NYU CS-GY 9223: Cloud Computing Spring 2025

Assignment 2

Name: Aaron Bengochea - ab6503
School: New York University – Tandon School of Engineering
Program: Masters of Science in Computer Science

Part 1 & 2: Deploying the Application on Minikube

1) We create the Dockerfile for the application

Dockerfile > ...

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt


# Expose the port that your Flask app runs on
EXPOSE 3000

# Define environment variable for Flask
ENV FLASK_APP=app.py

# Run the Flask application
CMD ["flask", "run", "--host=0.0.0.0", "--port=3000"]

```

2) We create the Docker-compose.yaml file using both the flask-app and mongodb images to create a new combined image.

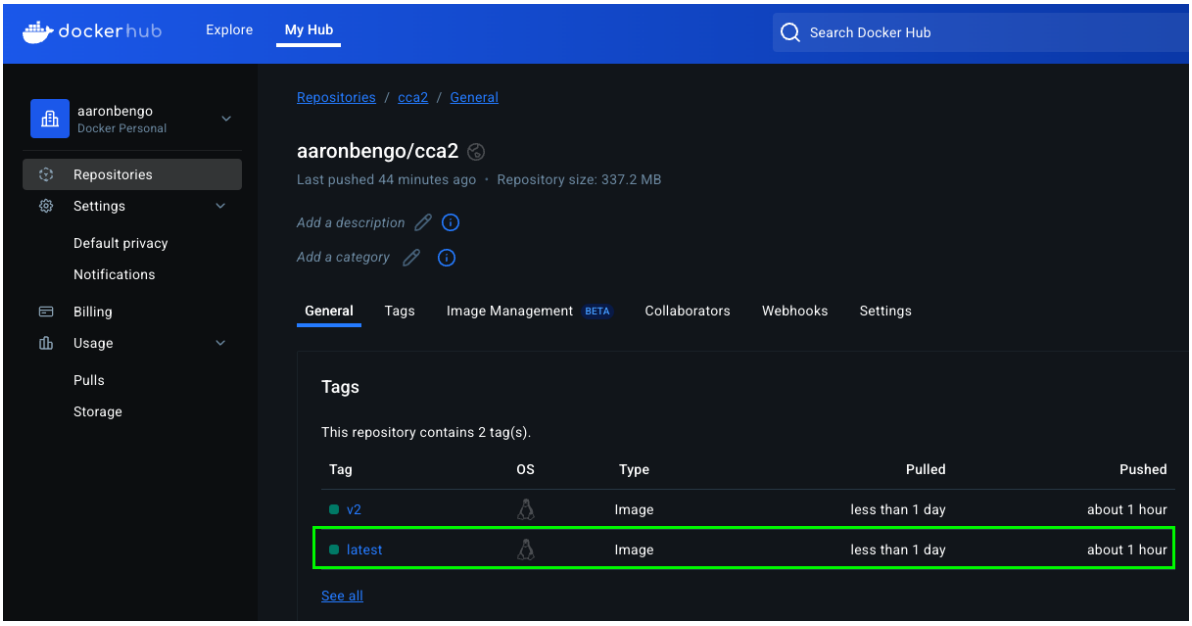
docker-compose.yml

```yaml
version: '3.8'

services:
  web:
    build: .
    image: aaronbengo/cca2:latest
    ports:
      - "3000:3000"
    depends_on:
      - mongo
    environment:
      - FLASK_ENV=development
      - MONGO_URI=mongodb://mongo:27017/todo_db
      - PORT=3000

  mongo:
    image: mongo:4.4
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db

volumes:
  mongo-data:
```

3) We push the finalized image to dockerhub using the following command in the CLI

```
docker buildx create --use docker buildx build --platform linux/amd64,linux/arm64 \
    -t aaronbengo/cca2:latest --push .
```

4) We observe that the image was successfully pushed to DockerHub with the name "aaronbengo/cca2:latest"



Part 3: Deploying the Application on Minikube

1) Apply the deployment and services to minikube

```
→  cca2 git:(main) ✗ kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
kubectl apply -f mongodb-deployment.yaml
kubectl apply -f mongodb-service.yaml

deployment.apps/flask-app-deployment created
service/flask-app-service created
error: the path "mongodb-deployment.yaml" does not exist
error: the path "mongodb-service.yaml" does not exist
→  cca2 git:(main) ✗ kubectl apply -f mongo-deployment.yaml
kubectl apply -f mongo-service.yaml

deployment.apps/mongo-deployment created
service/mongo created
```

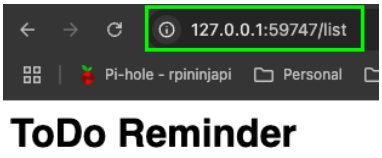
2) We check deployments, services, and pods in order to verify that our deployment is working as expected. We find that our services our running as expected. We are currently running two replicas of our flask application. We can observe that there are two replicas in the "kubectl get deployments" command, and we can observe their pod names and their respective status in "kubectl get pods".

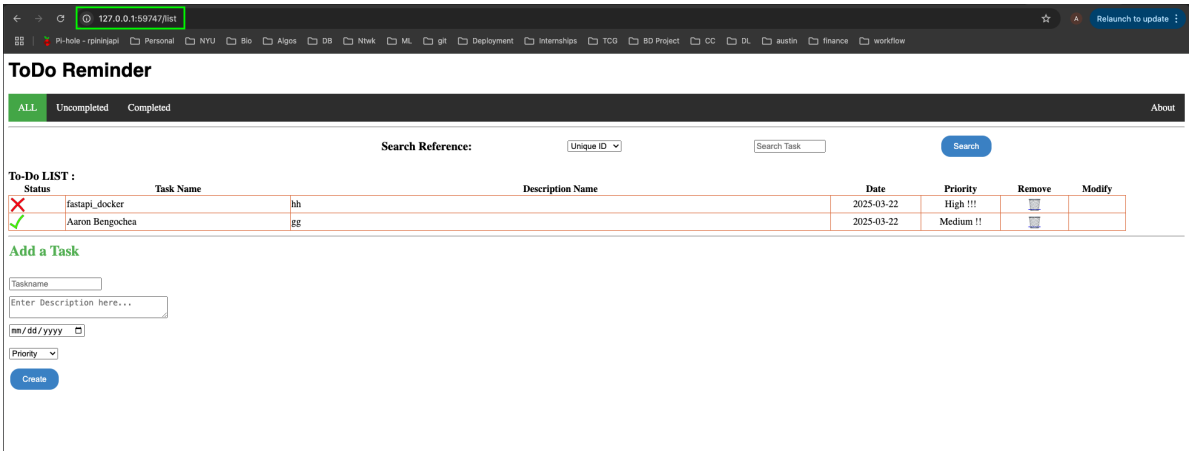
```
→  cca2 git:(main) ✗ kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
flask-app-deployment   2/2     2            2           118s
mongo-deployment       1/1     1            1           60s
→  cca2 git:(main) ✗ kubectl get services
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
flask-app-service   NodePort    10.110.183.94   <none>        3000:30001/TCP    2m11s
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP           55m
mongo               ClusterIP   10.103.83.253   <none>        27017/TCP         73s
→  cca2 git:(main) ✗ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
flask-app-deployment-b8cdcf6fb-dhpng   1/1     Running   0          2m46s
flask-app-deployment-b8cdcf6fb-rjvhp   1/1     Running   0          2m46s
mongo-deployment-cb654f8f-t84jl        1/1     Running   0          108s
```

3) We run "minikube service flask-app-service" in order have minikube mount the IP and ports for local access. We are successfully given a URL with our live local deployment.

```
➜  cca2 git:(main) ✗ minikube service flask-app-service
|-----------|-------------------|-------------|---------------------------|
| NAMESPACE |       NAME        | TARGET PORT |            URL            |
|-----------|-------------------|-------------|---------------------------|
| default   | flask-app-service |        3000 | http://192.168.49.2:30001 |
|-----------|-------------------|-------------|---------------------------|
🏃  Starting tunnel for service flask-app-service.
|-----------|-------------------|-------------|------------------------|
| NAMESPACE |       NAME        | TARGET PORT |          URL           |
|-----------|-------------------|-------------|------------------------|
| default   | flask-app-service |             | http://127.0.0.1:59747 |
|-----------|-------------------|-------------|------------------------|
🎉  Opening service default/flask-app-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

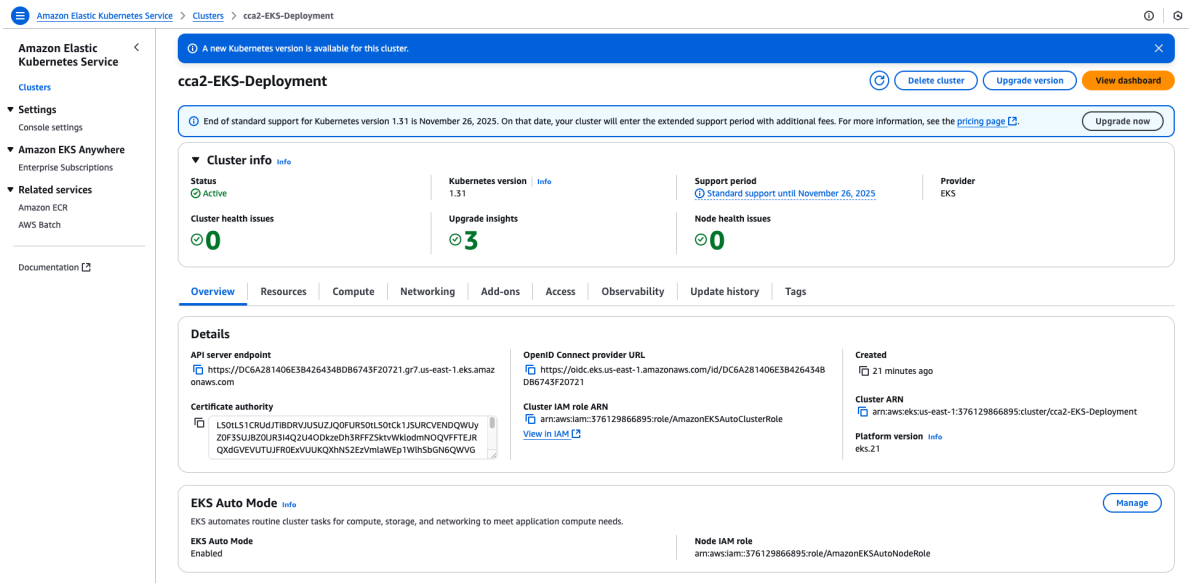
4) Finally, we navigate to the given URL and find that our application is deployed and working as expected.





Part 4: Deploying the Application on AWS EKS

1) We create an AWS EKS Cluster via the AWS Management Console



2) We then add the AWS EKS cluster to our kubeconfig

```
aws eks --region us-east-1 update-kubeconfig --cca2-EKS-Deployment
```

3) We switch context from minikube to AWS EKS

```
kubectl config use-context AWS EKS
```

4) We apply deployment and services to AWS EKS Cluster

```
kubectl apply -f flask-eks-deployment.yaml
kubectl apply -f flask-eks-service.yaml
kubectl apply -f mongo-eks-deployment.yaml
kubectl apply -f mongo-eks-deployment.yaml
```

5) Use helm to setup AWS Load Balancer Controller

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
-n kube-system \
--set clusterName=cca2-EKS-Deployment \
--set serviceAccount.create=true \
--set serviceAccount.name=aws-load-balancer-controller \
--set region=us-east-1 \
--set vpcId=vpc-0fd462f3add9daa4b \
--set serviceAccount.annotations."eks\.amazonaws\.com/role arn"
    ="arn:aws:iam::376129866895:role/EKSLoadBalancerControllerRole" \
--set replicaCount=2
```

Note that the EKSLoadBalancerControllerRole, has a AWSLoadBalancerControllerIAMPolicy attached to it while also having a trust relationship configured with our AWS EKS cluster.

AWS Load Balancer Controller IAM Policy:

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:CreateServiceLinkedRole"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeAccountAttributes",
            "ec2:DescribeAddresses",
            "ec2:DescribeAvailabilityZones",
            "ec2:DescribeInternetGateways",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcPeeringConnections",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeInstances",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeTags",
```

```
            "ec2:GetCoipPoolUsage",
            "ec2:DescribeCoipPools",
            "ec2:GetSecurityGroupsForVpc",
            "ec2:DescribeIpamPools",
            "elasticloadbalancing:DescribeLoadBalancers",
            "elasticloadbalancing:DescribeLoadBalancerAttributes",
            "elasticloadbalancing:DescribeListeners",
            "elasticloadbalancing:DescribeListenerCertificates",
            "elasticloadbalancing:DescribeSSLPolicies",
            "elasticloadbalancing:DescribeRules",
            "elasticloadbalancing:DescribeTargetGroups",
            "elasticloadbalancing:DescribeTargetGroupAttributes",
            "elasticloadbalancing:DescribeTargetHealth",
            "elasticloadbalancing:DescribeTags",
            "elasticloadbalancing:DescribeTrustStores",
            "elasticloadbalancing:DescribeListenerAttributes",
            "elasticloadbalancing:DescribeCapacityReservation"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "cognito-idp:DescribeUserPoolClient",
            "acm:ListCertificates",
            "acm:DescribeCertificate",
            "iam:ListServerCertificates",
            "iam:GetServerCertificate",
            "waf-regional:GetWebACL",
            "waf-regional:GetWebACLForResource",
            "waf-regional:AssociateWebACL",
            "waf-regional:DisassociateWebACL",
            "wafv2:GetWebACL",
            "wafv2:GetWebACLForResource",
            "wafv2:AssociateWebACL",
            "wafv2:DisassociateWebACL",
            "shield:GetSubscriptionState",
            "shield:DescribeProtection",
            "shield:CreateProtection",
            "shield:DeleteProtection"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:AuthorizeSecurityGroupIngress",
```

```
            "ec2:RevokeSecurityGroupIngress"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateSecurityGroup"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags"
        ],
        "Resource": "arn:aws:ec2:*:*:security-group/*",
        "Condition": {
            "StringEquals": {
                "ec2:CreateAction": "CreateSecurityGroup"
            },
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags",
            "ec2:DeleteTags"
        ],
        "Resource": "arn:aws:ec2:*:*:security-group/*",
        "Condition": {
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster": "true",
                "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:RevokeSecurityGroupIngress",
            "ec2:DeleteSecurityGroup"
        ],
```

```
        "Resource": "*",
        "Condition": {
            "Null": {
                "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateLoadBalancer",
            "elasticloadbalancing:CreateTargetGroup"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateListener",
            "elasticloadbalancing:DeleteListener",
            "elasticloadbalancing:CreateRule",
            "elasticloadbalancing:DeleteRule"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:AddTags",
            "elasticloadbalancing:RemoveTags"
        ],
        "Resource": [
            "arn:aws:elasticloadbalancing:*:*:targetgroup/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalancer/net/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalancer/app/*/*"
        ],
        "Condition": {
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster": "true",
                "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
            }
        }
```

```
},
{
    "Effect": "Allow",
    "Action": [
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:RemoveTags"
    ],
    "Resource": [
        "arn:aws:elasticloadbalancing:*:*:listener/net/*/*/*",
        "arn:aws:elasticloadbalancing:*:*:listener/app/*/*/*",
        "arn:aws:elasticloadbalancing:*:*:listener-rule/net/*/*/*",
        "arn:aws:elasticloadbalancing:*:*:listener-rule/app/*/*/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:SetIpAddressType",
        "elasticloadbalancing:SetSecurityGroups",
        "elasticloadbalancing:SetSubnets",
        "elasticloadbalancing:DeleteLoadBalancer",
        "elasticloadbalancing:ModifyTargetGroup",
        "elasticloadbalancing:ModifyTargetGroupAttributes",
        "elasticloadbalancing:DeleteTargetGroup",
        "elasticloadbalancing:ModifyListenerAttributes",
        "elasticloadbalancing:ModifyCapacityReservation",
        "elasticloadbalancing:ModifyIpPools"
    ],
    "Resource": "*",
    "Condition": {
        "Null": {
            "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "elasticloadbalancing:AddTags"
    ],
    "Resource": [
        "arn:aws:elasticloadbalancing:*:*:targetgroup/*/*",
        "arn:aws:elasticloadbalancing:*:*:loadbalancer/net/*/*",
        "arn:aws:elasticloadbalancing:*:*:loadbalancer/app/*/*"
    ],
    "Condition": {
```

```
                "StringEquals": {
                    "elasticloadbalancing:CreateAction": [
                        "CreateTargetGroup",
                        "CreateLoadBalancer"
                    ]
                },
                "Null": {
                    "aws:RequestTag/elbv2.k8s.aws/cluster": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:RegisterTargets",
                "elasticloadbalancing:DeregisterTargets"
            ],
            "Resource": "arn:aws:elasticloadbalancing:*:*:targetgroup/*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:SetWebAcl",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:AddListenerCertificates",
                "elasticloadbalancing:RemoveListenerCertificates",
                "elasticloadbalancing:ModifyRule",
                "elasticloadbalancing:SetRulePriorities"
            ],
            "Resource": "*"
        }
    ]
}
```

Trust Policy:

```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Federated":
                    ↪  "arn:aws:iam::376129866895:oidc-provider/oidc.eks.us-eas
                    ↪  t-1.amazonaws.com/id/DC6A281406E3B426434BDB6743F20721"
                },
                "Action": "sts:AssumeRoleWithWebIdentity",
```

```
                "Condition": {
                    "StringEquals": {
                        "oidc.eks.us-east-1.amazonaws.com/id/DC6A281406E3B426434
                        ↪ BDB6743F20721:aud":
                        ↪ "sts.amazonaws.com"
                    }
                }
            }
        ]
    }
```

6) Tag one of the VPC Subnets being used by the AWS EKS Cluster so that it may be eligible for load balancer provisioning

Key: kubernetes.io/cluster/cca2-EKS-Deployment
Value: shared

Key: kubernetes.io/role/internal-elb
Value: 1

7) Check to see if the Load Balancer pods are running as expected

```
→  cca2 git:(main) ✗ kubectl get pods -n kube-system | grep aws-load-balancer-controller

aws-load-balancer-controller-8dd54586c-htjqp   1/1     Running   0          6h32m
aws-load-balancer-controller-8dd54586c-m9hbr   1/1     Running   0          6h32m
```

8) Check to see if deployment, services, and pods are running as expected

```
→  cca2 git:(main) ✗ kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
flask-app-deployment   2/2     2            2           8h
mongo-deployment       1/1     1            1           8h
```

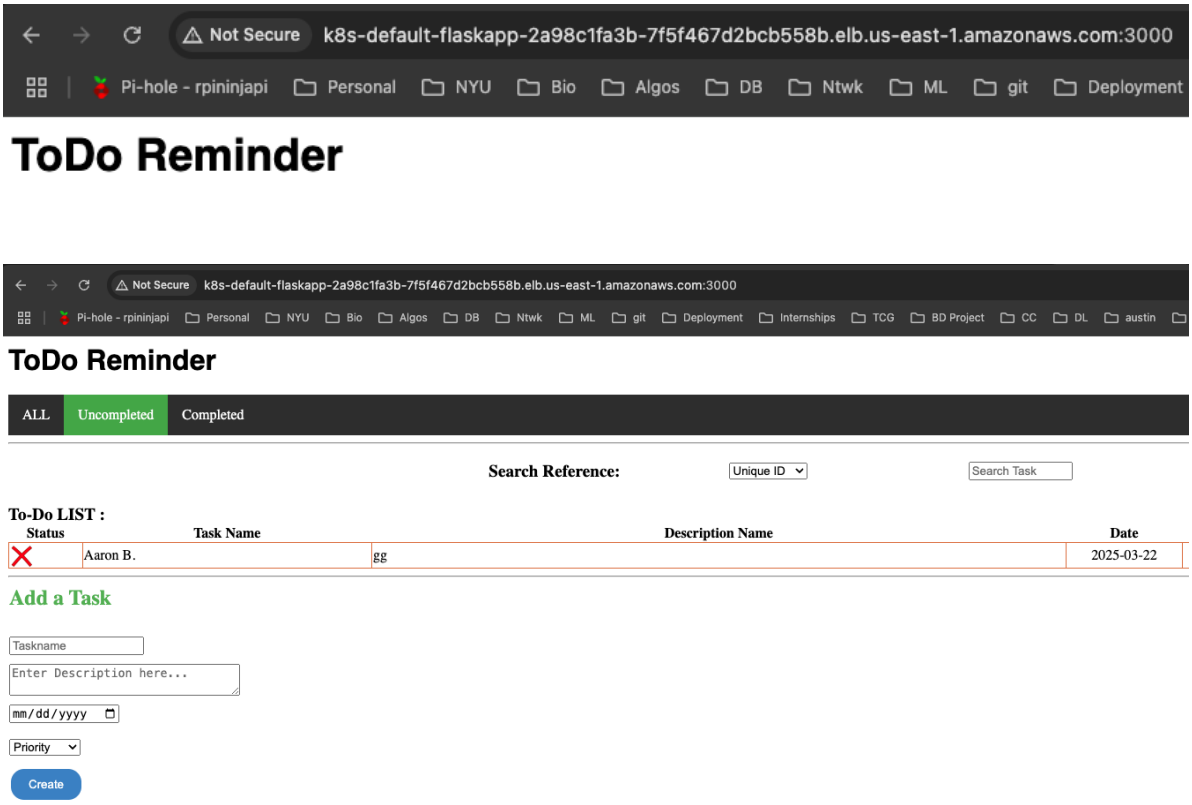
```
→  cca2 git:(main) ✗ kubectl get pods
NAME                                READY   STATUS    RESTARTS      AGE
flask-app-deployment-8698687d8d-k6hj8   1/1     Running   10 (8h ago)   8h
flask-app-deployment-8698687d8d-wl8lg   1/1     Running   10 (8h ago)   8h
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0             8h
```

```
→  cca2 git:(main) ✗ kubectl get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP                                                                  PORT(S)          AGE
flask-app-service   LoadBalancer   10.100.84.34    k8s-default-flaskapp-2a98c1fa3b-7f5f467d2bcb558b.elb.us-east-1.amazonaws.com   3000:32558/TCP   31m
kubernetes        ClusterIP      10.100.0.1      <none>                                                                       443/TCP          9h
mongo             ClusterIP      10.100.157.48   <none>                                                                       27017/TCP        8h
```

We can see that our services are all successfully deployed, our flask-app-service is assigned an external IP.

External IP:
http://k8s-default-flaskapp-2a98c1fa3b-7f5f467d2bcb558b.elb.us-east-1.amazonaws.com:3000/

9) We observe that our application is being deployed as expected on AWS EKS



10) We show a final summary of the flask-app-service

```
➜  cca2 git:(main) ✗ kubectl describe service flask-app-service
Name:                     flask-app-service
Namespace:                default
Labels:                   <none>
Annotations:              service.beta.kubernetes.io/aws-load-balancer-internal: false
Selector:                 app=flask-app
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.100.84.34
IPs:                      10.100.84.34
LoadBalancer Ingress:     k8s-default-flaskapp-2a98c1fa3b-7f5f467d2bcb558b.elb.us-east-1.amazonaws.com
Port:                     <unset>  3000/TCP
TargetPort:               3000/TCP
NodePort:                 <unset>  32558/TCP
Endpoints:                172.31.94.32:3000,172.31.94.33:3000
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:
  Type    Reason                Age   From     Message
  ----    ------                ----  ----     -------
  Normal  SuccessfullyReconciled  33m   service  Successfully reconciled
```

Part 5: Building the Replication Controller

1) We create a flask-rc.yaml file with the specifications of our replication controller, we will be starting with 2 replicas. We apply the replication controller yaml file using kubectl. Note, we are using "kind: Deployment" in our flask-eks-deployment.yaml file. The "Deployment" kind is a modern version of ReplicationController, with added features, in order to demonstrate that the replication controller is working as expected, we set the replica count in the flask-eks-deployment.yaml file to 0. Please reference the screenshots of the pods in sections 3 and 4 for additional context.

```
→  cca2 git:(main) ✗ kubectl apply -f flask-eks-deployment.yaml

deployment.apps/flask-app-deployment configured
→  cca2 git:(main) ✗ kubectl apply -f flask-rc.yaml

replicationcontroller/flask-app-rc unchanged
```

We print a summary of the pods after applying the changes in order to verify that the pod amount created matches our replica controllers replica count, we observe that this is true. The pod highlighted in green is to be deleted in the next step.

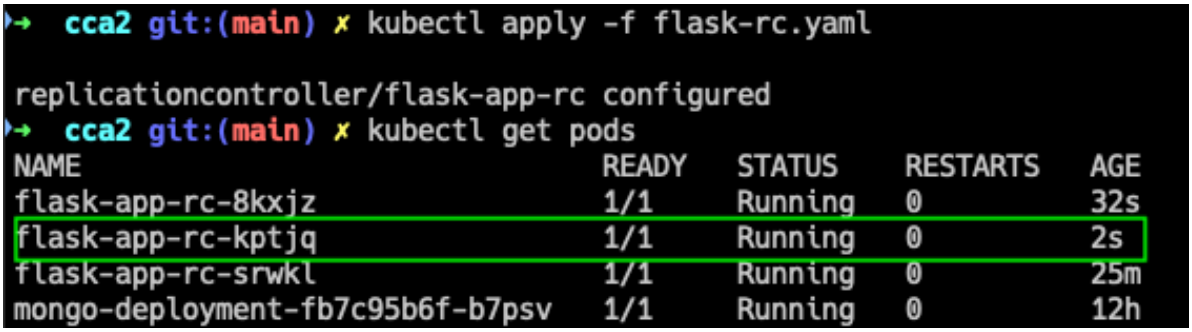
```
→  cca2 git:(main) ✗ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-app-rc-jz4qc                  1/1     Running   0          23m
flask-app-rc-srwkl                  1/1     Running   0          23m
mongo-deployment-fb7c95b6f-b7psv    1/1     Running   0          12h
```

2) We manually delete a pod in order to test our replica controller's response, the expected response is that a new flask-app pod will be created in place of the deleted flask-app pod. We observe that we obtain the expected response, a new flask-app pod is created.

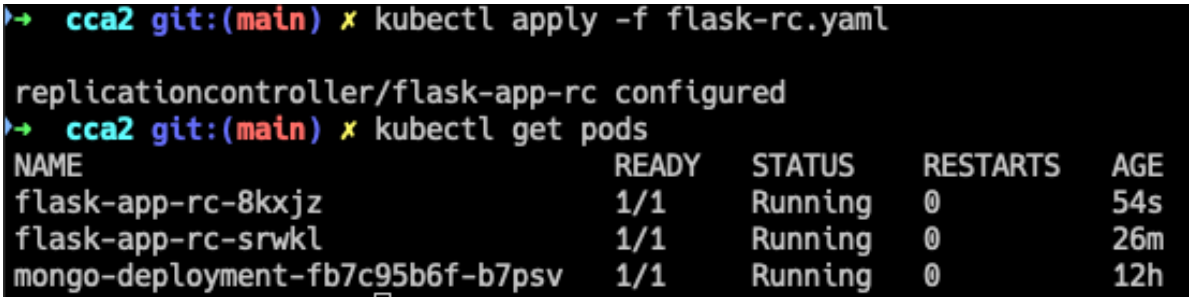
```
→  cca2 git:(main) ✗ kubectl delete pod flask-app-rc-jz4qc
pod "flask-app-rc-jz4qc" deleted
```

```
→  cca2 git:(main) ✗ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-app-rc-8kxjz                  1/1     Running   0          5s
flask-app-rc-srwkl                  1/1     Running   0          25m
mongo-deployment-fb7c95b6f-b7psv    1/1     Running   0          12h
```

3) We now scale up the replica count in our replica controller and apply the changes to our cluster



4) Finally, we scale our replica count in our replica controller back to the original 2 replicas, in order to demonstrate de-scaling. The application is still working and accessible as expected.



Part 6: Applying a Rolling Update Strategy

1) We update our flask-eks-deployment.yaml file to include Strategy: RollingUpdate

flask-eks-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1 # Allows up to x additional pods above desired replica count at time of update
      maxUnavailable: 0 # Max unavailable pods during update
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: aaronbengo/cca2:v2
        ports:
        - containerPort: 3000
        env:
        - name: FLASK_ENV
          value: "development"
        - name: MONGO_URI
          value: "mongodb://mongo:27017/todo_db"
        - name: PORT
          value: "3000"

```

2) We build and push an updated docker image named aaronbengo/cca2:v2

```
→  cca2 git:(main) ✗ docker buildx create --use
docker buildx build --platform linux/amd64,linux/arm64 \
  -t aaronbengo/cca2:v2 --push .

brave_kalam
[+] Building 30.7s (17/17) FINISHED
```

3) We check pods for status, apply the changes to our cluster, and recheck the status of the pods. We find that the pods are updated successfully.

```
→  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-8698687d8d-m4cmm   1/1     Running   0          13m
flask-app-deployment-8698687d8d-xpv5k   1/1     Running   0          13m
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          19h
→  cca2 git:(main) ✗ kubectl apply -f flask-eks-deployment.yaml
deployment.apps/flask-app-deployment configured
→  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-6544c6b4c5-n8j6w   1/1     Running   0          5s
flask-app-deployment-6544c6b4c5-sp4j5   1/1     Running   0          9s
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          19h
→  cca2 git:(main) ✗ kubectl rollout status deployment/flask-app-deployment

deployment "flask-app-deployment" successfully rolled out
```

4) We check the details of one of the new flask-app-deployment pods in order to verify the new image and check logs. We find that we indeed pulled and successfully updated the pods in the Events section.

```
→  cca2 git:(main) ✗ kubectl describe pod flask-app-deployment-6544c6b4c5-n8j6w
Name:             flask-app-deployment-6544c6b4c5-n8j6w
Namespace:        default
Priority:         0
Service Account:  default
Node:             i-01dbade56c868046b/172.31.81.203
Start Time:       Sun, 23 Mar 2025 08:04:44 -0400
Labels:           app=flask-app
                  pod-template-hash=6544c6b4c5
Annotations:      <none>
Status:           Running
IP:               172.31.94.35
IPs:
  IP:           172.31.94.35
Controlled By:  ReplicaSet/flask-app-deployment-6544c6b4c5
Containers:
  flask-app:
    Container ID:   containerd://dd75ce712357bbab0ea80f3778576d1f77425ccab2b3056c4c3058c7e6e46d9b
    Image:          aaronbengo/cca2:v2
    Image ID:       docker.io/aaronbengo/cca2@sha256:7eea4bd8fc1e08be214db50469adc84b88b0cb0f2aecf608e454e607b724088d
    Port:           3000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 23 Mar 2025 08:04:46 -0400
    Ready:          True
    Restart Count:  0
    Environment:
      FLASK_ENV:  development
      MONGO_URI:  mongodb://mongo:27017/todo_db
      PORT:       3000
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-btwgj (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-btwgj:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  7m1s   default-scheduler  Successfully assigned default/flask-app-deployment-6544c6b4c5-n8j6w to i-01dbade56c868046b
  Normal  Pulling    7m     kubelet            Pulling image "aaronbengo/cca2:v2"
  Normal  Pulled     6m59s  kubelet            Successfully pulled image "aaronbengo/cca2:v2" in 701ms (701ms including waiting). Image size: 160395139 bytes.
  Normal  Created    6m59s  kubelet            Created container flask-app
  Normal  Started    6m59s  kubelet            Started container flask-app
```

5) We check the details of the flask deployment in order to verify that the updates occured and that the new image version reflects the changes. We find that the updates were indeed successful

and that our deployment uses the new image named "aaronbengo/cca2:v2".

```
→  cca2 git:(main) ✗ kubectl describe deployment flask-app-deployment
Name:                   flask-app-deployment
Namespace:              default
CreationTimestamp:      Sat, 22 Mar 2025 12:44:37 -0400
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=flask-app
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  0 max unavailable, 1 max surge
Pod Template:
  Labels:  app=flask-app
  Containers:
   flask-app:
    Image:      aaronbengo/cca2:v2
    Port:       3000/TCP
    Host Port:  0/TCP
    Environment:
      FLASK_ENV:   development
      MONGO_URI:   mongodb://mongo:27017/todo_db
      PORT:        3000
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  flask-app-deployment-8698687d8d (0/0 replicas created)
NewReplicaSet:   flask-app-deployment-6544c6b4c5 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  38m   deployment-controller  Scaled up replica set flask-app-deployment-8698687d8d to 2 from 0
  Normal  ScalingReplicaSet  24m   deployment-controller  Scaled up replica set flask-app-deployment-6544c6b4c5 to 1
  Normal  ScalingReplicaSet  24m   deployment-controller  Scaled down replica set flask-app-deployment-8698687d8d to 1 from 2
  Normal  ScalingReplicaSet  24m   deployment-controller  Scaled up replica set flask-app-deployment-6544c6b4c5 to 2 from 1
  Normal  ScalingReplicaSet  24m   deployment-controller  Scaled down replica set flask-app-deployment-8698687d8d to 0 from 1
```

Part 7: Applying a Health Monitoring Strategy

1) We add an API endpoint in order to allow for health checks

```
@app.route("/health")
def hello ():
    return "OK", 200
```

2) We build and push the new image named "aaronbengo/cca2:v3"

```
→  cca2 git:(main) ✗ docker buildx create --use
docker buildx build --platform linux/amd64,linux/arm64 \
  -t aaronbengo/cca2:v3 --push .

elated_hamilton
[+] Building 28.8s (17/17) FINISHED
```

3) We update our flask-eks-development.yaml file to include the livenessProbe and readinessProbe. Our probes check our app's health by performing an HTTP get to our /health endpoint.

flask-eks-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1 # Allows up to x additional pods above desired replica count at time of update
      maxUnavailable: 0 # Max unavailable pods during update
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: aaronbengo/cca2:v3
        ports:
        - containerPort: 3000
        env:
        - name: FLASK_ENV
          value: "development"
        - name: MONGO_URI
          value: "mongodb://mongo:27017/todo_db"
        - name: PORT
          value: "3000"
        livenessProbe:
          httpGet:
            path: /health # Uses HTTP get to check health status of app
            port: 3000
          initialDelaySeconds: 10 # Performs the first check after 10s have passed from container creation
          periodSeconds: 10 # Checks every 10 seconds
          failureThreshold: 3 # Amount of consecutive failures before k8 restarts the pod
        readinessProbe:
          httpGet:
            path: /health # Uses HTTP get to check health status of app
            port: 3000
          initialDelaySeconds: 5 # Performs the first check after 5s have passed from container creation
          periodSeconds: 5 # Checks every 5 seconds
          failureThreshold: 2 # Amount of consecutive failures before k8 restarts the pod

```

4) We apply the changes to our cluster and we observe that new pods are spun up and used to replace the old pods.

```
➜  cca2 git:(main) ✗ kubectl apply -f flask-eks-deployment.yaml
deployment.apps/flask-app-deployment configured
➜  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-6544c6b4c5-n8j6w   1/1     Running   0          88m
flask-app-deployment-6544c6b4c5-sp4j5   1/1     Running   0          88m
flask-app-deployment-6cdfb5b4c5-fnr2x   0/1     Running   0          6s
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          20h
➜  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-6544c6b4c5-sp4j5   1/1     Running   0          89m
flask-app-deployment-6cdfb5b4c5-dqdrd   0/1     Running   0          8s
flask-app-deployment-6cdfb5b4c5-fnr2x   1/1     Running   0          19s
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          20h
➜  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-6cdfb5b4c5-dqdrd   1/1     Running   0          21s
flask-app-deployment-6cdfb5b4c5-fnr2x   1/1     Running   0          32s
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          20h
➜  cca2 git:(main) ✗ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
flask-app-deployment-6cdfb5b4c5-dqdrd   1/1     Running   0          38s
flask-app-deployment-6cdfb5b4c5-fnr2x   1/1     Running   0          49s
mongo-deployment-fb7c95b6f-b7psv        1/1     Running   0          20h
```

5) We check the description of the new pods for Liveness and Readiness metadata fields, we confirm that the new pods are using the appropriate image.

```
→  cca2 git:(main) ✗ kubectl describe pod flask-app-deployment-6cdfb5b4c5-dqdrd
Name:             flask-app-deployment-6cdfb5b4c5-dqdrd
Namespace:        default
Priority:         0
Service Account:  default
Node:             i-01dbade56c868046b/172.31.81.203
Start Time:       Sun, 23 Mar 2025 09:33:40 -0400
Labels:           app=flask-app
                  pod-template-hash=6cdfb5b4c5
Annotations:      <none>
Status:           Running
IP:               172.31.94.33
IPs:
  IP:           172.31.94.33
Controlled By:  ReplicaSet/flask-app-deployment-6cdfb5b4c5
Containers:
  flask-app:
    Container ID:   containerd://ebc2a482dba079ebe56ff8b92e5617df297c9a2bc9c7f8e44be79e8020e8b4f1
    Image:          aaronbengo/cca2:v3
    Image ID:       docker.io/aaronbengo/cca2@sha256:d773af01ec73aaae4a7de51a65c1c4511b4a621284fd06df983a09fe70e16a03
    Port:           3000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 23 Mar 2025 09:33:41 -0400
    Ready:          True
    Restart Count:  0
    Liveness:       http-get http://:3000/health delay=10s timeout=1s period=10s #success=1 #failure=3
    Readiness:      http-get http://:3000/health delay=5s timeout=1s period=5s #success=1 #failure=2
    Environment:
      FLASK_ENV:  development
      MONGO_URI:  mongodb://mongo:27017/todo_db
      PORT:       3000
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-l7z6l (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-l7z6l:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  72s   default-scheduler  Successfully assigned default/flask-app-deployment-6cdfb5b4c5-dqdrd to i-01dbade56c868046b
  Normal  Pulling    72s   kubelet            Pulling image "aaronbengo/cca2:v3"
  Normal  Pulled     71s   kubelet            Successfully pulled image "aaronbengo/cca2:v3" in 118ms (118ms including waiting). Image size: 160395290 bytes.
  Normal  Created    71s   kubelet            Created container flask-app
  Normal  Started    71s   kubelet            Started container flask-app
```

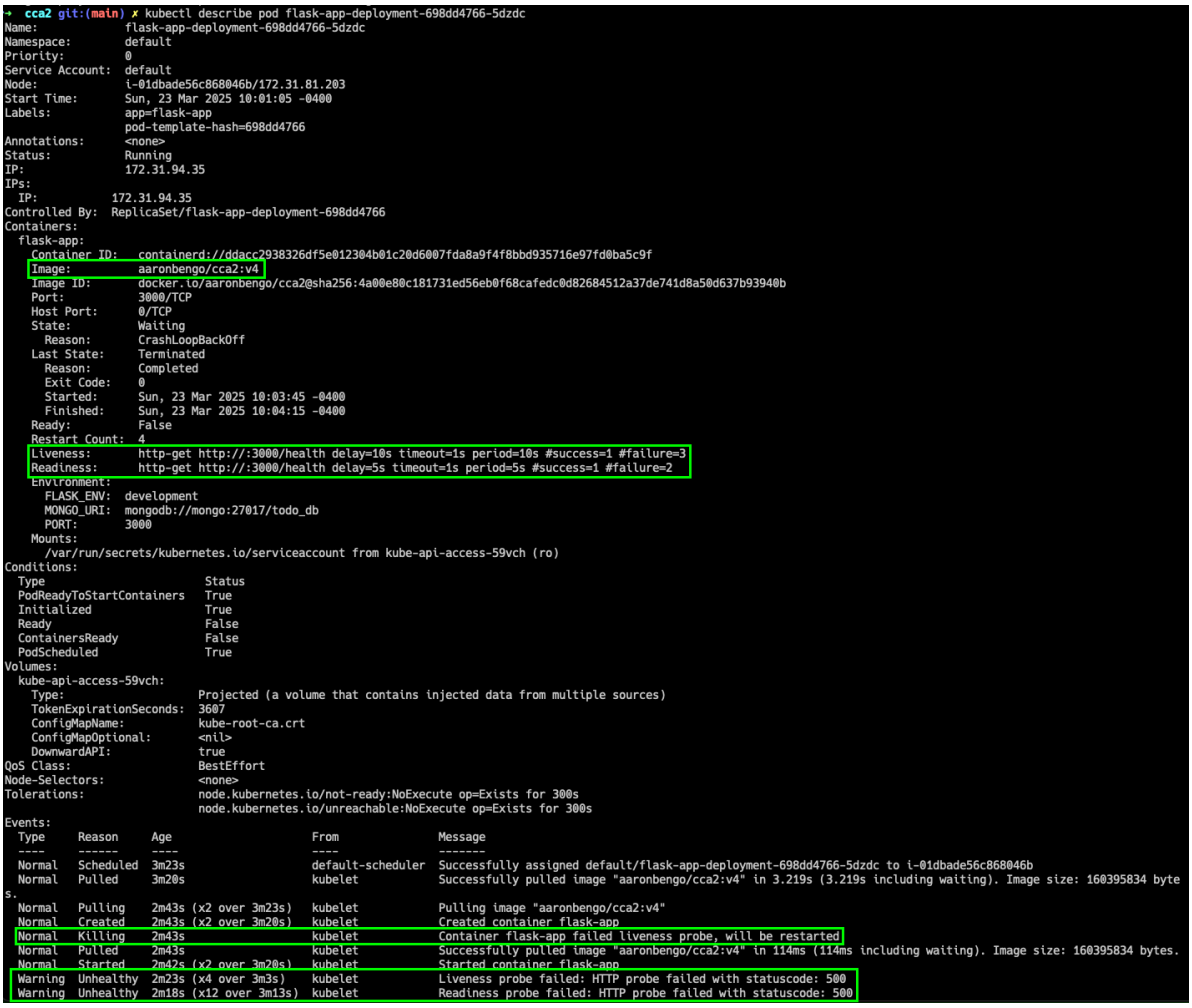
6) Now we simulate an error by returning an error flag in the /health api response

```python
@app.route("/health")
def hello ():
    return "Error", 500
```

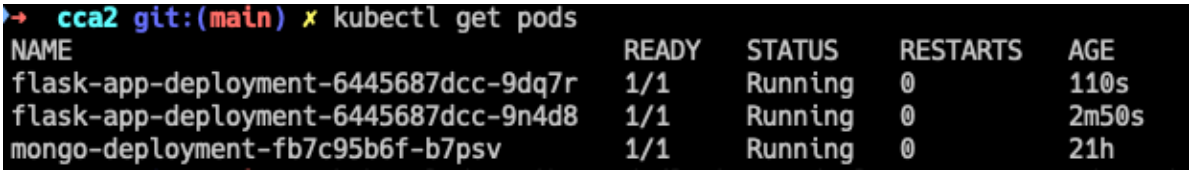
7) We rebuild the image with the /health api update, we name it "aaronbengo/cca2:v4" and we push it to DockerHub

```
→  cca2 git:(main) ✗ docker buildx create --use
docker buildx build --platform linux/amd64,linux/arm64 \
  -t aaronbengo/cca2:v4 --push .

elastic_clarke
[+] Building 34.6s (17/17) FINISHED
```

8) We apply the changes to our EKS cluster and monitor our pods, we see that a new pod is created but not yet ready. We check the pods description for image name, Liveness, Readiness, and Event logs. We find that the service status is switched to Unhealthy due to the 500 statuscode responses, both the Liveness and Readiness probes fail, which prompts the system to try and restart the pod. Since our system continually gets 500 status codes, the pod never actually mounts, we can assume that if the 500 status codes became 200 at one point, then the pod would successfully mount instead of attempting to restart.



9) We conclude by resetting the /health api to return a status code of 200 so that the livenessProbe and readinessProbe can work as expected. With this approach, our system will be able to discern the health of our application tactically. We also confirm that our deployment is still working as expected, and our webpage is live.



We show the logs of one the new pods showing it is checking the /health api endpoint for a 200 status code.

```
→  cca2 git:(main) ✗ kubectl logs flask-app-deployment-6445687dcc-9n4d8
 * Serving Flask app 'app.py' (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.94.36:3000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 527-281-284
172.31.81.203 - - [23/Mar/2025 14:23:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:23:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:24:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:24:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:25:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:25:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:26:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:26:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:27:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:27:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:28:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:28:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:29:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:29:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:30:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:30:18] "GET /health HTTP/1.1" 200 -
172.31.22.84 - - [23/Mar/2025 14:30:34] "GET /static/assets/style.css HTTP/1.1" 304 -
172.31.22.84 - - [23/Mar/2025 14:30:34] "GET /static/images/yes.png HTTP/1.1" 304 -
172.31.22.84 - - [23/Mar/2025 14:30:34] "GET /static/images/no.png HTTP/1.1" 304 -
172.31.81.203 - - [23/Mar/2025 14:31:18] "GET /health HTTP/1.1" 200 -
172.31.81.203 - - [23/Mar/2025 14:31:18] "GET /health HTTP/1.1" 200 -
```

Part 7: Alerting with Prometheus

1) First we install Prometheus onto our cluster

```
→  cca2 git:(main) ✗ helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring --create-namespace

NAME: prometheus
LAST DEPLOYED: Sun Mar 23 17:52:31 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=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets prometheus-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=prometheus" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
```

2) We then check if the Prometheus and Alertmanager pods were set up succesfully

```
→  cca2 git:(main) ✗ kubectl get pods -n monitoring

NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   0          41s
prometheus-grafana-75bb7d6986-pvlkc                      3/3     Running   0          46s
prometheus-kube-prometheus-operator-65c669f8f9-95hfk     1/1     Running   0          46s
prometheus-kube-state-metrics-645c667b6-xqrfk            1/1     Running   0          46s
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0          41s
prometheus-prometheus-node-exporter-42cp9                1/1     Running   0          46s
prometheus-prometheus-node-exporter-4rjtb                1/1     Running   0          46s
```

3) We check the active rules in the prometheusrule monitoring namespace

```
➜  cca2 git:(main) ✗ kubectl get prometheusrule -n monitoring

NAME                                                              AGE
prometheus-kube-prometheus-alertmanager.rules                     16s
prometheus-kube-prometheus-config-reloaders                       16s
prometheus-kube-prometheus-etcd                                   16s
prometheus-kube-prometheus-general.rules                          16s
prometheus-kube-prometheus-k8s.rules.container-cpu-usage-second   16s
prometheus-kube-prometheus-k8s.rules.container-memory-cache       16s
prometheus-kube-prometheus-k8s.rules.container-memory-rss         16s
prometheus-kube-prometheus-k8s.rules.container-memory-swap        16s
prometheus-kube-prometheus-k8s.rules.container-memory-working-s   16s
prometheus-kube-prometheus-k8s.rules.container-resource           16s
prometheus-kube-prometheus-k8s.rules.pod-owner                    16s
prometheus-kube-prometheus-kube-apiserver-availability.rules      16s
prometheus-kube-prometheus-kube-apiserver-burnrate.rules          16s
prometheus-kube-prometheus-kube-apiserver-histogram.rules         16s
prometheus-kube-prometheus-kube-apiserver-slos                    16s
prometheus-kube-prometheus-kube-prometheus-general.rules          16s
prometheus-kube-prometheus-kube-prometheus-node-recording.rules   16s
prometheus-kube-prometheus-kube-scheduler.rules                   16s
prometheus-kube-prometheus-kube-state-metrics                     16s
prometheus-kube-prometheus-kubelet.rules                          16s
prometheus-kube-prometheus-kubernetes-apps                        16s
prometheus-kube-prometheus-kubernetes-resources                   16s
prometheus-kube-prometheus-kubernetes-storage                     16s
prometheus-kube-prometheus-kubernetes-system                      16s
prometheus-kube-prometheus-kubernetes-system-apiserver            16s
prometheus-kube-prometheus-kubernetes-system-controller-manager   16s
prometheus-kube-prometheus-kubernetes-system-kube-proxy           16s
prometheus-kube-prometheus-kubernetes-system-kubelet              16s
prometheus-kube-prometheus-kubernetes-system-scheduler            16s
prometheus-kube-prometheus-node-exporter                          16s
prometheus-kube-prometheus-node-exporter.rules                    16s
prometheus-kube-prometheus-node-network                           16s
prometheus-kube-prometheus-node.rules                             16s
prometheus-kube-prometheus-prometheus                             16s
prometheus-kube-prometheus-prometheus-operator                    16s
```

4) We setup a custom-alerts.yaml manifest for our PrometheusRule rules

custom-alerts.yaml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: custom-alerts
  namespace: monitoring
  labels:
    release: prometheus    # This label matches the ruleSelector of your Prometheus instance.
    role: alert-rules      # Optionally, you can keep additional labels if desired.
spec:
  groups:
  - name: alerts
    rules:
    - alert: PodDeletionDetectedForFlaskDeployment
      expr: kube_deployment_status_replicas_available{deployment="flask-app-deployment", namespace="default"}
            < kube_deployment_spec_replicas{deployment="flask-app-deployment", namespace="default"}
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "Pod deletion detected for deployment {{ $labels.deployment }}"
        description: "The deployment {{ $labels.deployment }} is missing pods: desired replicas are less than available replica
    - alert: PodCrashLooping
      expr: kube_pod_container_status_restarts_total{namespace="default", container="flask-app"} > 1
      for: 2m
      labels:
        severity: critical
      annotations:
        summary: "Pod {{ $labels.pod }} in namespace {{ $labels.namespace }} is crash looping"
        description: "The pod has restarted more than 1 time in the last 2 minutes."
    - alert: DeploymentHasTooFewReplicas
      expr: kube_deployment_status_replicas_unavailable{deployment="flask-app-deployment", namespace="default"} > 0
      for: 2m
      labels:
        severity: warning
      annotations:
        summary: "Deployment {{ $labels.deployment }} has unavailable replicas"
        description: "The deployment does not have all replicas running."
```

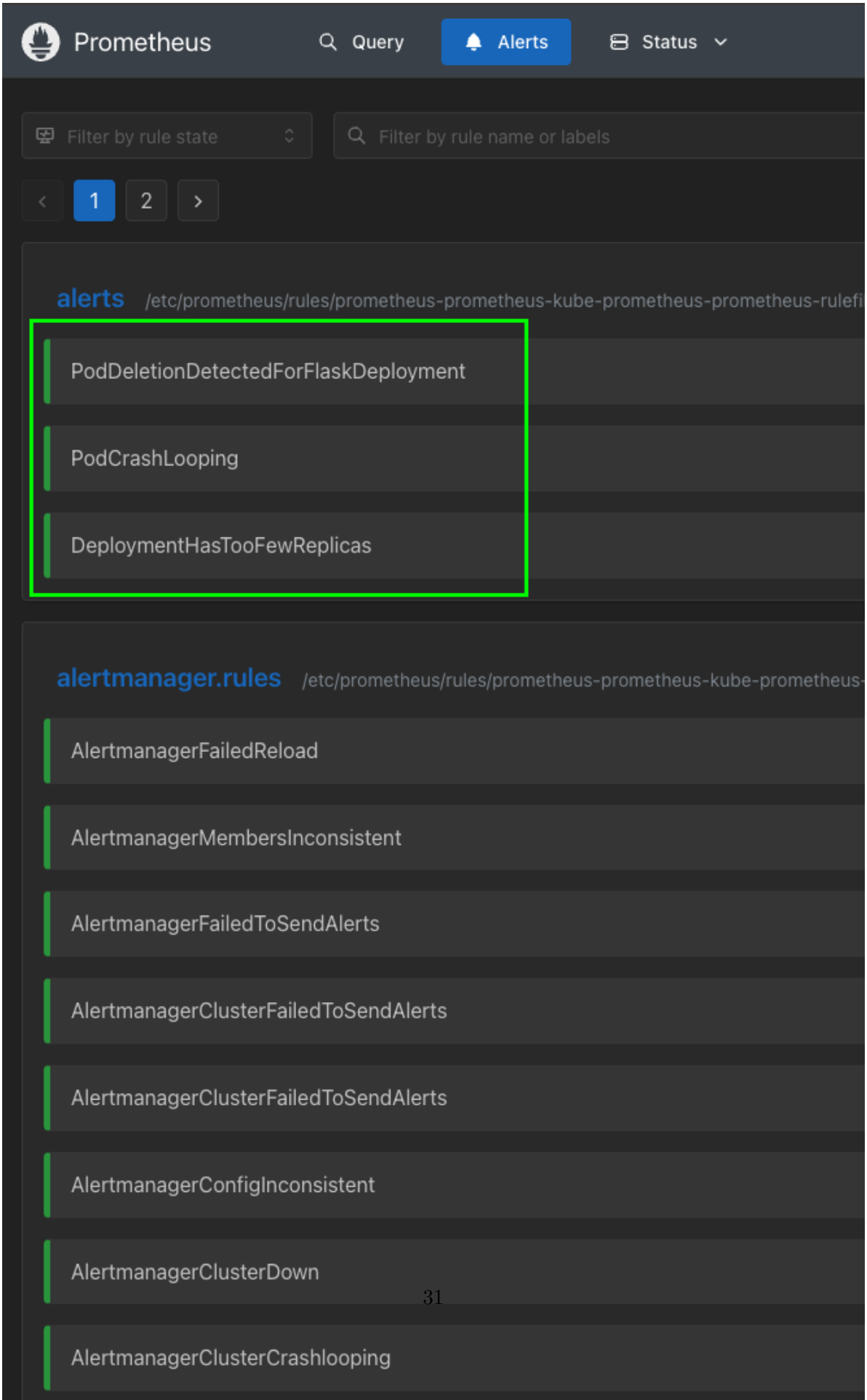
5) We apply the custom-alerts manifest to our cluster

```
→  cca2 git:(main) ✗ kubectl apply -f custom-alerts.yaml -n monitoring

prometheusrule.monitoring.coreos.com/custom-alerts created
```

6) Again we check the active rules in the prometheusrule monitoring namespace

```
→  cca2 git:(main) ✗ kubectl get prometheusrule -n monitoring

NAME                                                              AGE
custom-alerts                                                     35s
prometheus-kube-prometheus-alertmanager.rules                     4m5s
prometheus-kube-prometheus-config-reloaders                       4m5s
prometheus-kube-prometheus-etcd                                   4m5s
prometheus-kube-prometheus-general.rules                          4m5s
prometheus-kube-prometheus-k8s.rules.container-cpu-usage-second   4m5s
prometheus-kube-prometheus-k8s.rules.container-memory-cache       4m5s
prometheus-kube-prometheus-k8s.rules.container-memory-rss         4m5s
prometheus-kube-prometheus-k8s.rules.container-memory-swap        4m5s
prometheus-kube-prometheus-k8s.rules.container-memory-working-s   4m5s
prometheus-kube-prometheus-k8s.rules.container-resource           4m5s
prometheus-kube-prometheus-k8s.rules.pod-owner                    4m5s
prometheus-kube-prometheus-kube-apiserver-availability.rules      4m5s
prometheus-kube-prometheus-kube-apiserver-burnrate.rules          4m5s
prometheus-kube-prometheus-kube-apiserver-histogram.rules         4m5s
prometheus-kube-prometheus-kube-apiserver-slos                    4m5s
prometheus-kube-prometheus-kube-prometheus-general.rules          4m5s
prometheus-kube-prometheus-kube-prometheus-node-recording.rules   4m5s
prometheus-kube-prometheus-kube-scheduler.rules                   4m5s
prometheus-kube-prometheus-kube-state-metrics                     4m5s
prometheus-kube-prometheus-kubelet.rules                          4m5s
prometheus-kube-prometheus-kubernetes-apps                        4m5s
prometheus-kube-prometheus-kubernetes-resources                   4m5s
prometheus-kube-prometheus-kubernetes-storage                     4m5s
prometheus-kube-prometheus-kubernetes-system                      4m5s
prometheus-kube-prometheus-kubernetes-system-apiserver            4m5s
prometheus-kube-prometheus-kubernetes-system-controller-manager   4m5s
prometheus-kube-prometheus-kubernetes-system-kube-proxy           4m5s
prometheus-kube-prometheus-kubernetes-system-kubelet              4m5s
prometheus-kube-prometheus-kubernetes-system-scheduler            4m5s
prometheus-kube-prometheus-node-exporter                          4m5s
prometheus-kube-prometheus-node-exporter.rules                    4m5s
prometheus-kube-prometheus-node-network                           4m5s
prometheus-kube-prometheus-node.rules                             4m5s
prometheus-kube-prometheus-prometheus                             4m5s
prometheus-kube-prometheus-prometheus-operator                    4m5s
```

7) We show a snapshot of the prometheus UI's alert tab, which shows our three custom rules active, this means that prometheus was able to sync them using the prometheus operator.


Prometheus
Query
Alerts
Status
Filter by rule state
Filter by rule name or labels
1
2
alerts
/etc/prometheus/rules/prometheus-prometheus-kube-prometheus-prometheus-rulefi
PodDeletionDetectedForFlaskDeployment
PodCrashLooping
DeploymentHasTooFewReplicas
alertmanager.rules
/etc/prometheus/rules/prometheus-prometheus-kube-prometheus-
AlertmanagerFailedReload
AlertmanagerMembersInconsistent
AlertmanagerFailedToSendAlerts
AlertmanagerClusterFailedToSendAlerts
AlertmanagerClusterFailedToSendAlerts
AlertmanagerConfigInconsistent
AlertmanagerClusterDown
AlertmanagerClusterCrashlooping

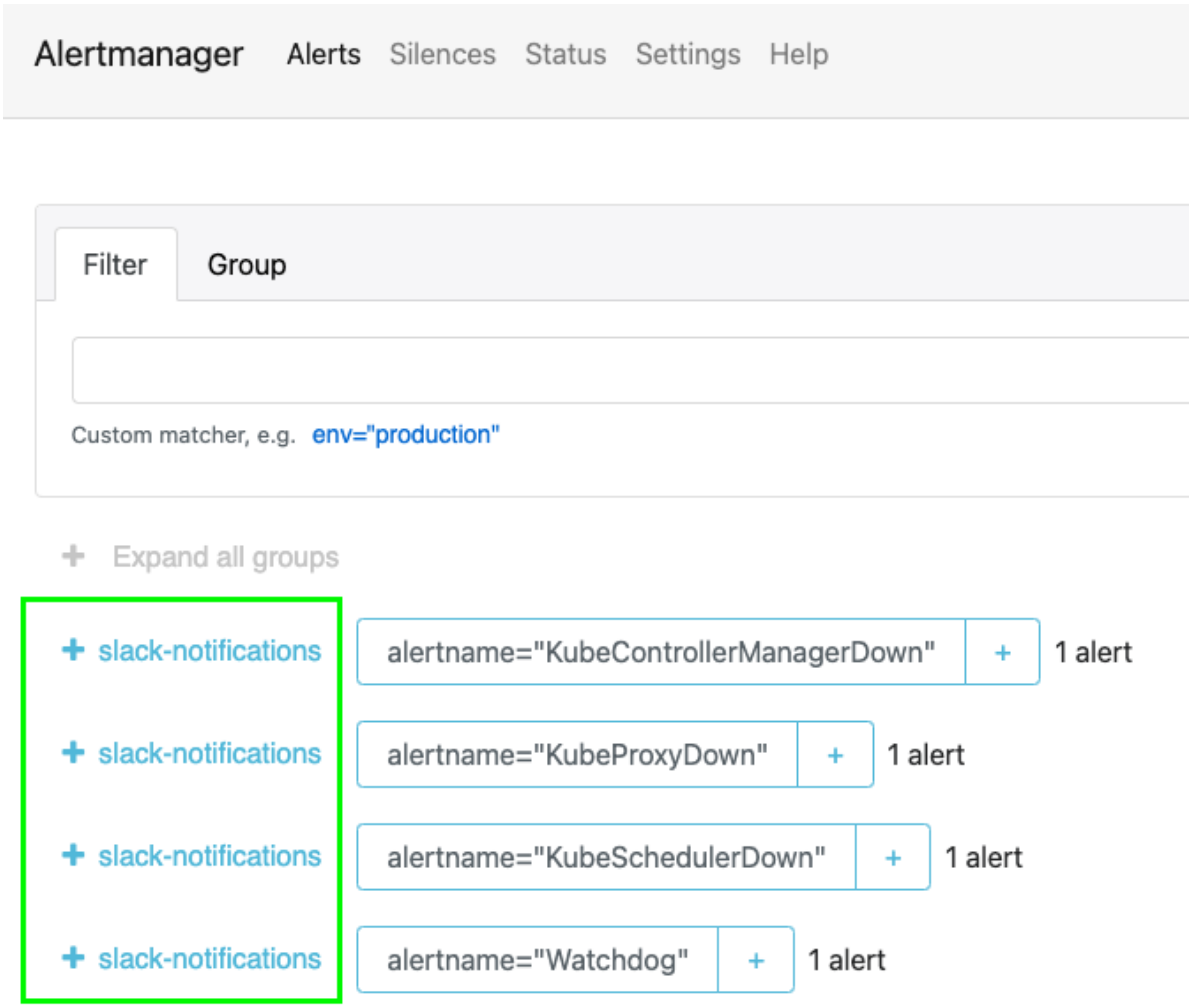
8) Next, we shift our focus to slack messages on the event that one of our alerts is triggered. We create the alertmanager-override.yaml manifest in order to specify the alertmanager config.

```
! alertmanager-overrides.yaml
alertmanager:
  config:
    global:
      slack_api_url: "https://hooks.slack.com/services/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"
    route:
      group_by: ['alertname']            # grouping keys (adjust as needed)
      group_wait: 30s
      group_interval: 5m
      repeat_interval: 12h
      receiver: slack-notifications      # send all alerts to Slack by default
      routes:
        - matchers:
            - alertname = "Watchdog"     # drop the Watchdog heartbeat alert
          receiver: null
    receivers:
      - name: "null"
      - name: "slack-notifications"
        slack_configs:
          - channel: "#test"        # Slack channel for alerts
            send_resolved: true
            # Optional: title, text, etc., or rely on Alertmanager defaults/templates
```

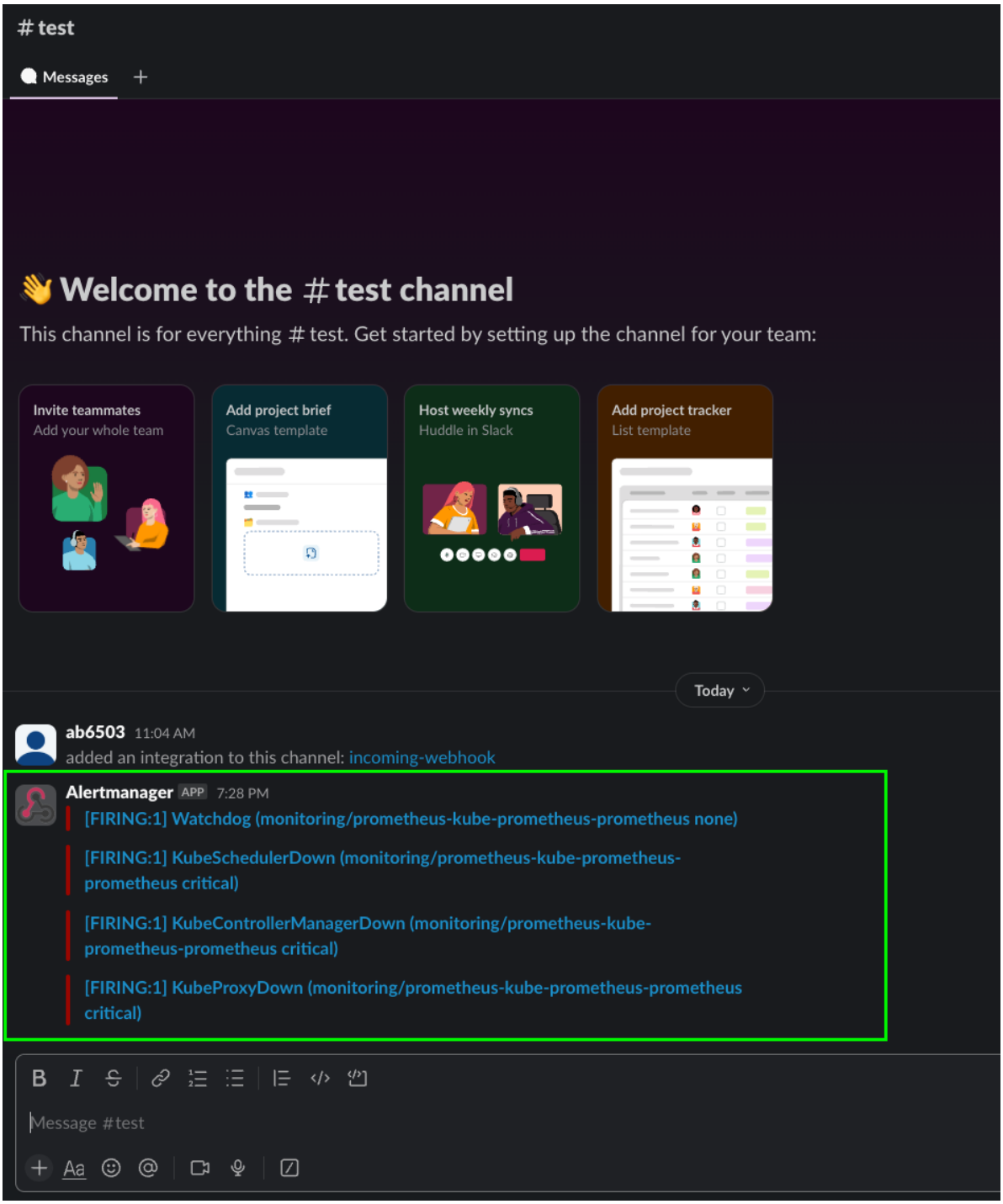
9) We use helm in order to upgrade the prometheus deployment, this time we pass our alertmanager-override.yaml manifest as an argument for the alertmanager service config.

```
→  cca2 git:(main) ✗ helm upgrade prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  -f alertmanager-overrides.yaml

Release "prometheus" has been upgraded. Happy Helming!
NAME: prometheus
LAST DEPLOYED: Sun Mar 23 19:14:47 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 3
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets prometheus-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=prometheus" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
```

10) We observe that the receiver is now live on the Alertmanager UI. It is important to note that my submission lacks my 3 defined custom alerts. I had issues trying to get them added an was unable to in by the deadline of the assignment, I am sharing all the progress that I was able to make.



11) I conclude by showing that the services that appead in the Alermanager UI were live and ready to communicate with my slack channel.


test
Messages
Welcome to the # test channel
This channel is for everything # test. Get started by setting up the channel for your team:
Invite teammates
Add your whole team
Add project brief
Canvas template
Host weekly syncs
Huddle in Slack
Add project tracker
List template
Today
ab6503 11:04 AM
added an integration to this channel: incoming-webhook
Alertmanager APP 7:28 PM
[FIRING:1] Watchdog (monitoring/prometheus-kube-prometheus-prometheus none)
[FIRING:1] KubeSchedulerDown (monitoring/prometheus-kube-prometheus-prometheus critical)
[FIRING:1] KubeControllerManagerDown (monitoring/prometheus-kube-prometheus-prometheus critical)
[FIRING:1] KubeProxyDown (monitoring/prometheus-kube-prometheus-prometheus critical)
Message #test