Cloud Computing and Big Data Systems - Spring 2024 Assignment 2

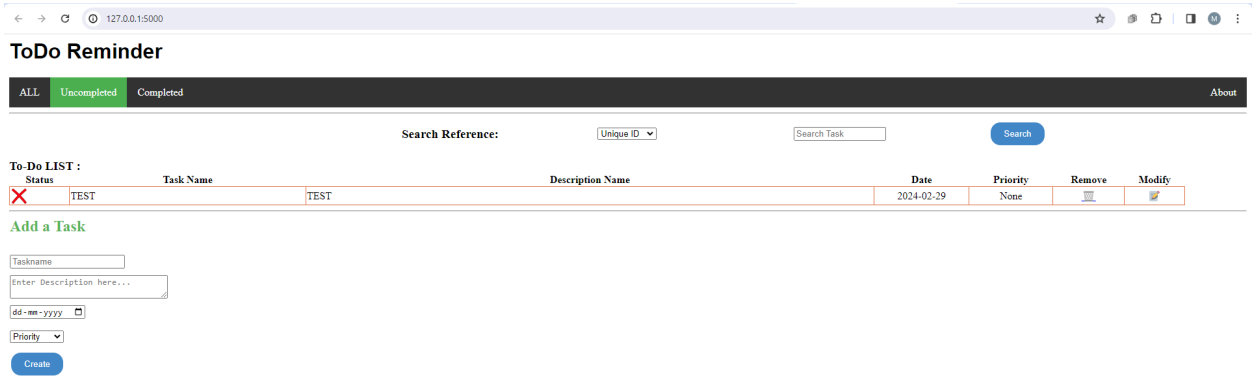
Moin Khan(mk8793), Vir Jhangiani(vrj2006)

Objective:

In this assignment, you will learn how to deploy a web application on Kubernetes using Docker containers. You will start by containerizing the application using Docker, create a persistence volume, and push the image to Docker Hub. Then, you will deploy the container on a local Kubernetes cluster using Minikube and then deploy it on AWS EKS.You will also explore various Kubernetes features such as a replication controller, health monitoring, rolling updates, and alerting.

Part 1: Creating application locally

We ran the application locally to make sure it was working properly.



Part 2: Containerizing the Application on Docker

Wrote Dockerfile for the flask application with all required detail:

Dockerfile.yaml

```
FROM python:3.9-slim
RUN apt-get update; apt-get install iputils-ping curl -y
WORKDIR /flask-docker

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

Docker build(Building the Docker image using the Docker CLI (command-line interface):

```
docker build . -t flask-docker-image
```

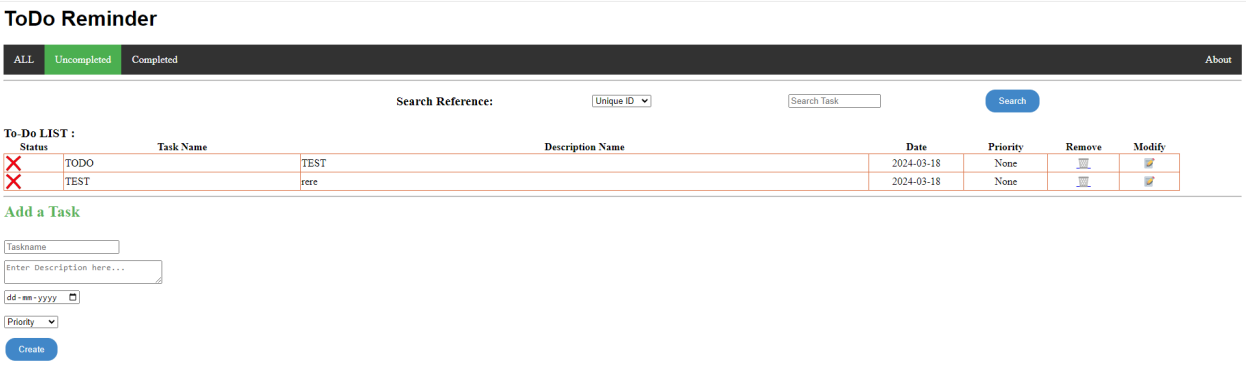
Testing the application locally, using docker-compose file that defines the services for the flask app and MongoDB container. The docker-compose file specifies the images to use, the ports to expose, and a volume to mount to store the database information.

```
docker compose up
```

docker-compose.yaml

```
version: '3'
services:
  flask:
    image: flask-docker-image
    environment:
      - MONGO_HOST=mongodb
      - MONGO_PORT=27017
      - FLASK_ENV=development
      - PORT=5000
    depends_on:
      - mongodb
    ports:
      - "5000:5000"
  mongodb:
    image: mongo:4.4.15
    ports:
      - "27019:27017"
    expose:
      - "27017"
    environment:
      - MONGODB_INITDB_ROOT_USERNAME=admin
      - MONGODB_INITDB_ROOT_PASSWORD=admin
    volumes:
      - type: bind
        source: ./data
```

```
        target: /data/db
```



Pushing the Docker image to Docker Hub to make it available for deployment to Kubernetes. Below are the steps using Docker CLI to push the image to the Docker Hub registry.

Docker image tag

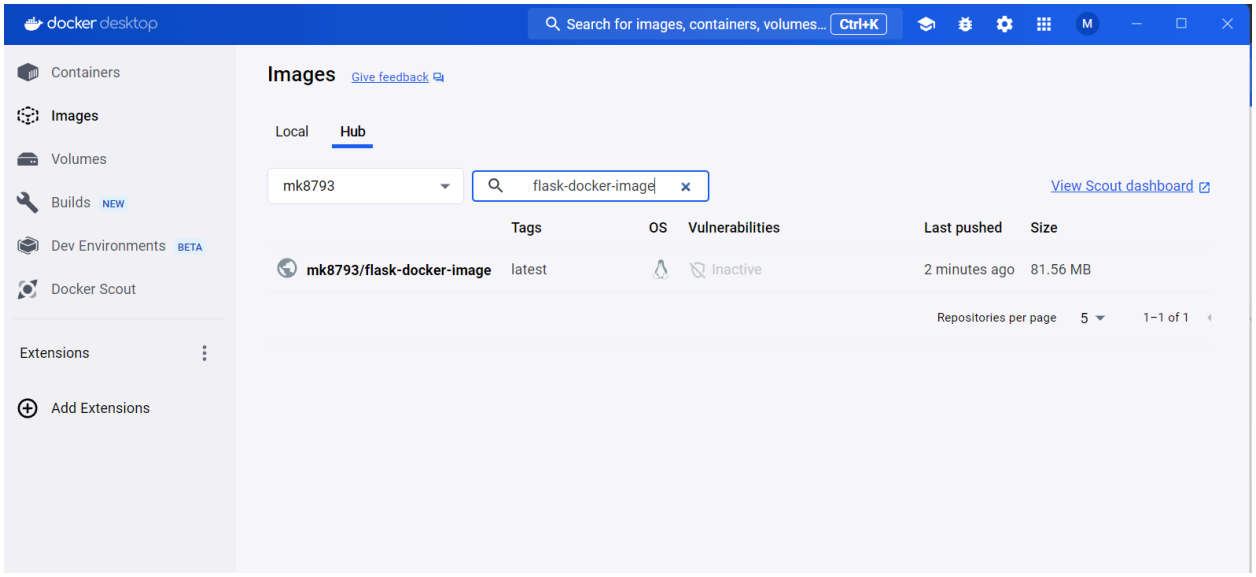
```
docker tag flask-docker-image mk8793/flask-docker-image
```

Docker push image

```
docker image push mk8793/flask-docker-image:latest
```

```
C:\Users\moink\Downloads\CC\HW2\Docker>docker image push mk8793/flask-docker-image:latest
The push refers to repository [docker.io/mk8793/flask-docker-image]
80c47523f658: Pushed
d814c5ae2bd9: Mounted from mk8793/flask-docker-new
24a0d725fa79: Mounted from mk8793/flask-docker-new
e5588c97b653: Mounted from mk8793/flask-docker-new
5c8fe12dc456: Mounted from mk8793/flask-docker-new
4a7ac3585b06: Mounted from mk8793/flask-docker-new
6be461d39d4d: Mounted from mk8793/flask-docker-new
d91aa0e727e2: Mounted from mk8793/flask-docker-new
c8f253aef560: Mounted from mk8793/flask-docker-new
a483da8ab3e9: Mounted from mk8793/flask-docker-new
latest: digest: sha256:5cd454502433a6188bd8c6b1263740c15bcd4f8f2f96b259f0eaccd65b354d31 size: 2419
```

Below you will find a screenshot of the image available on docker hub



Part 3: Deploying the Application on Minikube:

Start Minikube using the command-line interface.

```
minikube start
```

Creating PV using file mongo_pv.yml

```
kubectl apply -f mongo_pv.yml
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f mongo_pv.yml
persistentvolume/mongo-pv created

C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f mongo_pv.yml

C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                 STORAGECLASS   REASON   AGE
mongo-pv                                   256Mi      RWO            Retain           Available                                                  17s
pvc-f221ee67-696f-4746-9847-ebd1af7246e8   256Mi      RWO            Delete           Released    default/mongo-pvc     standard               15d
```

mongo_pv.yml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongo-pv
spec:
  capacity:
```

```
    storage: 256Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/db
  storageClassName: "standard"
```

Create Persistent Volume Claim using mongo_pvc.yml

```
kubectl apply -f mongo_pvc.yml
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f mongo_pvc.yml
persistentvolumeclaim/mongo-pvc created

C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mongo-pvc   Bound    pvc-97a17253-37d6-43ac-beb9-2c0795fd7f14   256Mi      RWO            standard       3s
```

mongo_pvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
```

Creating mongo Service using mongo_svc.yml:

```
Kubectl apply -f mongo_svc.yml
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f mongo_svc.yml
service/mongodb-service configured
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl get service
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
kubernetes        ClusterIP      10.96.0.1        <none>        443/TCP           17d
mongodb-service   LoadBalancer   10.111.170.240   <pending>     27019:30218/TCP   6d3h
```

mongo_svc.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb
  ports:
    - port: 27017
    targetPort: 27017
  type: LoadBalancer
```

Here the mongodb service would be accessible outside the cluster through **LoadBalancer on port 27017**. Any pod which wants to communicate with the mongodb can use the mongodb-service as a valid hostname for mongodb.

Creating mongo container in a pod using mongo_db_deployment.yml:

```
kubectl apply -f mongo_db_deployment.yml
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f mongo_db_deployment.yml
deployment.apps/mongodb-deployment created
```

mongo_db_deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    app: mongodb
template:
  metadata:
    labels:
      app: mongodb
  spec:
    containers:
    - name: mongodb-container
      image: mongodb/mongodb-community-server:6.0-ubi8
      ports:
      - containerPort: 27017
      volumeMounts:
      - name: storage
        mountPath: /data/db
      readinessProbe:
        exec:
          command:
          - /bin/sh
          - -c
          - mongo --eval "db.adminCommand('ping')"
        initialDelaySeconds: 15
        periodSeconds: 10
    volumes:
      - name: storage
        persistentVolumeClaim:
          claimName: mongo-pvc
```

The mongodb deployment yaml will create a pod with mongodb-container. The ReplicationController will manage 1 replica. To check whether mongodb db started or not, we created the readinessProbe which would check the mongodb status before sending any request to the pod.

Now deploying flask container on pod with replication controller using flask_rc.yml:

```
kubectl apply -f flask_rc.yml
```

```
C:\Users\moink\Downloads\CC\HW2\Minikube>kubectl apply -f flask_rc.yml
deployment.apps/flask-rc created
```

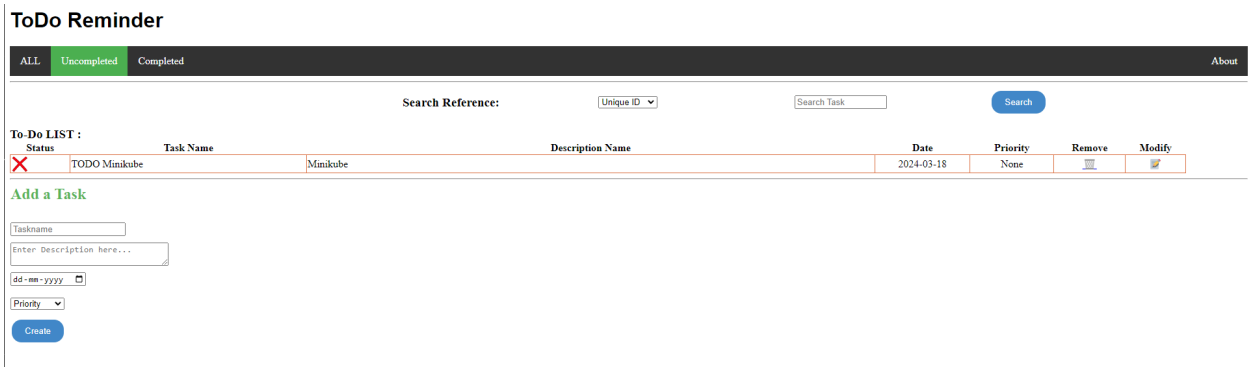
flask_rc.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-rc
spec:
  replicas: 3
  selector:
    matchLabels:
      app: flask
  template:
    metadata:
      name: flask
      labels:
        app: flask
    spec:
      containers:
      - name: flask-app-container
        image: mk8793/flask-docker-new:latest
        ports:
        - containerPort: 5000
        env:
          - name: MONGO_HOST
            value: mongodb-service
          - name: MONGO_PORT
            value: "27017"
          - name: MONGODB_USERNAME
            value: admin
          - name: MONGODB_PASSWORD
            value: admin
          - name: FLASK_ENV
            value: development
          - name: PORT
            value: "5000"
        livenessProbe:
            httpGet:
              path: /
              port: 5000
            initialDelaySeconds: 10
```

```
      periodSeconds: 30
      failureThreshold: 3
    readinessProbe:
      exec:
        command:
        - /bin/sh
        - -c
        - curl --fail http://mongodb-service:27017/ || exit 1
      initialDelaySeconds: 10
      periodSeconds: 30
      failureThreshold: 4
```

To test the app locally, run

```
minikube tunnel
```

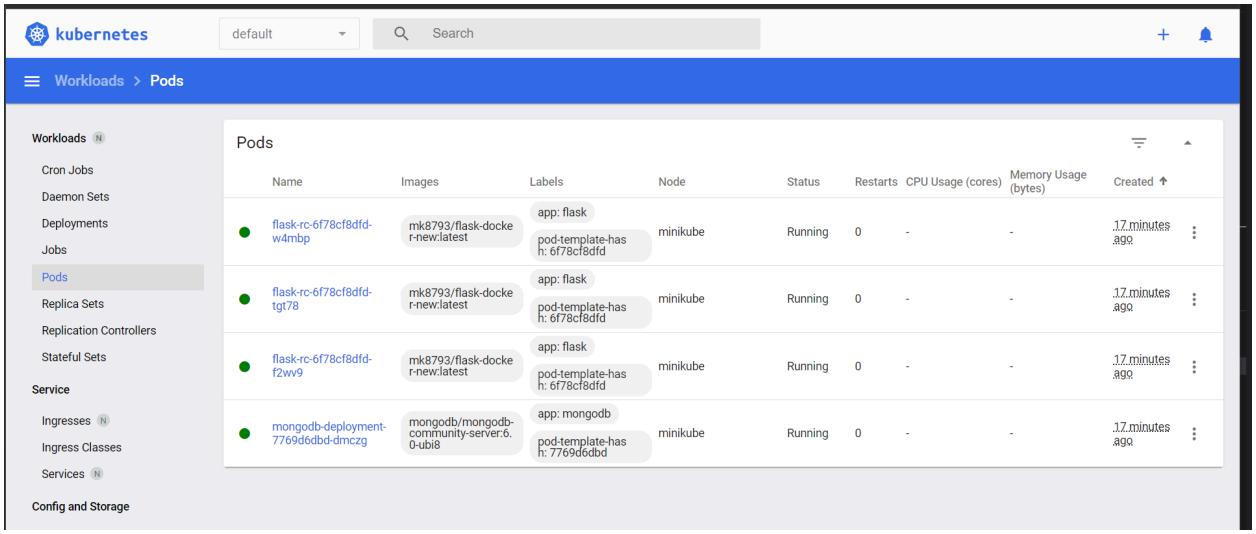


Part 5: Replication controller feature:

In the YAML file above flask deployment will pull the image from mk8793/flask-docker-image:latest from the docker hub. The deployment also has replicationController enabled, which would manage 3 replicas, i.e 3 pods should be there at least incase of failure/scale down.

To Observe our running pods including the Replication Controller(3 pods for flask):

```
minikube dashboard
```



Part 7: Health monitoring

The deployment has two probes you can observe their definition in the flask_rc.yml file above:

- **Liveness probe**, which sends an HTTP Get request to the flask application to check whether the flask is active or not.
- **Readiness Probe** checks whether the flask application is able to make a connection with the mongodb container before accepting any request.

Part 6: RollingUpdate

Update docker image version to v2

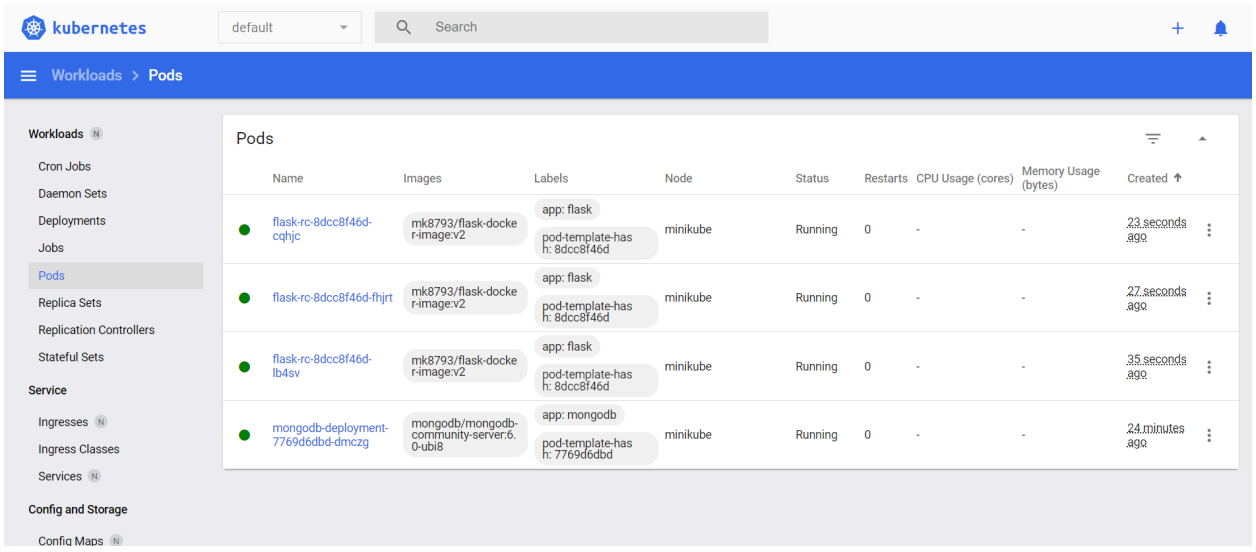
```
C:\Users\moink\Downloads\CC\HW2\TODO>docker tag flask-docker-image mk8793/flask-docker-image:v2

C:\Users\moink\Downloads\CC\HW2\TODO>docker image push mk8793/flask-docker-image:v2
The push refers to repository [docker.io/mk8793/flask-docker-image]
80c47523f658: Layer already exists
d814c5ae2bd9: Layer already exists
24a0d725fa79: Layer already exists
e5588c97b653: Layer already exists
5c8fe12dc456: Layer already exists
4a7ac3585b06: Layer already exists
6be461d39d4d: Layer already exists
d91aa0e727e2: Layer already exists
c8f253aef560: Layer already exists
a483da8ab3e9: Layer already exists
v2: digest: sha256:5cd454502433a6188bd8c6b1263740c15bcd4f8f2f96b259f0eaccd65b354d31 size: 2419
```

Then update kubernetes image from latest to v2

```
C:\Users\moink\Downloads\CC\HW2\TODO>kubectl set image deployments/flask-rc flask-app-container=mk8793/flask-docker-image:v2
deployment.apps/flask-rc image updated
```

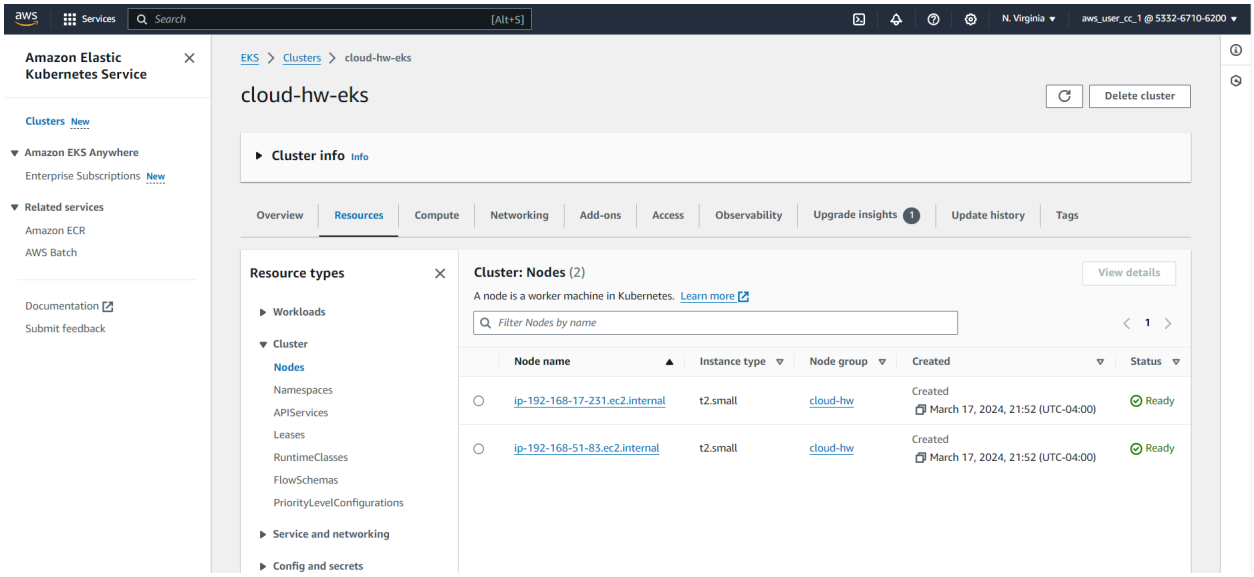
Below you can observe that our image has changed from
`mk8793/flask-docker-new:latest`
to
`mk8793/flask-docker-new:v2`



Part 4: Deploying the Application on AWS EKS

Create cluster:

```
eksctl create cluster --name cloud-hw-eks --region us-east-1
--nodegroup-name cloud-hw --node-type t2.small --nodes 3 --nodes-min 1
--nodes-max 5 --managed
```



IAM permission for IAM role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:Attach*",
                "autoscaling:Describe*",
                "iam:List*",
                "ec2:DeleteTags",
                "iam:UntagRole",
                "iam:TagRole",
                "iam:TagMFADevice",
                "iam:TagSAMLProvider",
                "iam:Put*",
                "iam:PassRole",
                "iam:Get*",
                "autoscaling:UpdateAutoScalingGroup",
                "iam:UntagSAMLProvider",
```

```
                "ssm:Get*",
                "ec2:CreateTags",
                "iam:Create*",
                "iam:Detach*",
                "cloudformation:*",
                "iam:UntagServerCertificate",
                "iam:TagPolicy",
                "iam:TagUser",
                "ec2:CreateVolume",
                "iam:UntagUser",
                "iam:Delete*",
                "ec2:Describe*",
                "iam:Update*",
                "iam:UntagMFADevice",
                "iam:TagServerCertificate",
                "iam:UntagPolicy",
                "ssm:List*",
                "iam:UntagOpenIDConnectProvider",
                "eks:*",
                "iam:UntagInstanceProfile",
                "iam:TagOpenIDConnectProvider",
                "iam:TagInstanceProfile"
            ],
            "Resource": "*"
        }
    ]
}
```

Configure Kuberntes:

```
aws eks update-kubeconfig --region us-east-1 --name cloud-hw-eks
```

Create OIDC Provider:

```
eksctl utils associate-iam-oidc-provider --cluster cloud-hw-eks --approve
```

Check OIDC:

```
aws iam list-open-id-connect-providers
```

Create Storage Class (EBS) for persistentVolumes (required for MongoDb) using **kubectl create -f eks_ebs_sc.yml**

```
C:\Users\moink\Downloads\CC\HW2\EKS>kubectl create -f eks_ebs_sc.yml
storageclass.storage.k8s.io/gp2 created

C:\Users\moink\Downloads\CC\HW2\EKS>kubectl get storageclass
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false                  6s
```

eks_ebs_sc.yml

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: gp2
parameters:
  fsType: ext4
  type: gp2
provisioner: kubernetes.io/aws-ebs
volumeBindingMode: WaitForFirstConsumer
```

Create IAM Role:

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster cloud-hw-eks \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn
arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
    --approve
```

Attach policy:

```
aws iam attach-role-policy --policy-arn
arn:aws:iam::533267106200:role/AmazonEKS_EBS_CSI_DriverRole --role-name
AmazonEKS_EBS_CSI_DriverRole
```

Create Amazon EBS CSI Driver Addon:

```
eksctl create addon --name aws-ebs-csi-driver --cluster cloud-hw-eks
--service-account-role-arn
arn:aws:iam::533267106200:role/AmazonEKS_EBS_CSI_DriverRole --force
```

Create PersistentVolumeClaim (Do not require PersistentVolume in Dynamic Provisioning)
kubectl create -f eks_ebspvc.yml

eks_ebspvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
  storageClassName: gp2
```

MongoDb_EKS_Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
```

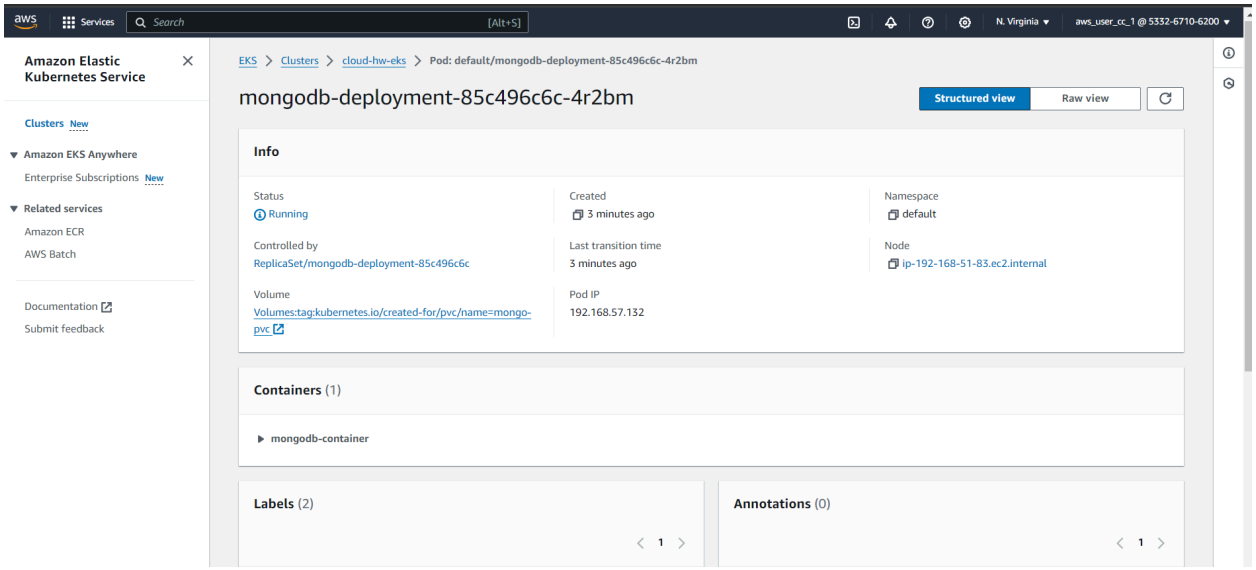
```
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb-container
        image: mongo:4.4.15
        ports:
        - containerPort: 27017
        env:
        - name: MONGODB_INITDB_ROOT_USERNAME
          value: admin
        - name: MONGODB_INITDB_ROOT_PASSWORD
          value: admin
        volumeMounts:
        - name: storage
          mountPath: /data/db
        readinessProbe:
          exec:
            command:
            - /bin/sh
            - -c
            - mongo --eval "db.adminCommand('ping')"
          initialDelaySeconds: 15
          periodSeconds: 10
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: mongo-pvc
```

Creating mongo service on AWS EKS:

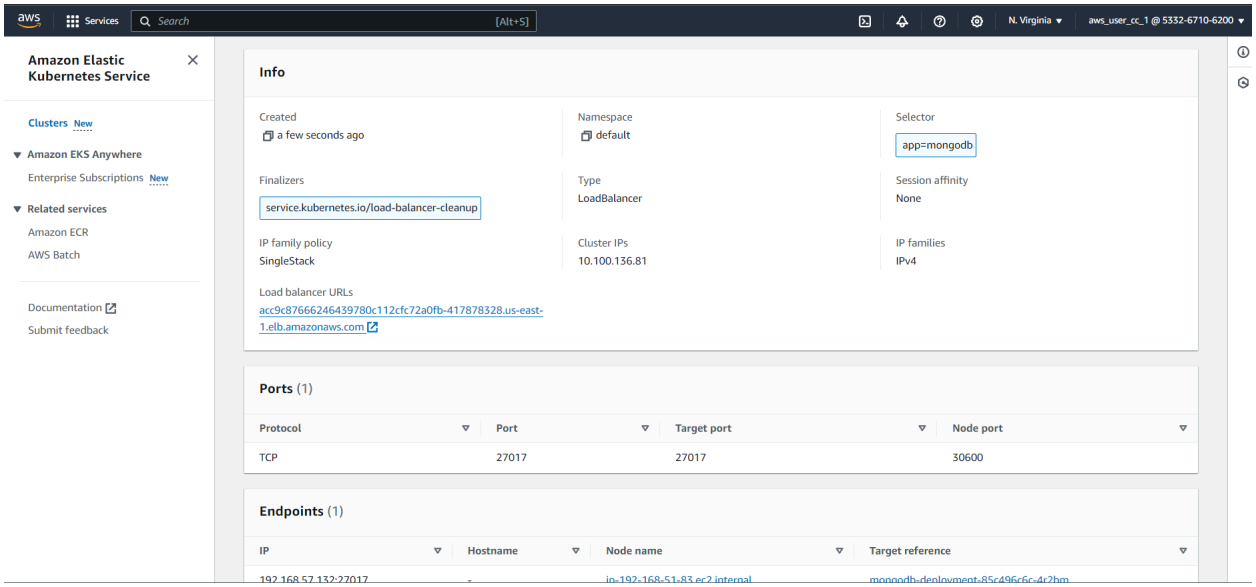
```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
```

```
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
  type: LoadBalancer
```

Status of mongo deployment on EKS:



Mongo service on EKS:



Observe the running mongo deployment and that mongo is accessed outside the cluster:

```
C:\Users\moink>curl http://acc9c87666246439780c112cfc72a0fb-417878328.us-east-1.elb.amazonaws.com:27017
It looks like you are trying to access MongoDB over HTTP on the native driver port.
```

```
C:\Users\moink\Downloads\CC\HW2\EKS>kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
mongodb-deployment-85c496c6c-4r2bm   1/1     Running   0          82s
```

Flask deployment on EKS:

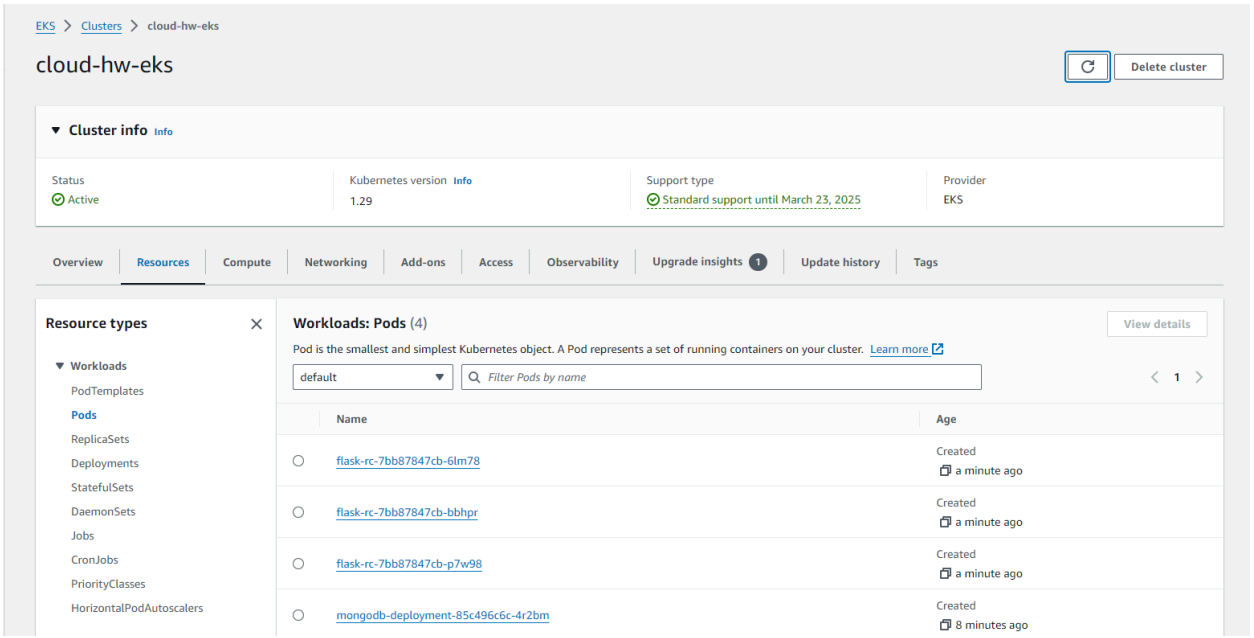
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-rc
spec:
  replicas: 3
  selector:
    matchLabels:
      app: flask
  template:
    metadata:
      name: flask
      labels:
        app: flask
    spec:
      containers:
      - name: flask-app-container
        image: mk8793/flask-docker-new:latest
        ports:
        - containerPort: 5000
        env:
          - name: MONGO_HOST
            value: mongodb-service
          - name: MONGO_PORT
            value: "27017"
          - name: FLASK_ENV
            value: development
          - name: PORT
            value: "5000"
        livenessProbe:
            httpGet:
```

```
            path: /
            port: 5000
          initialDelaySeconds: 10
          periodSeconds: 30
          failureThreshold: 3
      readinessProbe:
        exec:
          command:
          - /bin/sh
          - -c
          - curl --fail http://mongodb-service:27017/ || exit 1
        initialDelaySeconds: 10
        periodSeconds: 30
        failureThreshold: 4
```

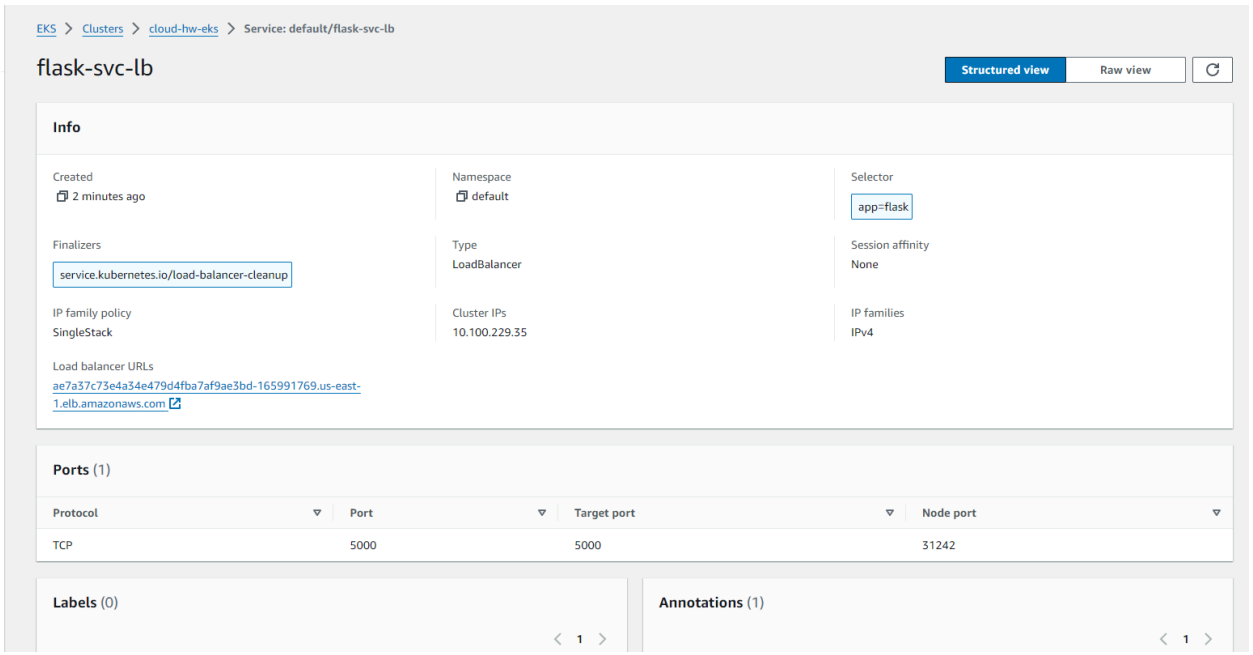
Creating Service for flask on EKS:

```
apiVersion: v1
kind: Service
metadata:
  name: flask-svc-lb
spec:
  selector:
    app: flask
  ports:
    - port: 5000
      targetPort: 5000
  type: LoadBalancer
```

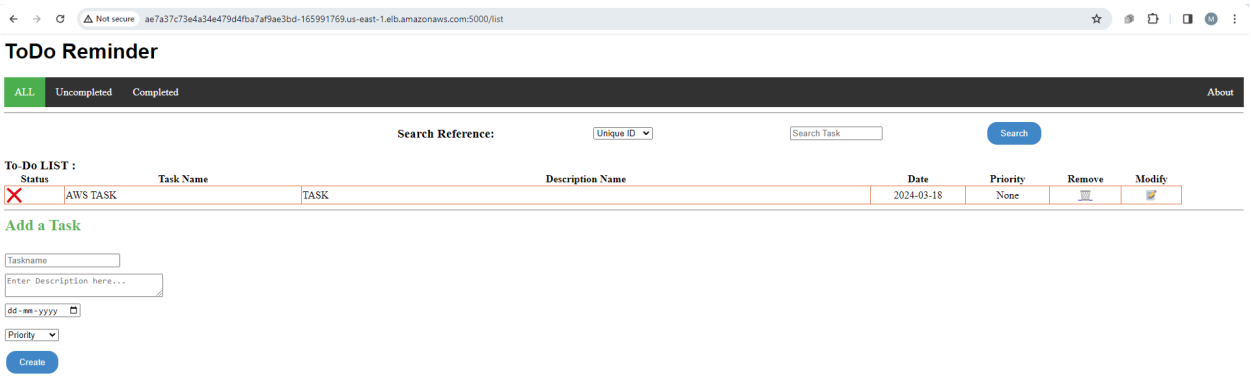
Check Pods on EKS - 3 replicas for flask and one pod for mongodb:



Observe the Flask Service on EKS and access it using load balancer url:



Access flask from outside:



Step 8: Alerting (Extra Credit 20 Points):

Deploy prometheus

Create namespace for prometheus:

```
kubectl create namespace monitoring
```

Create Cluster Role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
```

```
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring
```

Create config-map:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-conf
  labels:
    name: prometheus-server-conf
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
    - name: prometheus_alert
      rules:
      - alert: High Pod Memory
        expr: sum(container_memory_usage_bytes) > 1
        for: 1m
        labels:
          severity: slack
        annotations:
```

```
          summary: High Memory Usage
    - name: failing pods
      rules:
      - alert: PodFailingProbes
        expr:
sum(kube_pod_container_status_waiting_reason{reason=~"ProbeFailed|CrashLoo
pBackOff|ImagePullBackOff|InvalidImageName"}) by (namespace, pod) > 0
        for: 1m
        labels:
          severity: slack
        annotations:
          summary: Failing probs
    - name: pod-restart-alerts
      rules:
        - alert: PodRestartExceedsThreshold
          expr: increase(kube_pod_container_status_restarts_total[10m]) >
3
          for: 1m
          labels:
            severity: slack
          annotations:
            summary: "Pod {{ $labels.pod }} in namespace {{
$labels.namespace }} has restarted more than 3 times"
            description: "The pod {{ $labels.pod }} in namespace {{
$labels.namespace }} has restarted more than 3 times within the last 5
minutes."
  prometheus.yml: |-
    global:
      scrape_interval: 5s
      evaluation_interval: 5s
    rule_files:
      - /etc/prometheus/prometheus.rules
    alerting:
      alertmanagers:
      - scheme: http
        static_configs:
        - targets:
          - "alertmanager.monitoring.svc:9093"
    scrape_configs:
      - job_name: 'node-exporter'
```

```
        kubernetes_sd_configs:
          - role: endpoints
        relabel_configs:
        - source_labels: [__meta_kubernetes_endpoints_name]
          regex: 'node-exporter'
          action: keep
      - job_name: 'kubernetes-apiservers'
        kubernetes_sd_configs:
        - role: endpoints
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file:
/var/run/secrets/kubernetes.io/serviceaccount/token
        relabel_configs:
        - source_labels: [__meta_kubernetes_namespace,
__meta_kubernetes_service_name, __meta_kubernetes_endpoint_port_name]
          action: keep
          regex: default;kubernetes;https
      - job_name: 'kubernetes-nodes'
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file:
/var/run/secrets/kubernetes.io/serviceaccount/token
        kubernetes_sd_configs:
        - role: node
        relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)
        - target_label: __address__
          replacement: kubernetes.default.svc:443
        - source_labels: [__meta_kubernetes_node_name]
          regex: (.+)
          target_label: __metrics_path__
          replacement: /api/v1/nodes/${1}/proxy/metrics
      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
        - role: pod
        relabel_configs:
```

```
        - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
          action: replace
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
          target_label: __address__
        - action: labelmap
          regex: __meta_kubernetes_pod_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          action: replace
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_pod_name]
          action: replace
          target_label: kubernetes_pod_name
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets:
['kube-state-metrics.kube-system.svc.cluster.local:8080']
      - job_name: 'kubernetes-cadvisor'
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file:
/var/run/secrets/kubernetes.io/serviceaccount/token
        kubernetes_sd_configs:
        - role: node
        relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)
        - target_label: __address__
          replacement: kubernetes.default.svc:443
```

```
        - source_labels: [__meta_kubernetes_node_name]
          regex: (.+)
          target_label: __metrics_path__
          replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor
      - job_name: 'kubernetes-service-endpoints'
        kubernetes_sd_configs:
        - role: endpoints
        relabel_configs:
        - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scheme]
          action: replace
          target_label: __scheme__
          regex: (https?)
        - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__,
__meta_kubernetes_service_annotation_prometheus_io_port]
          action: replace
          target_label: __address__
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
        - action: labelmap
          regex: __meta_kubernetes_service_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          action: replace
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_service_name]
          action: replace
          target_label: kubernetes_name
```

Prometheus-deployment.yaml

```
apiVersion: apps/v1
```

```yaml
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
  labels:
    app: prometheus-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus/"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: prometheus-config-volume
              mountPath: /etc/prometheus/
            - name: prometheus-storage-volume
              mountPath: /prometheus/
      volumes:
        - name: prometheus-config-volume
          configMap:
            defaultMode: 420
            name: prometheus-server-conf

        - name: prometheus-storage-volume
          emptyDir: {}
```

Create prometheus deployment:

```
kubectl create  -f prometheus-deployment.yaml
```

prometheus_service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '9090'

spec:
  selector:
    app: prometheus-server
  type: NodePort
  ports:
    - port: 8080
      targetPort: 9090
      nodePort: 30000
```

Creating prometheus service:

```
kubectl create -f prometheus-service.yaml --namespace=monitoring
```

Run the following to access prometheus-service

```
minikube service prometheus-service --namespace=monitoring
```

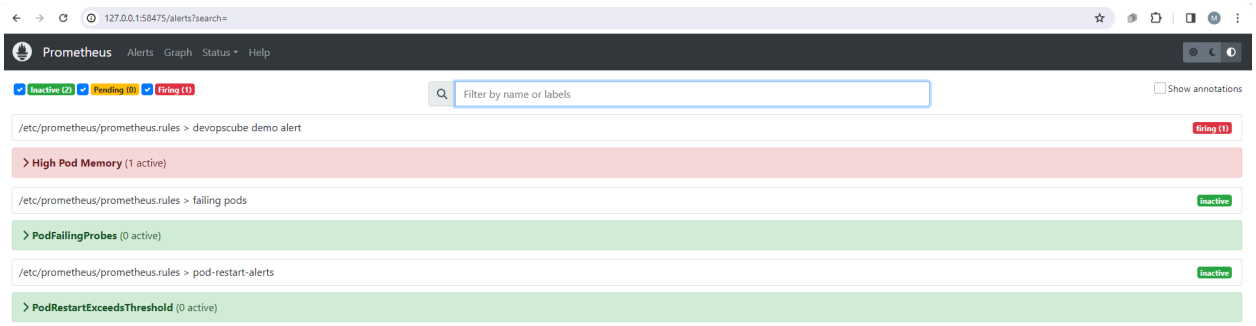
Rules for alerts:

1. High Pod Memory Alert:
   Alert Name: High Pod Memory

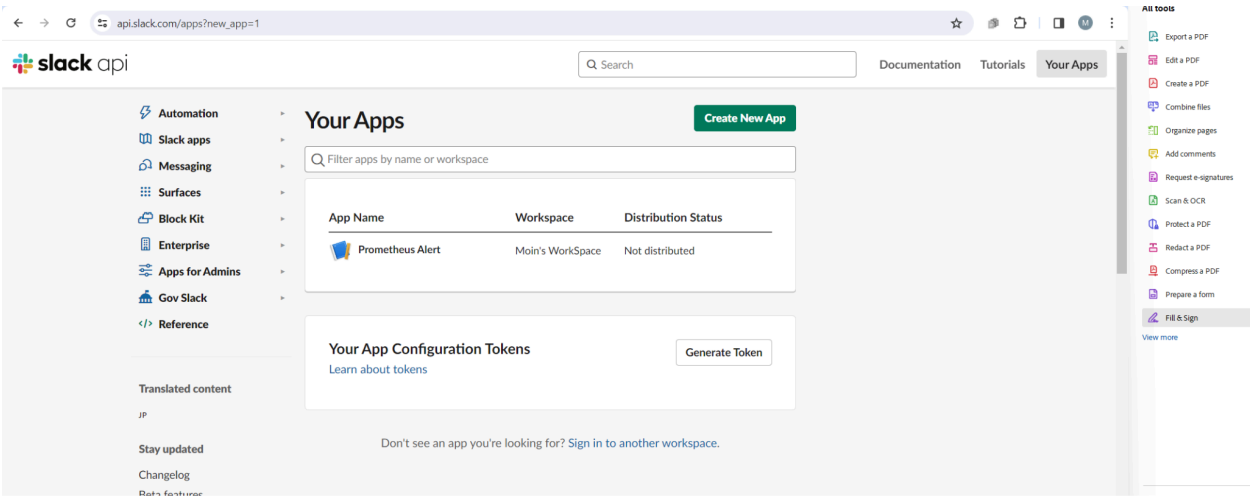
Expression (expr): sum(container_memory_usage_bytes) > 1
Condition: If the sum of container memory usage bytes exceeds 1 for 1 minute.
Labels: Sets severity label to "slack".
Annotations: Provides a summary message "High Memory Usage" for the alert.

2. Failing Pods Alert:
   Alert Name: PodFailingProbes
   Expression (expr):
   sum(kube_pod_container_status_waiting_reason{reason=~"ProbeFailed|CrashLoopBackOff|ImagePullBackOff|InvalidImageName"}) by (namespace, pod) > 0
   Condition: If there is at least one pod in a failing state (ProbeFailed, CrashLoopBackOff, ImagePullBackOff, InvalidImageName) within a namespace.
   Labels: Sets severity label to "slack".
   Annotations: Provides a summary message "Failing probes" for the alert.

3. Pod Restart Alerts:
   Alert Name: PodRestartExceedsThreshold
   Expression (expr): increase(kube_pod_container_status_restarts_total[10m]) > 3
   Condition: If the number of pod restarts exceeds 3 within the last 10 minutes.
   Labels: Sets severity label to "slack".
   Annotations: Provides a summary message "Pod {{ $labels.pod }} in namespace {{ $labels.namespace }} has restarted more than 3 times" and a description message "The pod {{ $labels.pod }} in namespace {{ $labels.namespace }} has restarted more than 3 times within the last 5 minutes."



Create Slack APP Webhook:



Alert Manager:
Add config of alertmanager to Prometheus config map

(we have already added in above Prometheus Config Map)

```
alerting:
   alertmanagers:
      - scheme: http
        static_configs:
        - targets:
          - "alertmanager.monitoring.svc:9093"
```

Crate config for alert manager:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: alertmanager-config
  namespace: monitoring
data:
  config.yml: |-
    global:
    templates:
    - '/etc/alertmanager/*.tmpl'
```

```
    route:
      receiver: slack_demo
      group_by: ['alertname', 'priority']
      group_wait: 10s
      repeat_interval: 30m
      routes:
        - receiver: slack_demo
        # Send severity=slack alerts to slack.
          match:
            severity: slack
          group_wait: 10s
          repeat_interval: 1m

    receivers:
    - name: slack_demo
      slack_configs:
      - api_url:
https://hooks.slack.com/services/T06PJA3QQUC/B06P98KGUA2/5EoPKYkvNen3m9ObL
SHt15Si
        channel: '#prometheus_alerts'
```

Here slack_configs stores has the configuration of slack service api url along with channel where the alerts will be posted

Create alert manager deployment:
alert_manger_deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alertmanager
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alertmanager
  template:
    metadata:
      name: alertmanager
      labels:
```

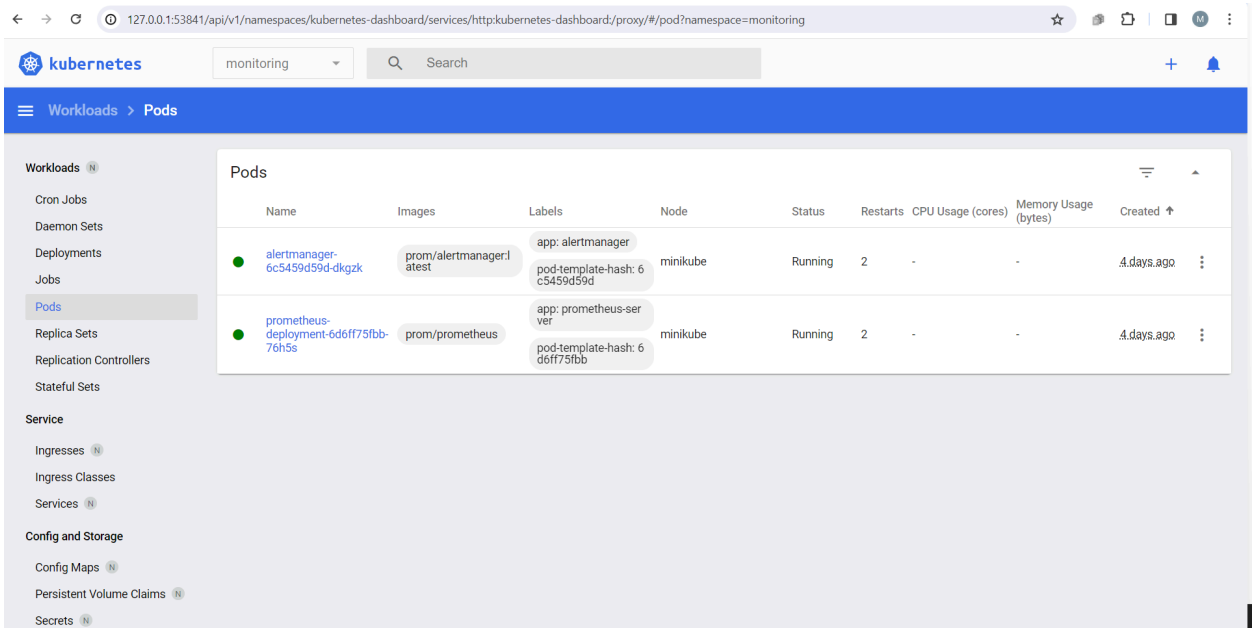
```
      app: alertmanager
  spec:
    containers:
    - name: alertmanager
      image: prom/alertmanager:latest
      args:
        - "--config.file=/etc/alertmanager/config.yml"
        - "--storage.path=/alertmanager"
      ports:
      - name: alertmanager
        containerPort: 9093
      resources:
          requests:
            cpu: 500m
            memory: 500M
          limits:
            cpu: 1
            memory: 1Gi
      volumeMounts:
      - name: config-volume
        mountPath: /etc/alertmanager
      - name: templates-volume
        mountPath: /etc/alertmanager-templates
      - name: alertmanager
        mountPath: /alertmanager
    volumes:
    - name: config-volume
      configMap:
        name: alertmanager-config
    - name: templates-volume
      configMap:
        name: alertmanager-templates
    - name: alertmanager
    emptyDir: {}
```

Create Service to access the alert manager:

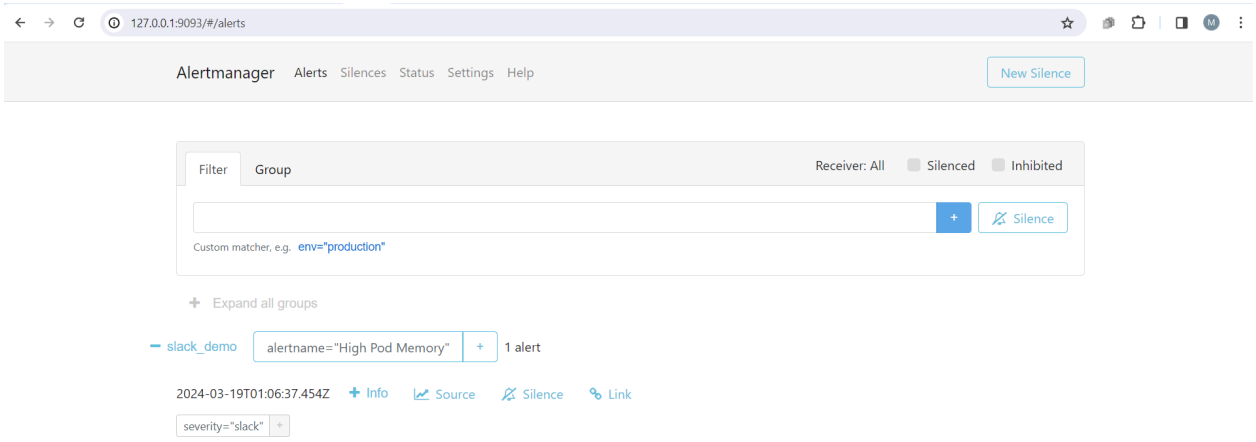
```
apiVersion: v1
kind: Service
```

```yaml
metadata:
  name: alertmanager
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '9093'
spec:
  selector:
    app: alertmanager
  type: NodePort
  ports:
    - port: 9093
      targetPort: 9093
      nodePort: 31000
```

Alert manager Pods can be observed below on our dashboard:



Alert Manager Dashboard:



Slack Notification: