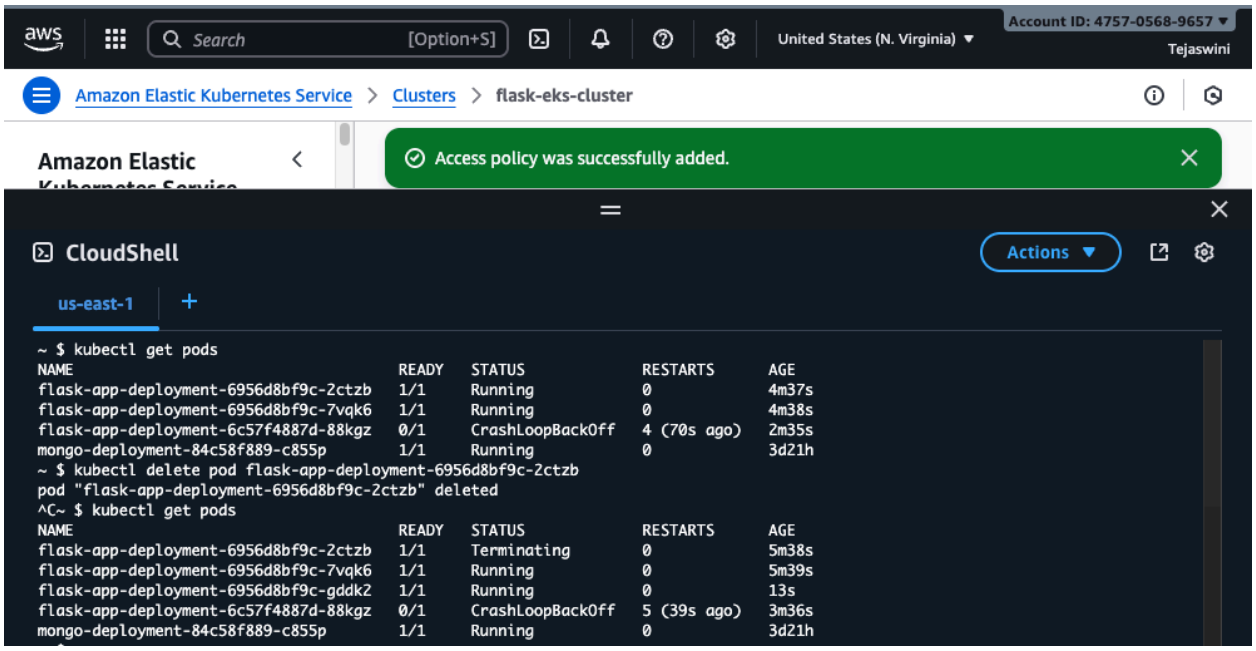
3. Apply the deployment using '**kubectl apply -f eks-deployment.yaml**' .



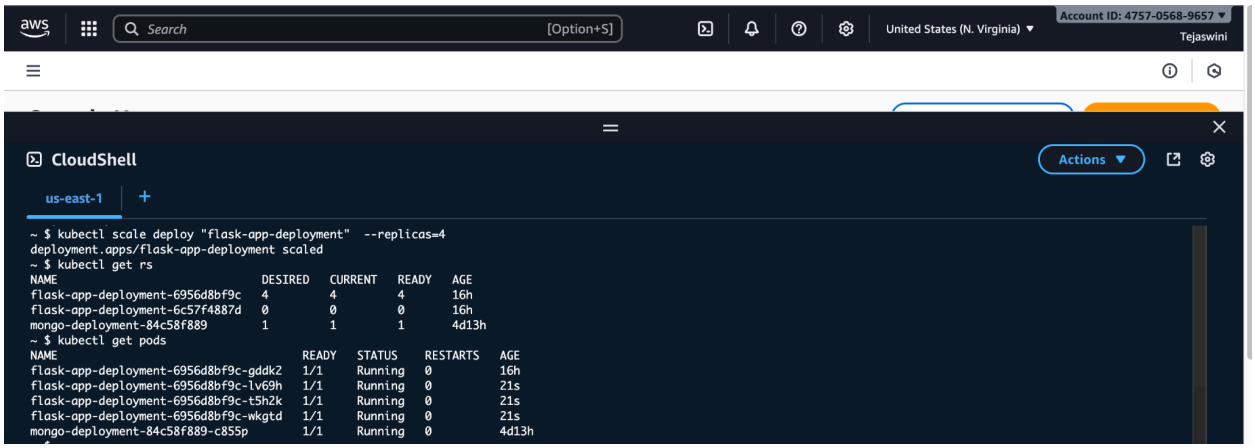
4. Verify the ReplicaSet with '**kubectl get rs**' .



5. Delete one pod and observe Kubernetes automatically creating a replacement pod.
   Kubectl get pods
   kubectl delete pod <POD_NAME>
   kubectl get pods

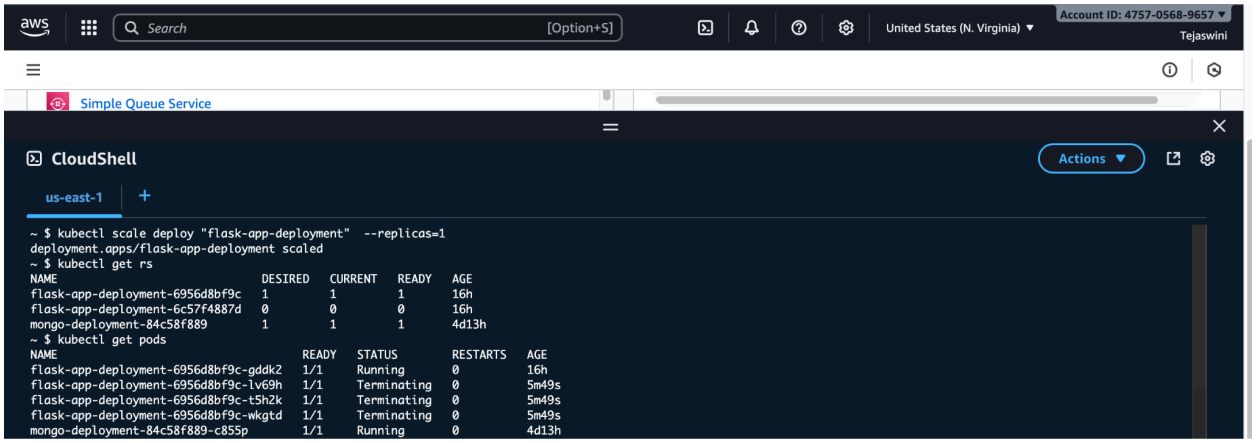


6. Scale up or down using either 'kubectl scale deployment --replicas=n' or by editing the YAML file
   a. Scale up:
      Kubectl scale deploy “flask-app-deployment” –replicas = 4



b. Scale down:

Kubectl scale deploy “flask-app-deployment” –replicas = 1

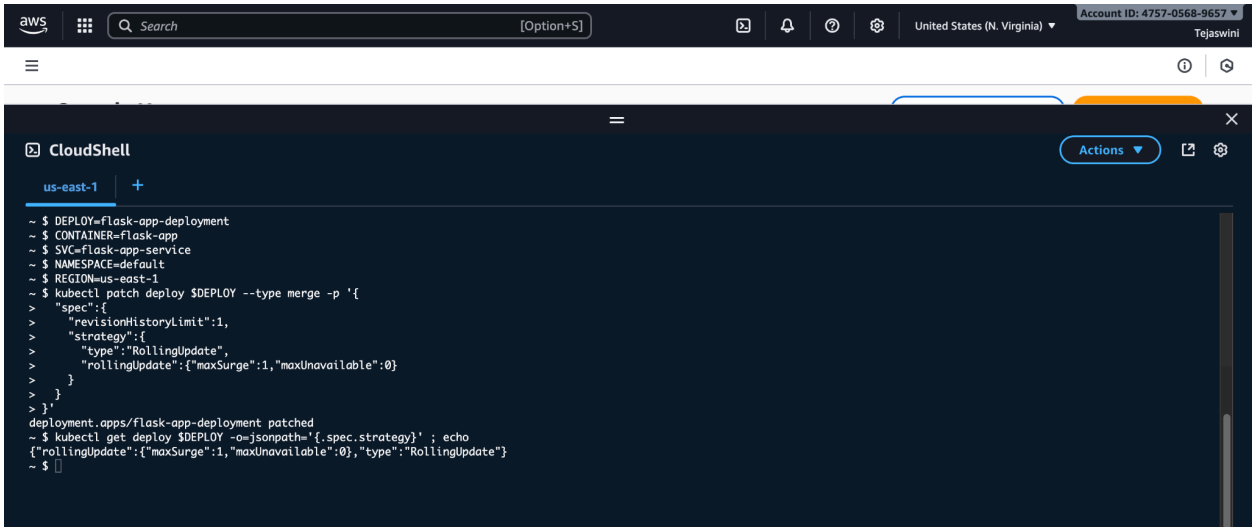


Part 6: Rolling update strategy:

1. Configure the Kubernetes deployment to use a rolling update strategy.

```
kubectl patch deploy $DEPLOY --type merge -p '{
  "spec":{
    "revisionHistoryLimit":1,
    "strategy":{
      "type":"RollingUpdate",
      "rollingUpdate":{"maxSurge":1,"maxUnavailable":0}
    }
  }
}'
kubectl get deploy $DEPLOY -o=jsonpath='{.spec.strategy}' ; echo
```

This keeps all existing pods serving while adding one new pod at a time.

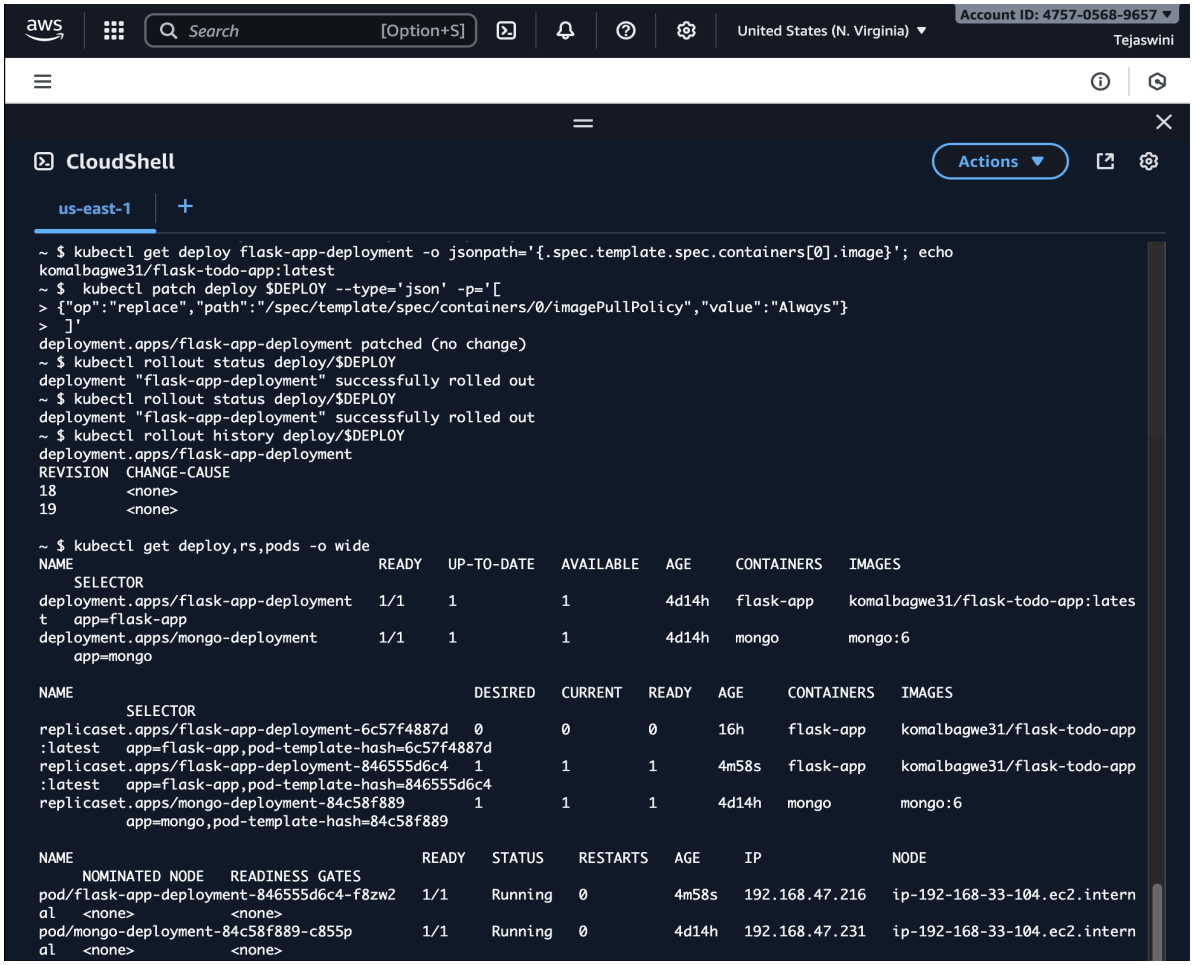


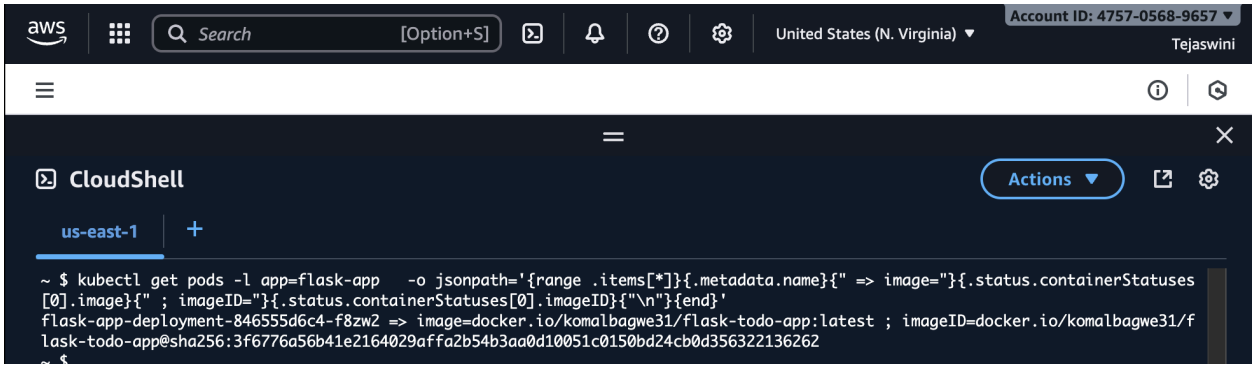
2. Update the Docker image for the deployment to a new version,
   kubectl set image deploy/$DEPLOY flask-app=komalbagwe31/flask-todo-app:latest

3. Monitor roll-out:
   kubectl rollout status deploy/flask-app-deployment
   kubectl rollout history deploy/flask-app-deployment

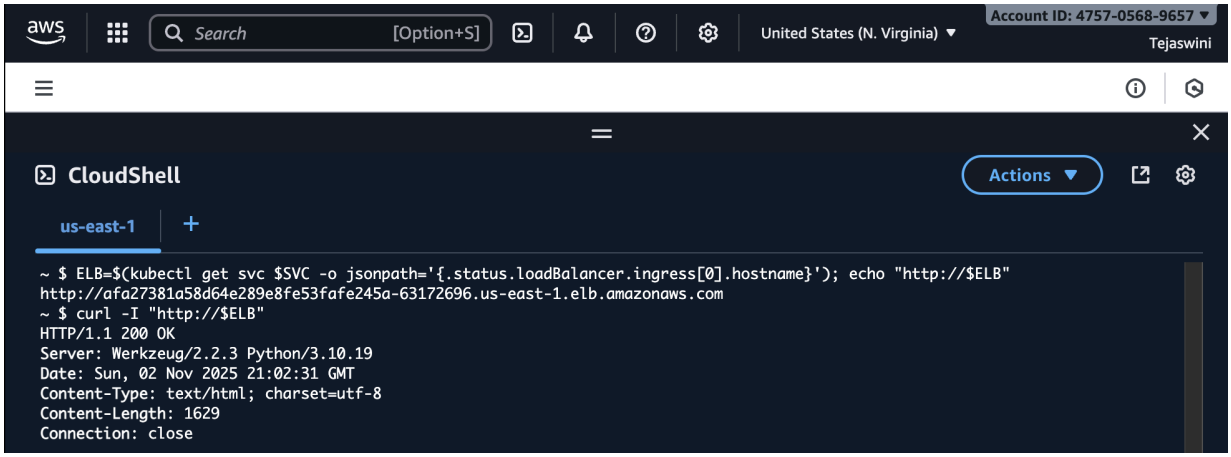
   Show RS + Pods during/after rollout
   Kubectl get deploy,rs,pods -o wide

4. Prove pods run the new image:
   kubectl get pods -l app=flask-app -o jsonpath='{range .items[*]}{.metadata.name}{" => "}{.status.containerStatuses[0].image}{"\n"}{end}'

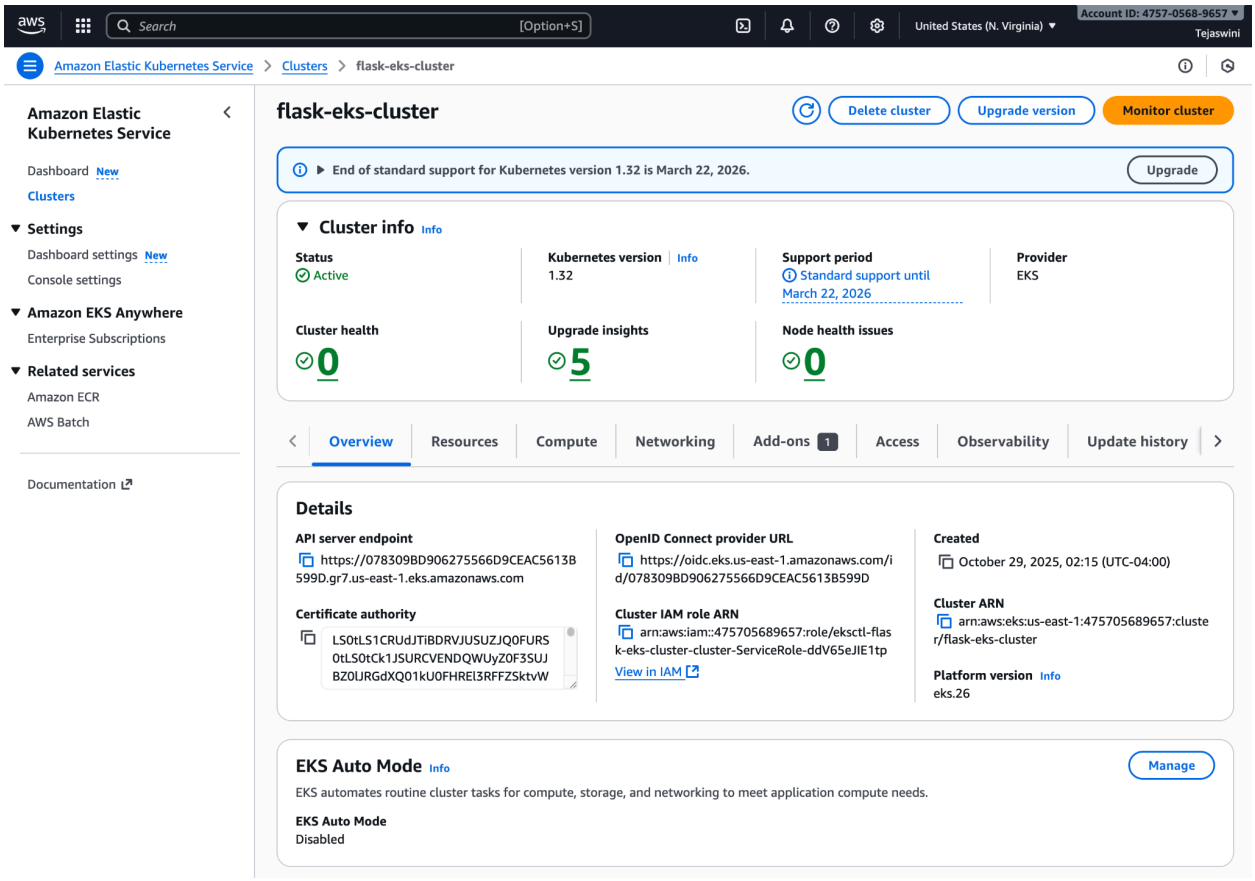




5. Hit the Service to validate app after update:
   ELB=$(kubectl get svc flask-app-service -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'); echo "http://$ELB" curl -I "http://$ELB"



6. Verify pods in Kuberneters dashboard:



Part 7: Health monitoring:

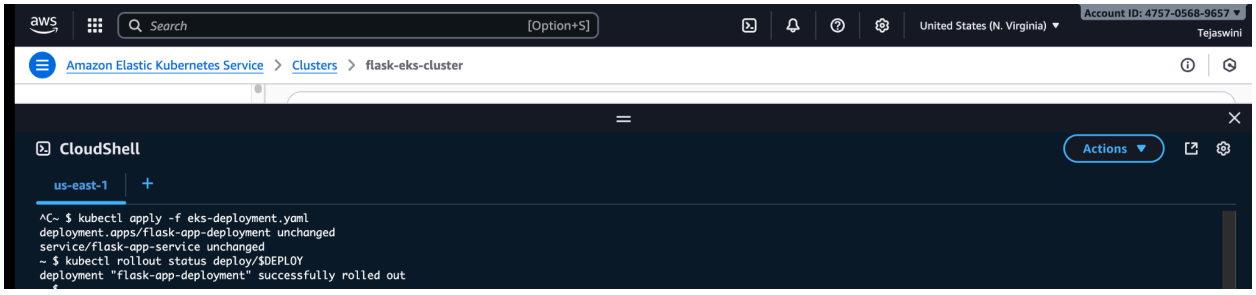
Step 1: Configure Kubernetes to monitor the health of the application by setting up probes for the pods

Updated **eks-deployment.yaml** file (lines: 21-37) to have readiness and liveness probes in the following way:

1. **Readiness probe fails** → Pod is marked **NotReady** and **removed from Service endpoints** (traffic stops).
2. **Liveness probe fails** repeatedly (per failureThreshold) → **Kubelet restarts the container**.

eks-deployment.yaml +23 -5

```
@@ -13,12 +13,29 @@ spec:
          app: flask-app
      spec:
        containers:
-       - name: flask-app
-         image: komalbagwe31/flask-todo-app:latest
-         ports:
-         - containerPort: 5000
+       - name: flask-app
+         image: komalbagwe31/flask-todo-app:latest
+         imagePullPolicy: Always
+         ports:
+           - containerPort: 5000
+         readinessProbe:
+           httpGet:
+             path: /
+             port: 5000
+           initialDelaySeconds: 5
+           periodSeconds: 5
+           timeoutSeconds: 2
+           failureThreshold: 2
+           successThreshold: 1
+         livenessProbe:
+           httpGet:
+             path: /
+             port: 5000
+           initialDelaySeconds: 10
+           periodSeconds: 10
+           timeoutSeconds: 2
+           failureThreshold: 3
```

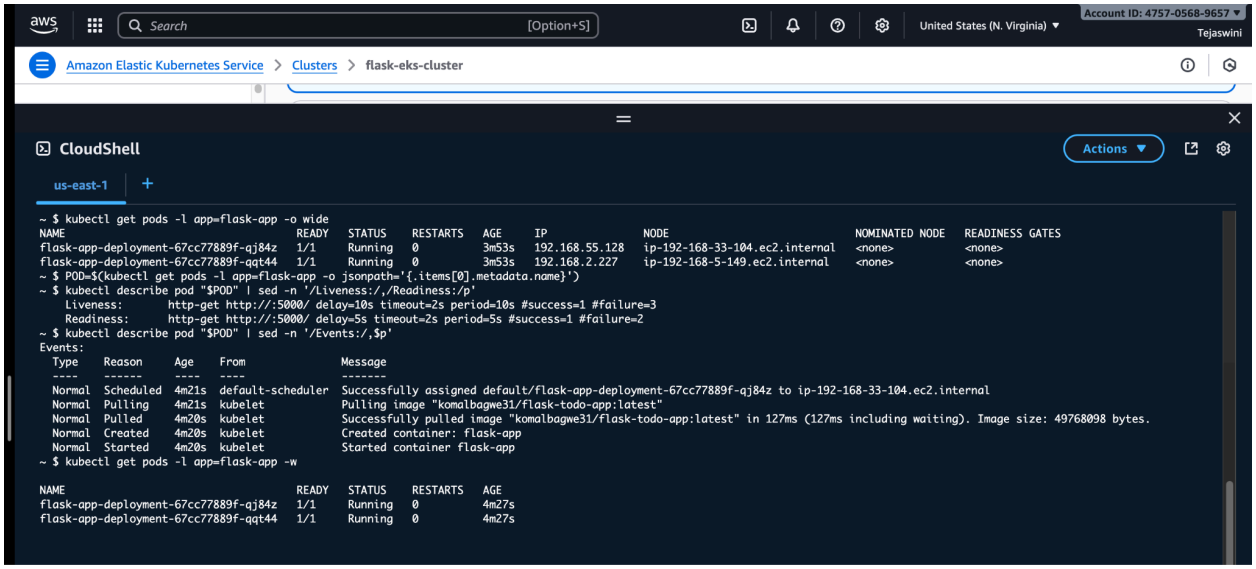


Step 2: Monitoring:

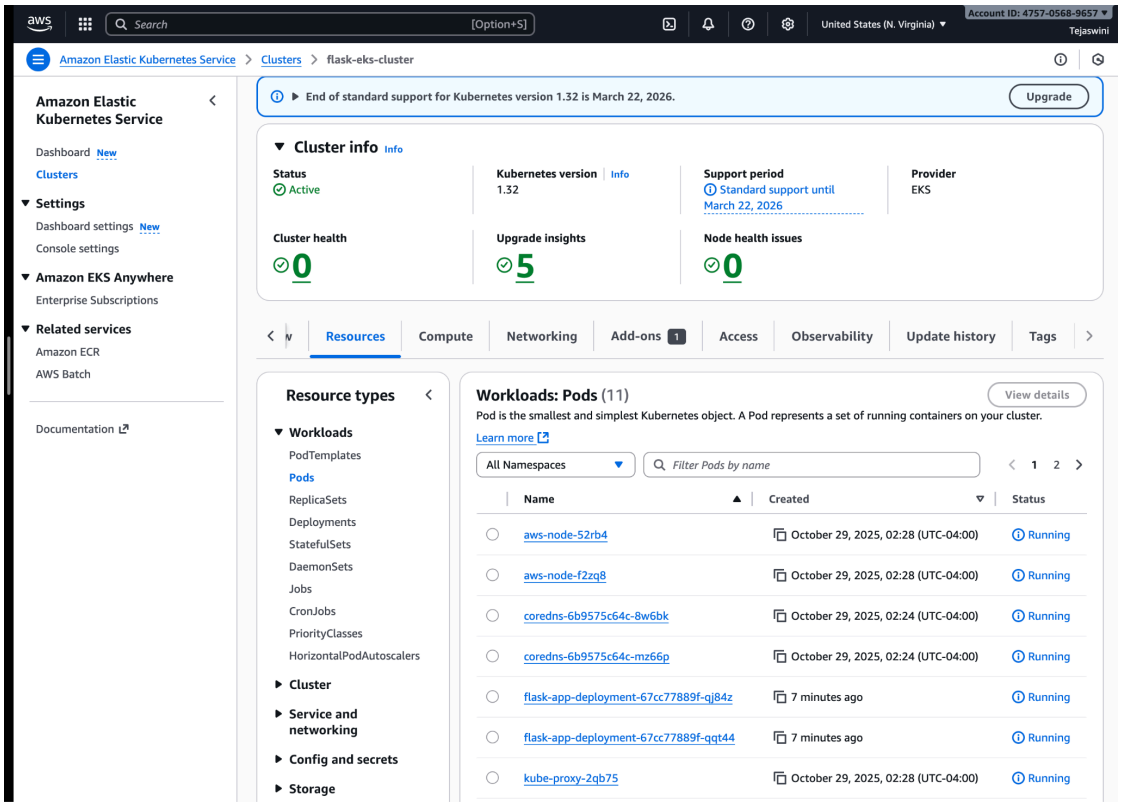
1. Get pods
   kubectl get pods -l app=flask-app -o wide

2. See probe configuration + last events
   POD=$(kubectl get pods -l app=flask-app -o jsonpath='{.items[0].metadata.name}')
   kubectl describe pod "$POD" | sed -n '/Liveness:/,/Readiness:/p'
   kubectl describe pod "$POD" | sed -n '/Events:/,$p'

3. Live watch of status changes
   kubectl get pods -l app=flask-app -w



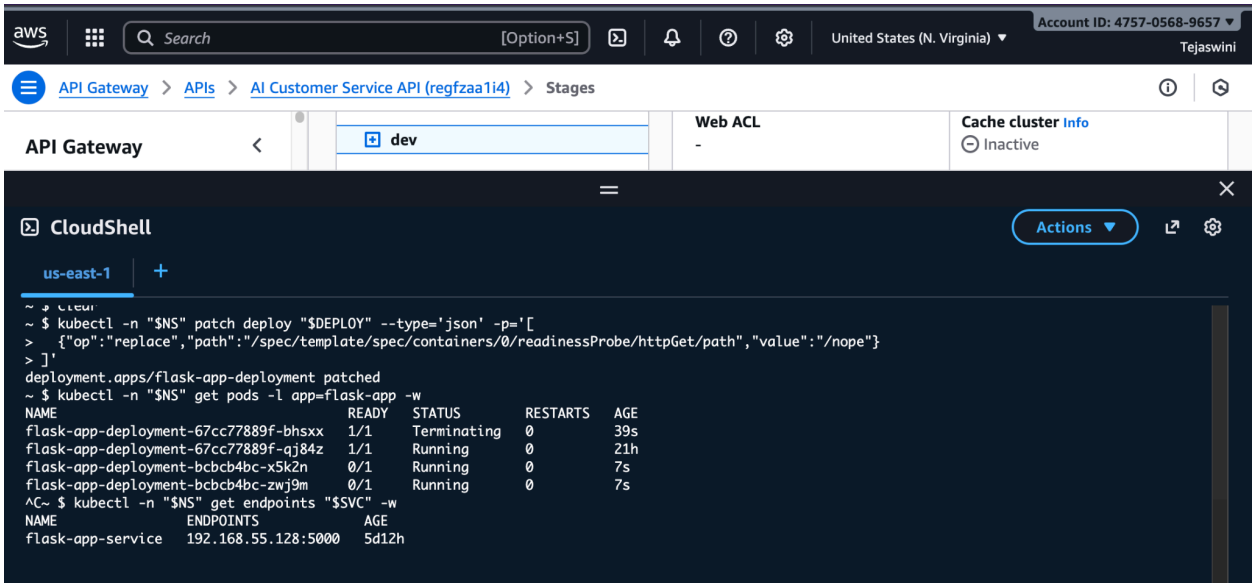
4. Verify in Dashboard:
   AWS Console → EKS → **Clusters → flask-eks-cluster → Workloads / Pods**



5. Test Readiness:
   a. Make readiness check a bad path
      kubectl patch deploy $DEPLOY --type='json' -p='[{"op":"replace","path":"/spec/template/spec/containers/0/readinessProbe/httpGet/path","value":"/nope"}]'

   b. Watch it go NotReady (READY column becomes 0/1)
      kubectl get pods -l app=flask-app -w

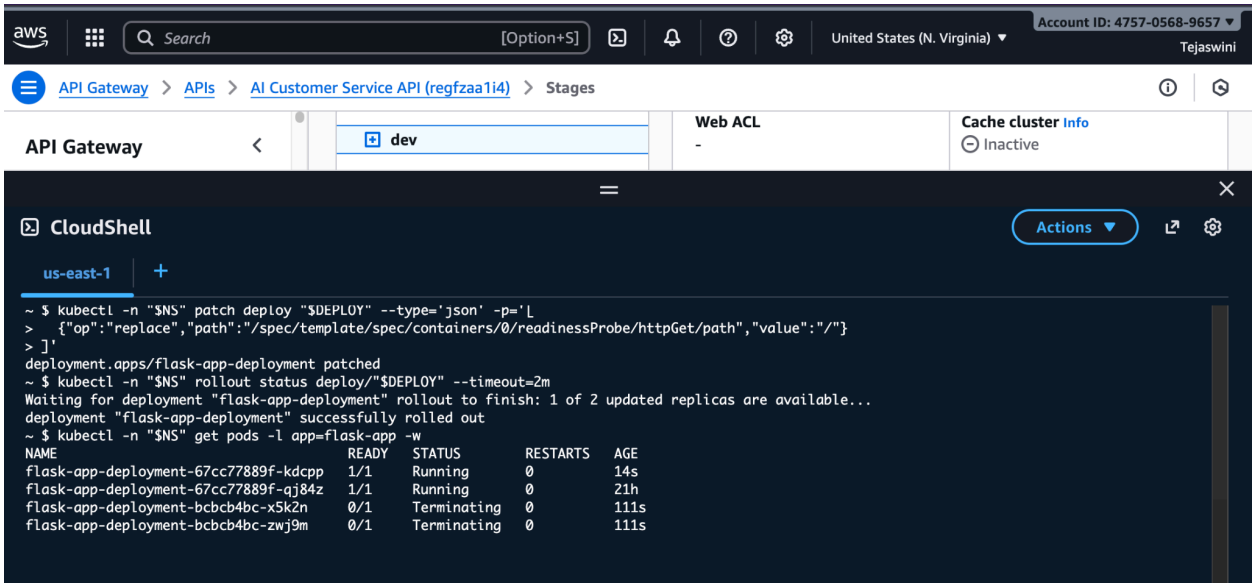
   c. Confirm Service stops sending traffic to that pod
      kubectl get endpoints $SVC -w



d. **Restore** and then check rollout

```
kubectl -n "$NS" patch deploy "$DEPLOY" --type='json' -p='[
{"op":"replace","path":"/spec/template/spec/containers/0/readinessProbe/httpGet/path","value":"/"}]'

kubectl -n "$NS" rollout status deploy/"$DEPLOY" --timeout=2m
```

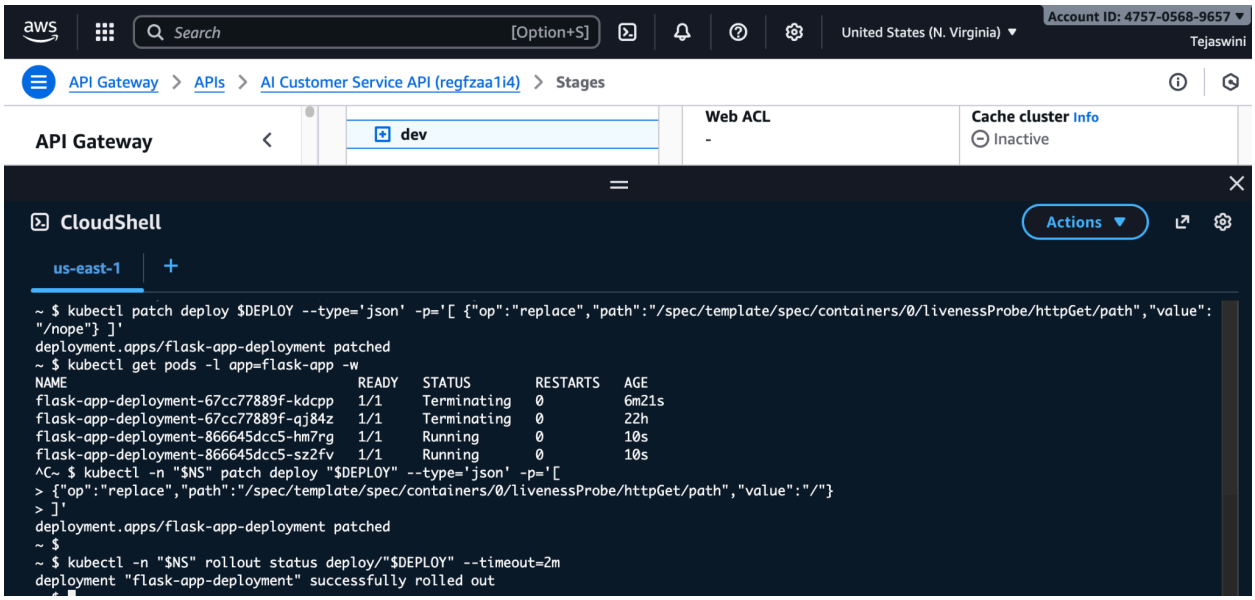


6. Test Liveness:
    a. Make liveness check a bad path so it fails repeatedly
    kubectl patch deploy $DEPLOY --type='json' -p='[{"op":"replace","path":"/spec/template/spec/containers/0/livenessProbe/httpGet/path","value":"/nope"}]'

    b. Watch restarts increment
    kubectl get pods -l app=flask-app -w

    c. **Restore** and then check rollout
    kubectl -n "$NS" patch deploy "$DEPLOY" --type='json' -p='[{"op":"replace","path":"/spec/template/spec/containers/0/livenessProbe/httpGet/path","value":"/"}]'

    kubectl -n "$NS" rollout status deploy/"$DEPLOY" --timeout=2m



Part 8: Alerting (Extra Credit):

1. Install the monitoring stack:
   We have used the **kube-prometheus-stack** Helm chart. It ships with default K8s alert rules like **KubePodNotReady, KubePodCrashLooping**, etc

   export REGION=us-east-1

   export CLUSTER=flask-eks-cluster
   aws eks update-kubeconfig --region "$REGION" --name "$CLUSTER"

   curl -fsSL
   https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

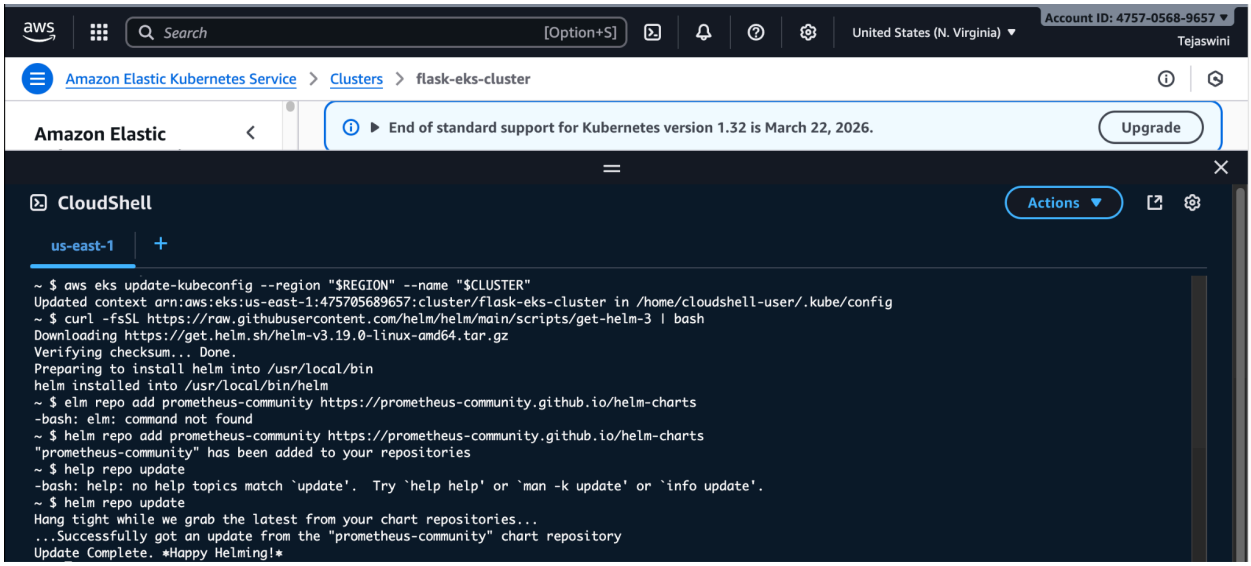
   helm repo add prometheus-community
   https://prometheus-community.github.io/helm-charts

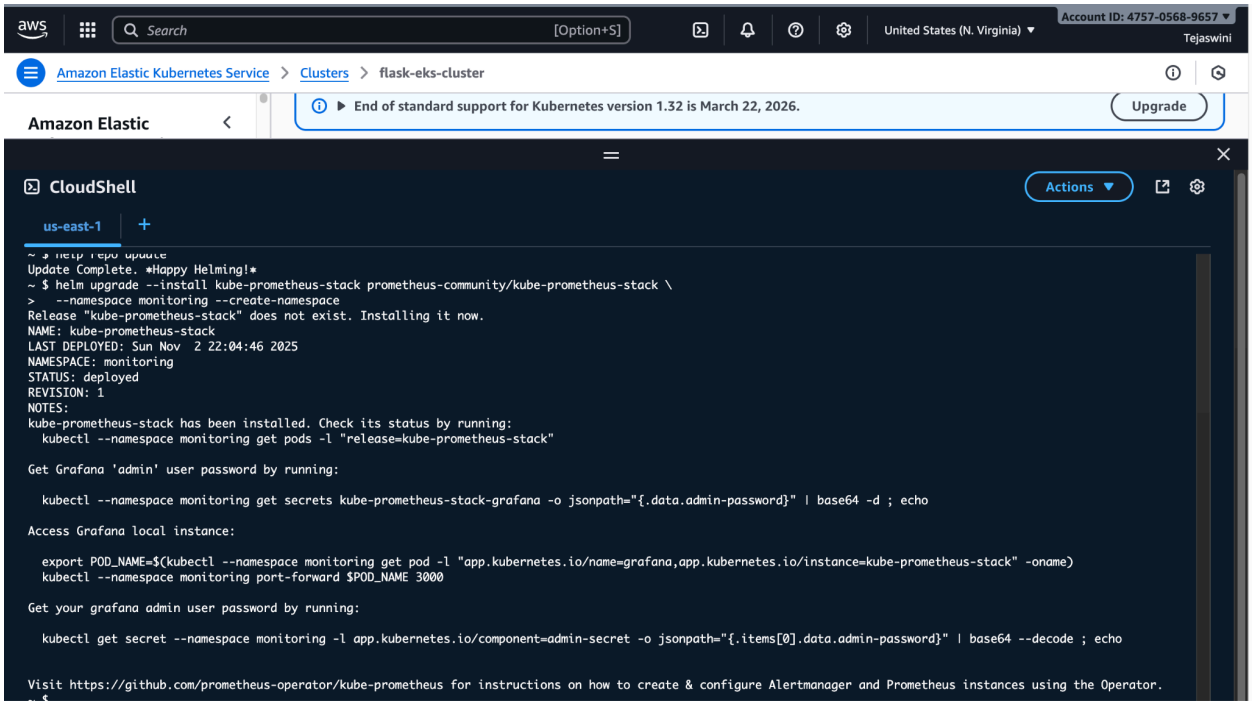
   helm repo update

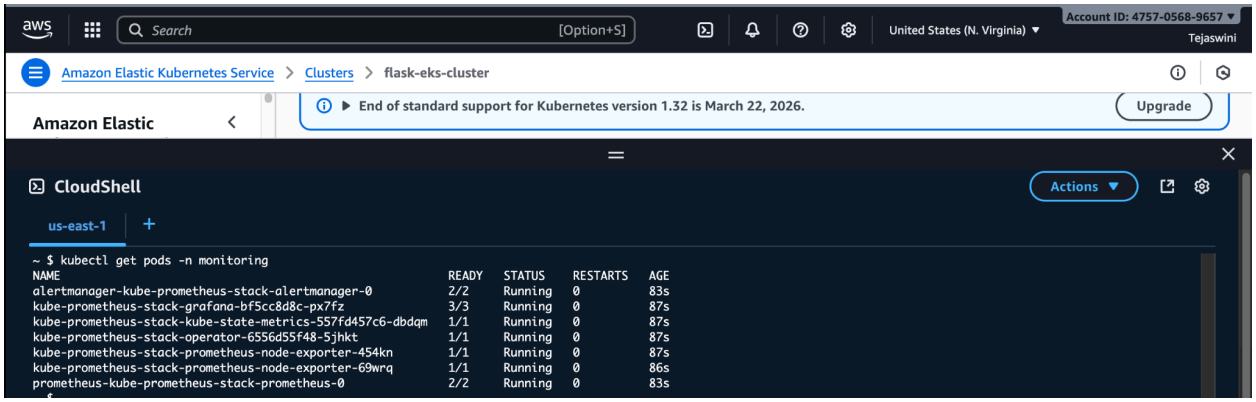
   helm upgrade --install kube-prometheus-stack
   prometheus-community/kube-prometheus-stack -n monitoring
   --create-namespace

   kubectl -n monitoring rollout status
   statefulset/kube-prometheus-stack-prometheus

   kubectl -n monitoring rollout status
   statefulset/kube-prometheus-stack-alertmanager

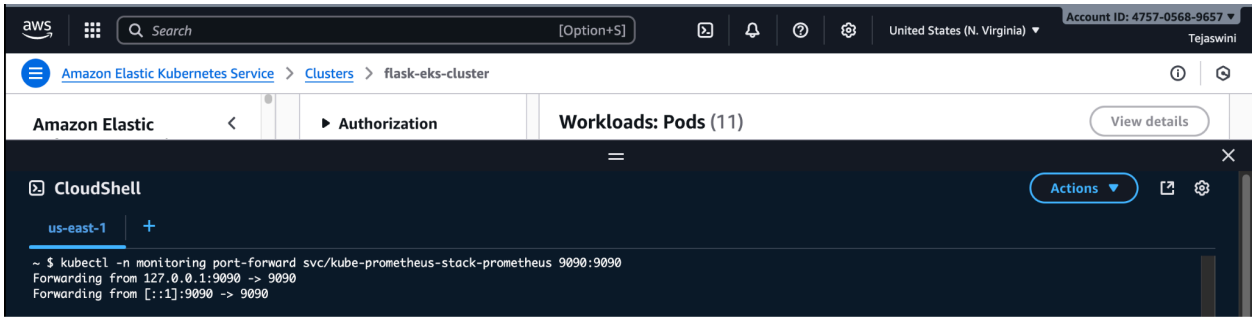

Search
[Option+S]
United States (N. Virginia)
Account ID: 4757-0568-9657
Tejaswini
Amazon Elastic Kubernetes Service
Clusters
flask-eks-cluster
Amazon Elastic
End of standard support for Kubernetes version 1.32 is March 22, 2026.
Upgrade
CloudShell
Actions
us-east-1
~ $ aws eks update-kubeconfig --region "$REGION" --name "$CLUSTER"
Updated context arn:aws:eks:us-east-1:475705689657:cluster/flask-eks-cluster in /home/cloudshell-user/.kube/config
~ $ curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
Downloading https://get.helm.sh/helm-v3.19.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
~ $ elm repo add prometheus-community https://prometheus-community.github.io/helm-charts
-bash: elm: command not found
~ $ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
~ $ help repo update
-bash: help: no help topics match `update'. Try `help help' or `man -k update' or `info update'.
~ $ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helming!*

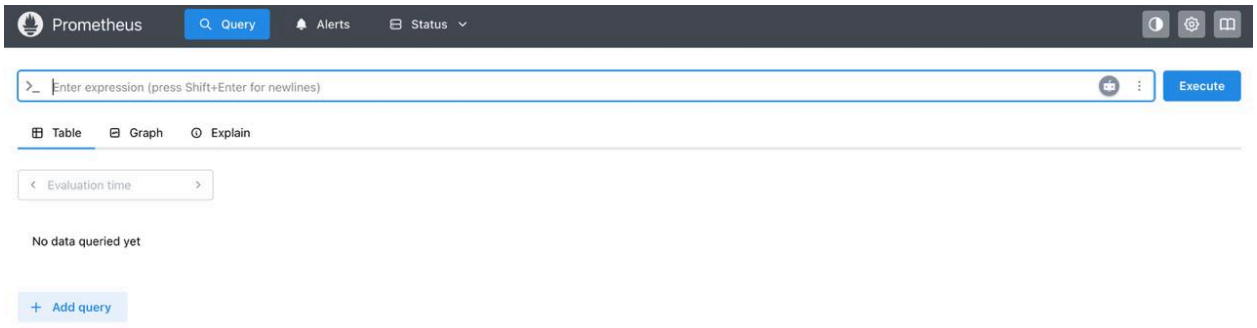

Search
[Option+S]
United States (N. Virginia)
Account ID: 4757-0568-9657
Tejaswini
Amazon Elastic Kubernetes Service
Clusters
flask-eks-cluster
Amazon Elastic
End of standard support for Kubernetes version 1.32 is March 22, 2026.
Upgrade
CloudShell
Actions
us-east-1
Update Complete. *Happy Helming!*
~ $ helm upgrade --install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
> --namespace monitoring --create-namespace
Release "kube-prometheus-stack" does not exist. Installing it now.
NAME: kube-prometheus-stack
LAST DEPLOYED: Sun Nov 2 22:04:46 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
kubectl --namespace monitoring get pods -l "release=kube-prometheus-stack"
Get Grafana 'admin' user password by running:
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo
Access Grafana local instance:
export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=kube-prometheus-stack" -oname)
kubectl --namespace monitoring port-forward $POD_NAME 3000
Get your grafana admin user password by running:
kubectl get secret --namespace monitoring -l app.kubernetes.io/component=admin-secret -o jsonpath="{.items[0].data.admin-password}" | base64 --decode ; echo
Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.



Prometheus:

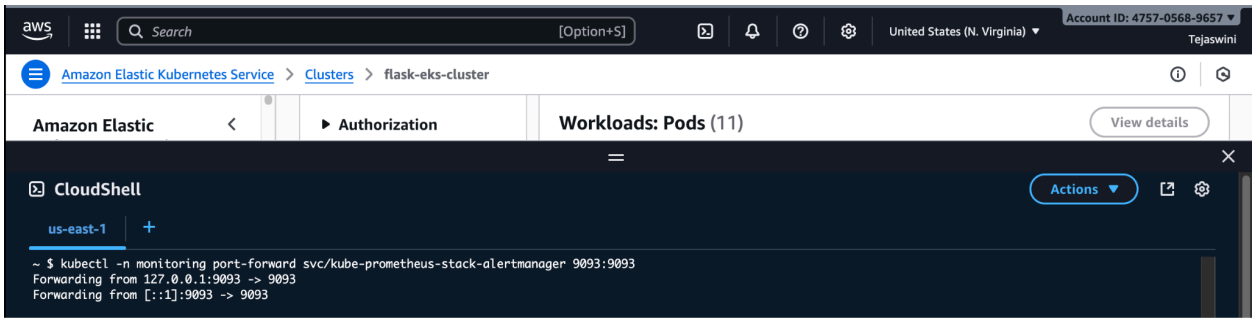
kubectl -n monitoring port-forward svc/kube-prometheus-stack-prometheus 9090:9090

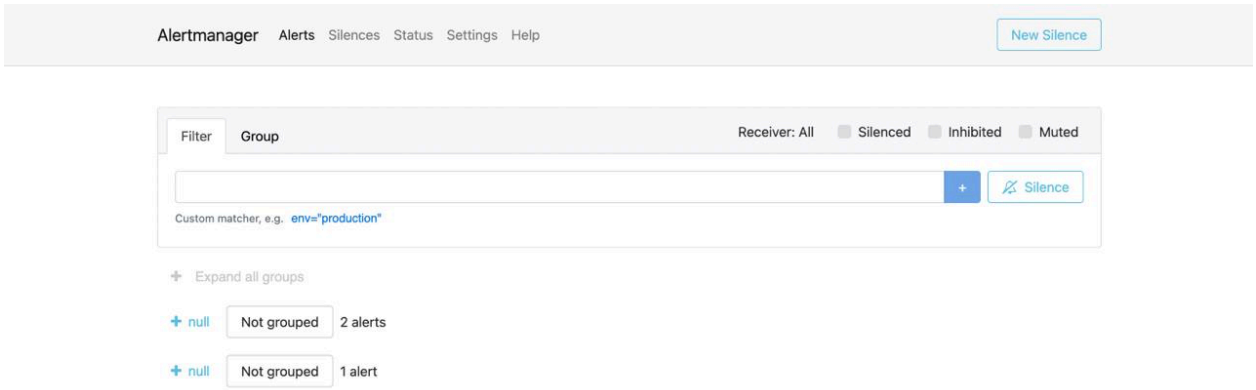




Alertmanager:

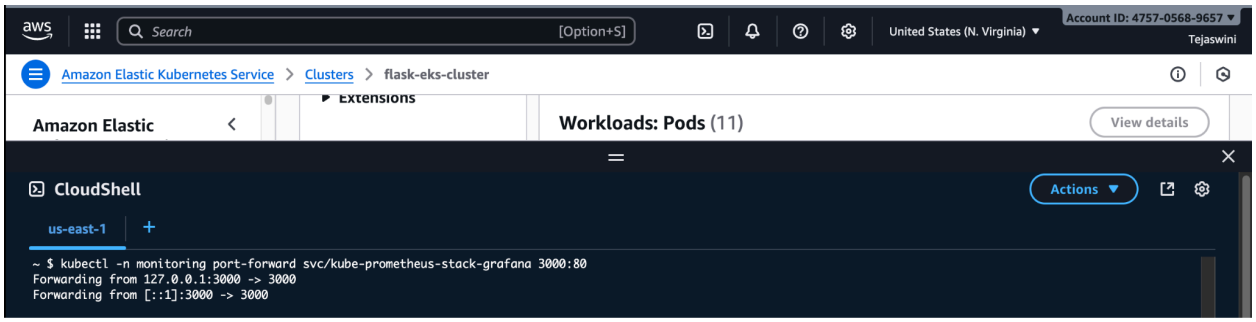
kubectl -n monitoring port-forward svc/kube-prometheus-stack-alertmanager 9093:9093

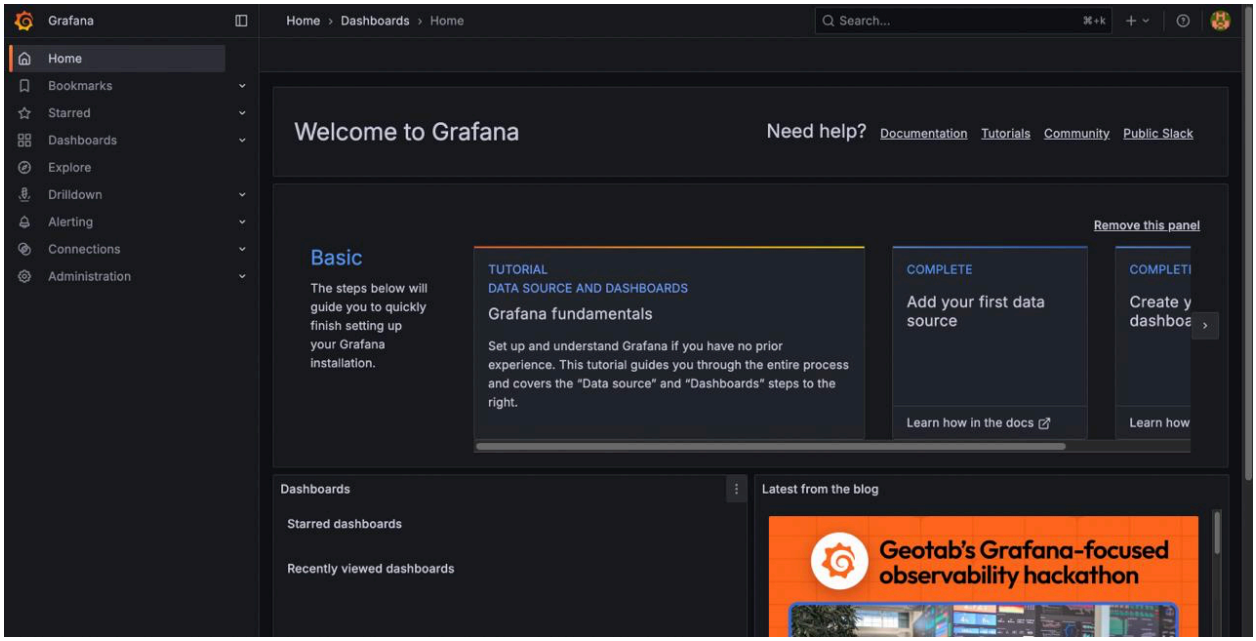




Grafana:

kubectl -n monitoring port-forward svc/kube-prometheus-stack-grafana 3000:80





2. Integrating Slack:
    a. Added Webhook:

```
~ $ kubectl -n monitoring create secret generic slack-webhook   --from-literal=url='https://hooks.slack.com/services/T09PV9KT2BH/B09PVA588UX/dBhnqz0NdZqkspx
W63UnIEUZ'   --dry-run=client -o yaml | kubectl apply -f -
secret/slack-webhook configured
~ $
```

```
~ $ kubectl -n monitoring create secret generic slack-webhook   --from-literal=url='https://hooks.slack.com/services/T09PV9KT2BH/B09PVA588UX/dBhnqz0NdZqkspx
W63UnIEUZ'   --dry-run=client -o yaml | kubectl apply -f -
secret/slack-webhook configured
~ $ cat <<'YAML' | kubectl -n monitoring apply -f -
> apiVersion: monitoring.coreos.com/v1alpha1
> kind: AlertmanagerConfig
> metadata:
>   name: slack-notify
>   labels:
>     alertmanagerConfig: "true"
> spec:
>   route:
>     receiver: slack-default
>     groupBy: ["job"]
>     groupWait: 30s
>     groupInterval: 5m
>     repeatInterval: 4h
>   receivers:
>   - name: slack-default
>     slackConfigs:
>     - channel: "#alerts"
>       sendResolved: true
>       apiURL:
>         name: slack-webhook
>         key: url
>       title: '[{{ .Status | toUpper }}] {{ .CommonLabels.alertname }}'
>       text: >-
>         *Cluster:* {{ .ExternalURL }}
>         *Severity:* `{{ .CommonLabels.severity }}`
>         {{- range .Alerts }}
>         • *{{ .Labels.alertname }}* on `{{ .Labels.instance }}` – {{ .Annotations.summary }} {{ .Annotations.description }}
>         {{- end }}
> YAML
alertmanagerconfig.monitoring.coreos.com/slack-notify created
```

    b. Test:

```
us-east-1 ×  +
~ »
~ $ cat <<'YAML' | kubectl -n monitoring apply -f -
> apiVersion: monitoring.coreos.com/v1
> kind: PrometheusRule
> metadata:
>   name: test-slack-alert
>   labels: { release: kube-prometheus-stack }
> spec:
>   groups:
>   - name: test
>     rules:
>     - alert: AlwaysFiring
>       expr: vector(1)
>       for: 1m
>       labels:
>         severity: critical
>         namespace: monitoring   # <-- required to match your routes
>       annotations:
>         summary: "Test Slack alert"
>         description: "This is a test alert from Alertmanager via Slack."
> YAML

prometheusrule.monitoring.coreos.com/test-slack-alert configured
```

Slack output: