\pagenumbering{gobble}

CS-GY 9223 Cloud Computing

Assignment 2

Suprateek Chatterjee sc10344@nyu.edu

Karmanya Mendiratta km6296@nyu.edu

Kubernetes and Docker

Containerizing the application

For this part of the assignment we need to have `docker` and `docker-compose` installed on the machine.

```
$ docker --version
Docker version 24.0.6, build ed223bc
```

```
$ docker compose version
Docker Compose version v2.21.0-desktop.1
```

To containerize the application, we need to create a `Dockerfile` with the following steps to create the image.

```
# 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 current directory contents into the container at /app
COPY . /app

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

# Make port 3000 available to the world outside this container
EXPOSE 3000

# Define environment variable
ENV FLASK_ENV=development
ENV PORT=3000

# Run app.py when the container launches
CMD ["flask", "run", "--host=0.0.0.0", "--port=3000"]
```

1. **`FROM python`**: This line specifies the base image for the container. In this case, it's using the official Python image as the base for the container.

2. **`WORKDIR /app`**: It sets the working directory within the container to `/app`. This is where subsequent commands will be executed.

3. **`COPY . /app`**: This line copies the contents of the local directory named `web` into the current working directory of the container. The dot **`.`** represents the current directory in the container.

4. **`RUN pip install -r requirements.txt`**: This command runs the pip install command inside the container. It installs the Python packages listed in the requirements.txt file, assuming that the file exists in the current working directory of the container.

5. **`EXPOSE 3000`**: This line informs Docker that the container will listen on port 5000.

However, it doesn't actually publish the port to the host system. We'll need to map this port when running the container.

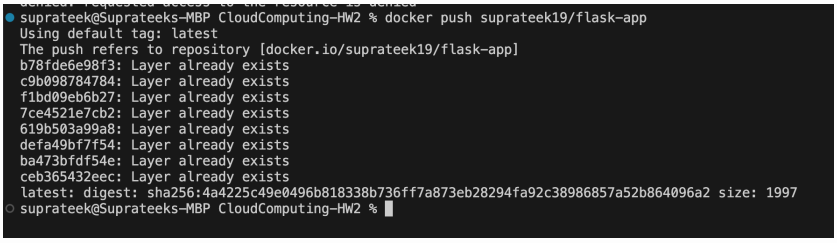
6. **`CMD flask run --host 0.0.0.0`**: This sets the default command that will be executed when the container is started. It runs the Flask application using `flask run` and binds it to all available network interfaces using `--host 0.0.0.0`. This makes the Flask app accessible externally.

Once the Dockerfile has been created, the next step is to build the image.

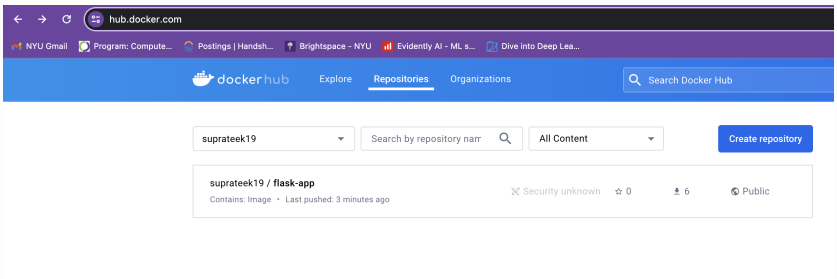
```
$ docker build -t flask-app .
```

Pushing to Dockerhub

```
$ docker push suprateek19/flask-app
```



We can see on Docker Hub that the image was pushed successfully.



Testing the application locally

We will utilize docker-compose to create containers for both the Flask application and a MongoDB instance. The Docker Compose file is written using YAML.

```yaml
version: '3.8'
services:
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      - FLASK_ENV=development
      - MONGO_HOST=mongodb
    depends_on:
      - mongodb

  mongodb:
    image: mongo:latest
    volumes:
      - mongo-data:/data/db

volumes:
  mongo-data:
```

- `version: '3.8'`: Specifies the version of the Docker Compose file format. This determines which features will be available.

- `services`: This is the top-level key in a Docker Compose file, and it defines the list of services or containers we want to create and manage.

Service: `web`

- This is the name of the first service, which is called "web."
- `build: .`: Indicates that Docker should build the image for this service using the Dockerfile in the current directory.
- `ports`:
  - This section defines port mapping for the container. It maps the host port 3000 to the container port 3000. This allows access to the Flask application on the host machine at port 3000, with Docker forwarding the traffic to the Flask application running inside the container on the same port.
- `environment`:
  - Sets environment variables for the service. `FLASK_ENV` is set to "development", and `MONGO_HOST` is set to "mongodb", directing the Flask application to the MongoDB service for database interactions.
- `depends_on`:
  - Specifies that the "web" service depends on the "mongodb" service, ensuring that the "mongodb" service starts before the "web" service. This is crucial because the Flask application relies on MongoDB for data storage.

Service: `mongodb`

- This is the name of the second service, which is called "mongodb."
- `image: mongo:latest`: Specifies the Docker image to use for the "mongodb" service, using the latest version of the official MongoDB image.
- `volumes`:
  - Defines a data volume named `mongo-data` for the "mongodb" service. This volume is mounted at `/data/db` inside the container, providing persistent storage for MongoDB data.

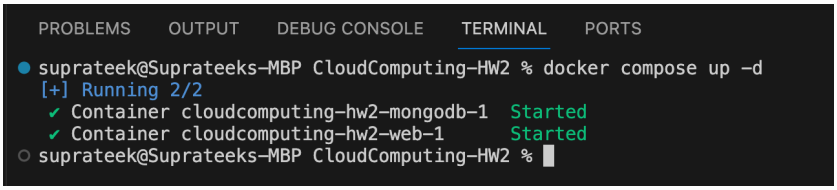
Volumes:

- `mongo-data`: Defines a named volume that persists data stored by the MongoDB container, ensuring data is not lost when the container is stopped or removed.

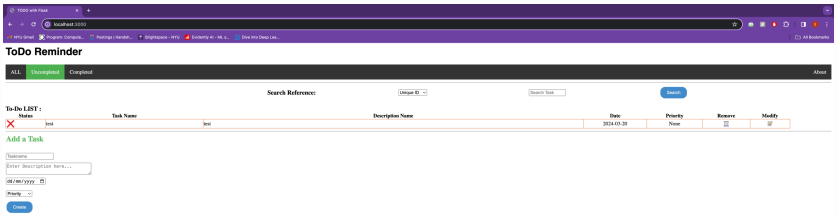
Summary

This Docker Compose file configures two services: one for a Flask web application (`web`) and another for a MongoDB database (`mongodb`). It outlines their Docker images, build contexts, port mappings, environment variables, dependencies, and volumes, creating a cohesive and interdependent application environment.

```
$ docker compose up -d
```



Here is the website's user interface when I visit `localhost:3000`



Deploying the application on Minikube

For this part of the assignment we need to have `minikube` and `kubectl` installed on the machine.

```
$ minikube version
minikube version: v1.31.2
commit: fd7ecd9c4599bef9f04c0986c4a0187f98a4396e
```

```
$ kubectl version
Client Version: v1.28.3
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.27.4
```

{{< pagebreak >}}

Next, I'll create the deployment and service for both the Flask app and MongoDB.

```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: suprateek19/flask-app:latest
        ports:
        - containerPort: 3000
        env:
        - name: MONGO_HOST
          value: mongodb
        - name: MONGO_PORT
          value: "27017"

---
apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  type: NodePort
  ports:
  - port: 3000
    targetPort: 3000
    nodePort: 30007
  selector:
    app: flask-app
```

- Deployment Section:
  - `apiVersion: apps/v1`: Specifies the API version for the Deployment.
  - `kind: Deployment`: Defines the type of Kubernetes resource as a Deployment.
  - `metadata`: Contains metadata for the Deployment, including the name and labels.
  - `spec`: Describes the desired state for the Deployment.
  - `replicas: 2`: Specifies that there should be two replicas of the pod.
  - `selector`: Defines how the Deployment finds which pods to manage.
  - `matchLabels`: Selects pods with the label "app: flask-app".
  - `template`: Describes the pods that will be created.
    - `metadata`: Contains labels for the pod.
    - `spec`: Specifies the pod's specification.
      - `containers`: Defines the containers within the pod.
        - `name`: Names the container "flask-app".
        - `image`: Specifies the Docker image for the Flask app as `suprateek19/flask-app:latest`.
        - `ports`: Specifies that the container will listen on port 3000.
        - `env`: Sets environment variables for the container, such as `MONGO_HOST` with the value `mongodb`.
- Service Section:
  - `apiVersion: v1`: Specifies the API version for the Service.
  - `kind: Service`: Defines the type of Kubernetes resource as a Service.
  - `metadata`: Contains metadata for the Service, including the name.
  - `spec`: Describes the desired state for the Service.
    - `type`: NodePort: Exposes the Service on each node's IP at a static port (nodePort: 30007).
    - `selector`: Selects pods with the label "app: flask-app".
    - `ports`: Specifies the ports that the Service will forward.
      - `port: 3000`: Specifies the port on the Service.
      - `targetPort: 3000`: Specifies the port on the pod to which the Service forwards.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-deployment
  labels:
    app: mongo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
        - name: mongo
          image: mongo
          ports:
            - containerPort: 27017
```

- This section defines a Kubernetes Deployment named "mongo-deployment" for a MongoDB instance.
- One replica of the pod is specified (replicas: 1).
- The pod selector is set to match pods with the label "app: mongo."
- The pod template includes a container named "mongo" using the official MongoDB Docker image.
- The container is configured to listen on port 27017.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  type: NodePort
  selector:
    app: mongo
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
      nodePort: 30100
```

- This section defines a Kubernetes Service named "mongo-service" to expose the MongoDB instance.
- The Service type is set to NodePort, exposing the MongoDB service on each node's IP at port 30100.
- The Service selects pods with the label "app: mongo."
- It forwards traffic from port 30100 to port 27017 on the selected pods.

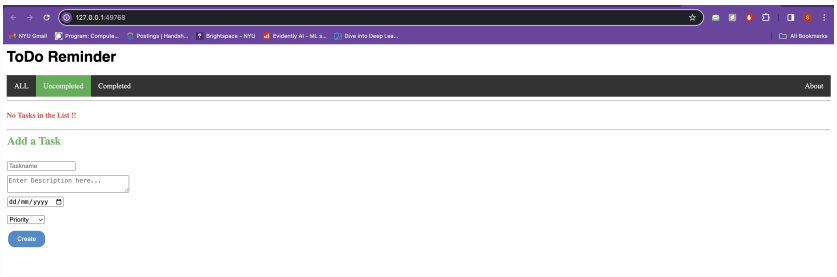
```
$ kubectl apply -f mongo-deployment.yml
deployment.apps/mongo-deployment unchanged
service/mongo-service unchanged
```

```
$ kubectl apply -f web-deployment.yml
deployment.apps/web-deployment unchanged
service/web-service unchanged
```

```
suprateek@Suprateeks-MBP CloudComputing-HW2 % kubectl get all
NAME                                        READY   STATUS    RESTARTS      AGE
pod/flask-app-deployment-5b6695c889-lb4gg   1/1     Running   2 (14m ago)   16d
pod/flask-app-deployment-5b6695c889-zc9ht   1/1     Running   2 (14m ago)   16d
pod/mongodb-deployment-7cfc99f98-kfzn8      1/1     Running   2 (14m ago)   16d

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
service/flask-app-service    NodePort    10.99.152.86    <none>        3000:30007/TCP    16d
service/kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP           16d
service/mongodb              ClusterIP   10.107.186.63   <none>        27017/TCP         16d

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flask-app-deployment   2/2     2            2           16d
deployment.apps/mongodb-deployment     1/1     1            1           16d

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/flask-app-deployment-5b6695c889   2         2         2       16d
replicaset.apps/flask-app-deployment-f8b7f9bcb    0         0         0       16d
replicaset.apps/mongodb-deployment-55b7c4c4cf     0         0         0       16d
replicaset.apps/mongodb-deployment-7cfc99f98      1         1         1       16d
suprateek@Suprateeks-MBP CloudComputing-HW2 %
```

```
$ minikube service --all
```

```
suprateek@Suprateeks-MBP CloudComputing-HW2 % minikube service --all
|-----------|--------------------|-------------|---------------------------|
| NAMESPACE |        NAME        | TARGET PORT |            URL            |
|-----------|--------------------|-------------|---------------------------|
| default   | flask-app-service  |        3000 | http://192.168.49.2:30007 |
|-----------|--------------------|-------------|---------------------------|
|-----------|------------|-------------|---------------|
| NAMESPACE |    NAME    | TARGET PORT |      URL      |
|-----------|------------|-------------|---------------|
| default   | kubernetes |             | No node port  |
|-----------|------------|-------------|---------------|
😿  service default/kubernetes has no node port
|-----------|---------|-------------|---------------|
| NAMESPACE |  NAME   | TARGET PORT |      URL      |
|-----------|---------|-------------|---------------|
| default   | mongodb |             | No node port  |
|-----------|---------|-------------|---------------|
😿  service default/mongodb has no node port
🏃  Starting tunnel for service flask-app-service.
🏃  Starting tunnel for service kubernetes.
🏃  Starting tunnel for service mongodb.
|-----------|--------------------|-------------|------------------------|
| NAMESPACE |        NAME        | TARGET PORT |          URL           |
|-----------|--------------------|-------------|------------------------|
| default   | flask-app-service  |             | http://127.0.0.1:49768 |
| default   | kubernetes         |             | http://127.0.0.1:49770 |
| default   | mongodb            |             | http://127.0.0.1:49772 |
|-----------|--------------------|-------------|------------------------|
🎉  Opening service default/flask-app-service in default browser...
🎉  Opening service default/kubernetes in default browser...
🎉  Opening service default/mongodb in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```



{{< pagebreak >}}

Adding load balancer

To incorporate the load balancer for the web application, we simply need to update the `spec.type` to `LoadBalancer`.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  type: LoadBalancer
  ports:
  - port: 3000
    targetPort: 3000
  selector:
    app: flask-app
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

suprateek@Suprateeks-MBP CloudComputing-HW2 % kubectl get all
NAME                                     READY   STATUS    RESTARTS      AGE
pod/flask-app-deployment-5b6695c889-lb4gg   1/1     Running   2 (21m ago)   16d
pod/flask-app-deployment-5b6695c889-zc9ht   1/1     Running   2 (21m ago)   16d
pod/mongodb-deployment-7cfc99f98-kfzn8      1/1     Running   2 (21m ago)   16d

NAME                        TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/flask-app-service   LoadBalancer   10.99.152.86    <pending>     3000:30007/TCP   16d
service/kubernetes          ClusterIP      10.96.0.1       <none>        443/TCP          16d
service/mongodb             ClusterIP      10.107.186.63   <none>        27017/TCP        16d

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flask-app-deployment   2/2     2            2           16d
deployment.apps/mongodb-deployment     1/1     1            1           16d

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/flask-app-deployment-5b6695c889   2         2         2       16d
replicaset.apps/flask-app-deployment-f8b7f9bcb    0         0         0       16d
replicaset.apps/mongodb-deployment-55b7c4c4cf     0         0         0       16d
replicaset.apps/mongodb-deployment-7cfc99f98      1         1         1       16d
suprateek@Suprateeks-MBP CloudComputing-HW2 %
```

Deploying the application on EKS

Rather than manually setting up the cluster on EKS through the UI, we can streamline the process by utilizing the eksctl CLI, which automates the creation using CloudFormation scripts.

```
$ eksctl version
0.174.0
```

We also have to update the OS of our build on docker to align with our EKS OS. I have run the following command to achieve the same:

```
Karmanyas-MBP:todo_app karmanyamendiratta$ docker buildx build --platform linux/amd64 -t karmanya1804/todoapp:latest --push .
[+] Building 3.3s (10/10) FINISHED                                  docker:desktop-linux
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 240B                                                0.0s
 => [internal] load metadata for docker.io/library/python:3                         0.3s
 => [1/4] FROM docker.io/library/python:3@sha256:336461f63f4eb1100e178d5acbfea3     0.0s
 => [internal] load build context                                                   0.0s
 => => transferring context: 2.69kB                                                 0.0s
 => CACHED [2/4] WORKDIR /app                                                       0.0s
 => CACHED [3/4] COPY . /app                                                        0.0s
 => CACHED [4/4] RUN pip install --no-cache-dir -r requirements.txt                 0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:7c0589f4c6b1ca0493b5c5646a1896ff8ba6c525c2b79303965     0.0s
 => => naming to docker.io/karmanya1804/todoapp:latest                              0.0s
 => pushing karmanya1804/todoapp:latest with docker                                 2.2s
 => => pushing layer 03f7306b0bed                                                   2.0s
 => => pushing layer 8cfafc5dc18f                                                   2.0s
 => => pushing layer 33bd3a424e6e                                                   2.0s
 => => pushing layer 9adbc4b1428d                                                   2.0s
 => => pushing layer f52093e4f67d                                                   2.0s
 => => pushing layer 1193f41e6b14                                                   2.0s
 => => pushing layer e077e19b6682                                                   2.0s
 => => pushing layer 21e1c4948146                                                   2.0s
 => => pushing layer 68866beb2ed2                                                   2.0s
 => => pushing layer e6e2ab10dba6                                                   2.0s
 => => pushing layer 0238a1790324                                                   2.0s
```

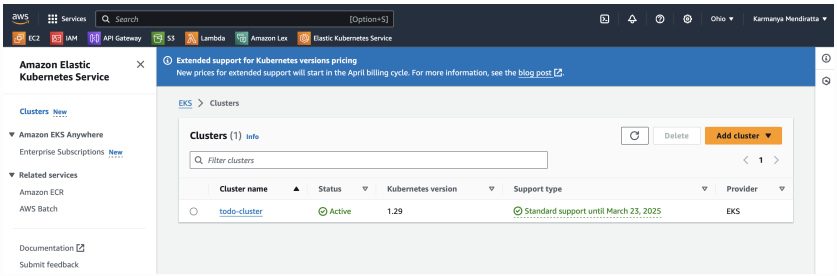
{{< pagebreak >}}

```
$ eksctl create cluster --name todo-cluster --node-type=t2.large --
nodes=4 --region=us-east-2
```

```
Karmanyas-MBP:todo_app karmanyamendiratta$ eksctl create cluster --name todo-cluster --node-type=t2.large --nodes=4 --region
=us-east-2
2024-03-17 17:40:35 [ℹ]  eksctl version 0.174.0
2024-03-17 17:40:35 [ℹ]  using region us-east-2
2024-03-17 17:40:35 [ℹ]  setting availability zones to [us-east-2b us-east-2c us-east-2a]
2024-03-17 17:40:35 [ℹ]  subnets for us-east-2b - public:192.168.0.0/19 private:192.168.96.0/19
2024-03-17 17:40:35 [ℹ]  subnets for us-east-2c - public:192.168.32.0/19 private:192.168.128.0/19
2024-03-17 17:40:35 [ℹ]  subnets for us-east-2a - public:192.168.64.0/19 private:192.168.160.0/19
2024-03-17 17:40:35 [ℹ]  nodegroup "ng-d5b3e5e7" will use "" [AmazonLinux2/1.29]
2024-03-17 17:40:35 [ℹ]  using Kubernetes version 1.29
2024-03-17 17:40:35 [ℹ]  creating EKS cluster "todo-cluster" in "us-east-2" region with managed nodes
2024-03-17 17:40:35 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-03-17 17:40:35 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-2 --cluster=todo-cluster'
2024-03-17 17:40:35 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "todo-cluster" in "us-east-2"
2024-03-17 17:40:35 [ℹ]  CloudWatch logging will not be enabled for cluster "todo-cluster" in "us-east-2"
2024-03-17 17:40:35 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-2 --
ster=todo-cluster'
2024-03-17 17:40:35 [ℹ]
2 sequential tasks: { create cluster control plane "todo-cluster",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "ng-d5b3e5e7",
    }
}
2024-03-17 17:40:35 [ℹ]  building cluster stack "eksctl-todo-cluster-cluster"
2024-03-17 17:40:35 [ℹ]  deploying stack "eksctl-todo-cluster-cluster"
2024-03-17 17:41:05 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:41:35 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:42:36 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:43:36 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:44:36 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:45:36 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:46:37 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:47:37 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:48:37 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:49:37 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-cluster"
2024-03-17 17:51:40 [ℹ]  building managed nodegroup stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:51:40 [ℹ]  deploying stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:51:40 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:52:10 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:53:10 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:55:08 [ℹ]  waiting for CloudFormation stack "eksctl-todo-cluster-nodegroup-ng-d5b3e5e7"
2024-03-17 17:55:08 [ℹ]  waiting for the control plane to become ready
2024-03-17 17:55:09 [✔]  saved kubeconfig as "/Users/karmanyamendiratta/.kube/config"
2024-03-17 17:55:09 [ℹ]  no tasks
2024-03-17 17:55:09 [✔]  all EKS cluster resources for "todo-cluster" have been created
2024-03-17 17:55:09 [ℹ]  nodegroup "ng-d5b3e5e7" has 4 node(s)
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-1-210.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-39-172.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-85-128.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-94-239.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  waiting for at least 4 node(s) to become ready in "ng-d5b3e5e7"
2024-03-17 17:55:09 [ℹ]  nodegroup "ng-d5b3e5e7" has 4 node(s)
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-1-210.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-39-172.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-85-128.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  node "ip-192-168-94-239.us-east-2.compute.internal" is ready
2024-03-17 17:55:09 [ℹ]  kubectl command should work with "/Users/karmanyamendiratta/.kube/config", try 'kubectl get nodes'
2024-03-17 17:55:09 [✔]  EKS cluster "todo-cluster" in "us-east-2" region is ready
```

{{< pagebreak >}}

This created a cluster on my AWS EKS as follows:



Applying the deployments and services for both the flask app and mongoDB:

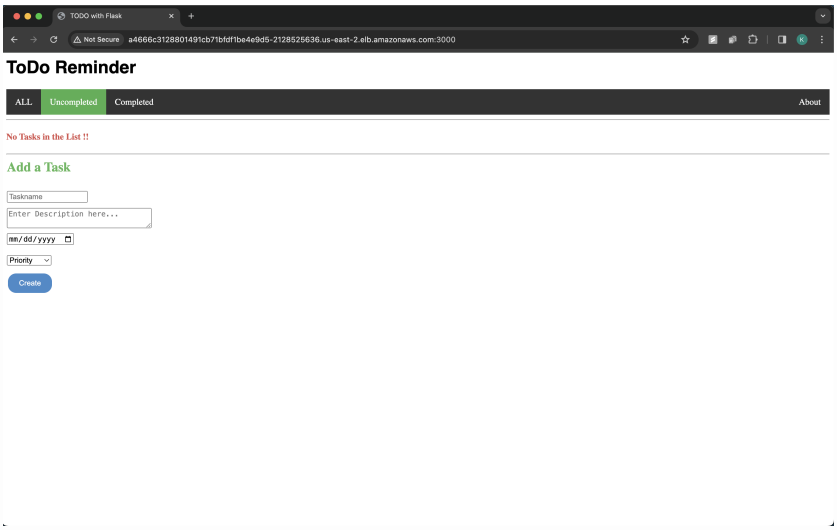
```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f flask-app-deployment.yaml
deployment.apps/flask-app-deployment created
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb-deployment created
```

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f flask-app-service.yaml
service/flask-app-service created
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f mongodb-service.yaml
service/mongodb created
```

After applying the deployments for both the Flask app and MongoDB.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
flask-app-deployment   2/2     2            2           51m
mongodb-deployment     1/1     1            1           51m
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS      AGE
flask-app-deployment-7dc476bc58-6cfk4   1/1     Running   7 (45m ago)   51m
flask-app-deployment-7dc476bc58-kvf7h   1/1     Running   7 (45m ago)   51m
mongodb-deployment-7cfc99f98-8k98h      1/1     Running   0             51m
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get svc
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP                                                                    PORT(S)           AGE
flask-app-service   LoadBalancer   10.100.123.3    a4666c3128801491cb71bfdf1be4e9d5-2128525636.us-east-2.elb.amazonaws.com   3000:30007/TCP    53m
kubernetes          ClusterIP      10.100.0.1      <none>                                                                         443/TCP           64m
mongodb             ClusterIP      10.100.68.120   <none>                                                                         27017/TCP         53m
```

{{< pagebreak >}}



Adding persistent volume

First, install the EBS CSI driver and controller on the nodes. Second, attach the IAM role.

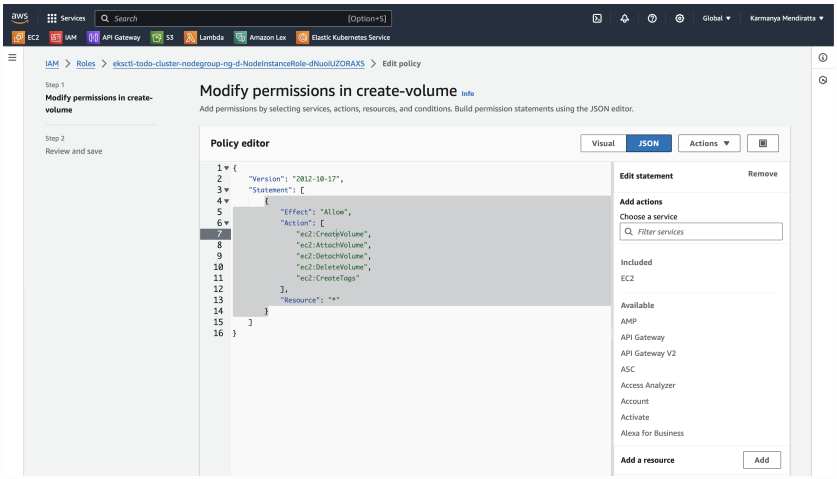
After this, we will first create a claim for the persistent volume, which will be fulfilled once a pod is attached to it

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-data
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

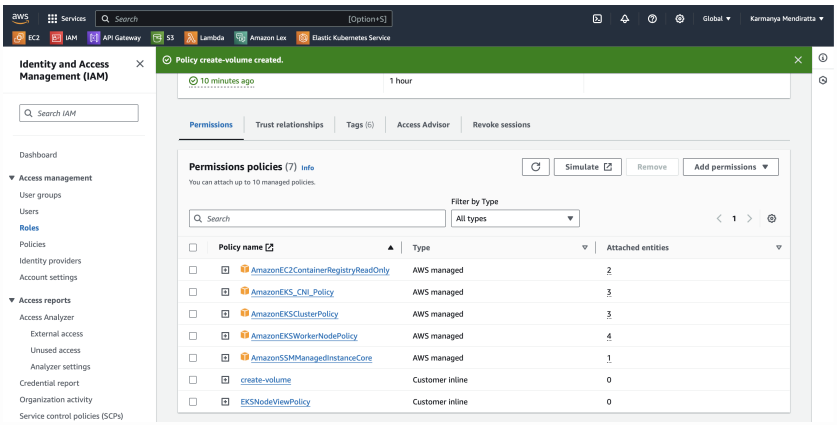
Then we need to attach the MongoDB pod to the persistent volume and mount the location where the data is saved.

```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
        image: mongo:latest
        ports:
        - containerPort: 27017
        volumeMounts:
            - name: mongo-storage
              mountPath: /data/db
      volumes:
        - name: mongo-storage
          persistentVolumeClaim:
            claimName: mongo-pvc

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

For this to work we also had to give some IAM permissions to the Node IAM Role arn. The following policies were added to our role:



Thus, the following are the policies attached to our node:



We can see that the PVC is bound to our mongo-pvc.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pvc
NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mongo-pvc                           Bound    pvc-748b4ff1-7060-4a0b-aa31-793853da26bb   1Gi        RWO            gp2            <unset>                 7s
prometheus-server                   Bound    pvc-2006dbc6-7a44-44df-9e27-d6fd99b94790   8Gi        RWO            gp2            <unset>                 7h31m
storage-prometheus-alertmanager-0   Bound    pvc-e6e8af9b-9f40-4998-9059-a8fed5a04ad9   2Gi        RWO            gp2            <unset>                 7h31m
```

Replacing Deployments with Replication Controller

To test out replication controller, we will delete the deployment that we previously applied. Here is the replication controller code that will be applied.

flask-app-rc.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: flaskapp-rc
spec:
  replicas: 6
  selector:
    app: flaskapp
  template:
    metadata:
      labels:
        app: flaskapp
    spec:
      containers:
      - name: flaskapp
        image: karmanya1804/todoapp:v2
        ports:
        - containerPort: 3000
```

Now, we will apply the replication controller:

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f flask-app-rc.yaml
replicationcontroller/flaskapp-rc created
```

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS       AGE
flask-app-deployment-7dc476bc58-6cfk4   1/1     Running   7 (175m ago)   3h1m
flask-app-deployment-7dc476bc58-kvf7h   1/1     Running   7 (175m ago)   3h1m
flaskapp-rc-bwmqz                       1/1     Running   0              44s
flaskapp-rc-rbm47                       1/1     Running   0              44s
flaskapp-rc-rvxh7                       1/1     Running   0              44s
mongodb-deployment-7cfc99f98-8k98h      1/1     Running   0              3h1m
```

In the above screenshot, we can see that the desired number of requested pods is 3, and currently, 3 pods are running.

On running kubectl get replicationcontroller, we can see that the replication controller is created with the 3 desired pods

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get replicationcontroller
NAME          DESIRED   CURRENT   READY   AGE
flaskapp-rc   3         3         3       58m
```

Deleting one of the pods

Now let's delete one of the replicas and see whether the replication controller is keeping the desired number of replicas or not.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl delete pod flaskapp-rc-rvxh7
pod "flaskapp-rc-rvxh7" deleted
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS       AGE
flask-app-deployment-7dc476bc58-6cfk4   1/1     Running   7 (177m ago)   3h3m
flask-app-deployment-7dc476bc58-kvf7h   1/1     Running   7 (177m ago)   3h3m
flaskapp-rc-bwmqz                     1/1     Running   0              2m43s
flaskapp-rc-h86wv                     1/1     Running   0              3s
flaskapp-rc-rbm47                     1/1     Running   0              2m43s
mongodb-deployment-7cfc99f98-8k98h    1/1     Running   0              3h3m
```

On deleting the pod flaskapp-rc-rvxh7, the replication controller instantly created a new pod and replaced the delete one

{{< pagebreak >}}

Updating the number of replicas

Specifying 6 replicas for the Flask app.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get replicationcontroller
NAME          DESIRED   CURRENT   READY   AGE
flaskapp-rc   6         6         6       59m
```

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f flask-app-rc.yaml
replicationcontroller/flaskapp-rc configured
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS       AGE
flask-app-deployment-7dc476bc58-6cfk4   1/1     Running   7 (178m ago)   3h4m
flask-app-deployment-7dc476bc58-kvf7h   1/1     Running   7 (178m ago)   3h4m
flaskapp-rc-2pjch                     1/1     Running   0              3s
flaskapp-rc-bwmqz                     1/1     Running   0              4m17s
flaskapp-rc-dj99n                     1/1     Running   0              3s
flaskapp-rc-fvkq7                     1/1     Running   0              3s
flaskapp-rc-h86wv                     1/1     Running   0              97s
flaskapp-rc-rbm47                     1/1     Running   0              4m17s
mongodb-deployment-7cfc99f98-8k98h    1/1     Running   0              3h4m
```

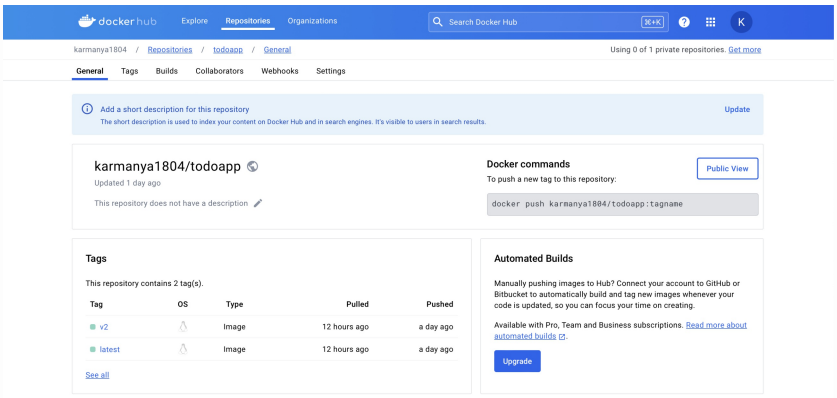
We can see that the number of replication went up to 6 hence our replication controller is working as expected.

Performing rolling update

First, we need to push a new version of the image. Here I'm creating a new tag called 'v2' and pushing it to dockerhub.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ docker tag karmanya1804/todoapp:latest karmanya1804/todoapp:v2
Karmanyas-MBP:todo_app karmanyamendiratta$ docker push karmanya1804/todoapp:v2
The push refers to repository [docker.io/karmanya1804/todoapp]
03f7306b0bed: Layer already exists
8cfafc5dc18f: Layer already exists
33bd3a424e6e: Layer already exists
9adbc4b1428d: Layer already exists
f52093e4f67d: Layer already exists
1193f41e6b14: Layer already exists
e077e19b6682: Layer already exists
21e1c4948146: Layer already exists
68866beb2ed2: Layer already exists
e6e2ab10dba6: Layer already exists
0238a1790324: Layer already exists
v2: digest: sha256:8b83053ec874262ec7cab11be289e86605fc00cacba7572fbaa5a6855194b2e8 size: 2634
```

{{< pagebreak >}}



Then we will update the strategy in the flask-app-deployment.yaml file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: karmanya1804/todoapp:v2
        ports:
        - containerPort: 3000
        env:
        - name: MONGO_HOST
          value: mongodb
        - name: MONGO_PORT
          value: "27017"
```

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl describe deployment flask-app-deployment
Name:                   flask-app-deployment
Namespace:              default
CreationTimestamp:      Sun, 17 Mar 2024 17:59:10 -0400
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=flask-app
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=flask-app
  Containers:
   flask-app:
    Image:      karmanya1804/todoapp:latest
    Port:       3000/TCP
    Host Port:  0/TCP
    Environment:
      MONGO_HOST:  mongodb
      MONGO_PORT:  27017
    Mounts:        <none>
  Volumes:         <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   flask-app-deployment-7dc476bc58 (2/2 replicas created)
Events:          <none>
```

We can see that the image here is karmanya1804/todoapp:latest

Now we will apply our new rolling update strategy

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f flask-app-deployment.yaml
deployment.apps/flask-app-deployment configured
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl rollout status deployment/flask-app-deployment
deployment "flask-app-deployment" successfully rolled out
```

{{< pagebreak >}}

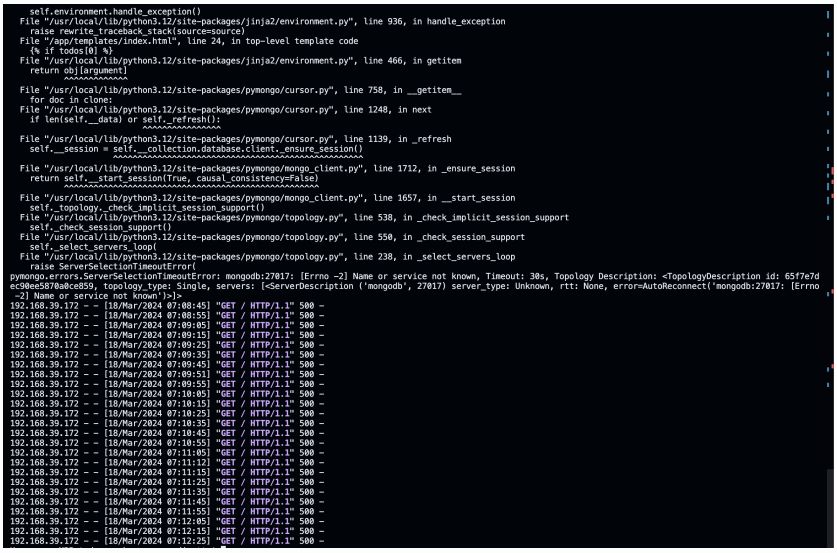
The image of the new pods are changed to karmanya1804/todoapp:v2

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-app-deployment-6765d8cb75-bmhgs   1/1     Running   0          27s
flask-app-deployment-6765d8cb75-h7n9c   1/1     Running   0          27s
flaskapp-rc-2pjch                   1/1     Running   0          31m
flaskapp-rc-bwmqz                   1/1     Running   0          35m
flaskapp-rc-dj99n                   1/1     Running   0          31m
flaskapp-rc-fvkq7                   1/1     Running   0          31m
flaskapp-rc-h86wv                   1/1     Running   0          33m
flaskapp-rc-rbm47                   1/1     Running   0          35m
mongodb-deployment-7cfc99f98-8k98h      1/1     Running   0          3h36m
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl describe deployment flask-app-deployment
Name:                   flask-app-deployment
Namespace:              default
CreationTimestamp:      Sun, 17 Mar 2024 17:59:10 -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:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  app=flask-app
  Containers:
   flask-app:
    Image:      karmanya1804/todoapp:v2
    Port:       3000/TCP
    Host Port:  0/TCP
    Environment:
      MONGO_HOST:  mongodb
      MONGO_PORT:  27017
    Mounts:         <none>
  Volumes:          <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  flask-app-deployment-7dc476bc58 (0/0 replicas created)
NewReplicaSet:   flask-app-deployment-6765d8cb75 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  35s   deployment-controller  Scaled up replica set flask-app-deployment-6765d8cb75 to 1
  Normal  ScalingReplicaSet  35s   deployment-controller  Scaled down replica set flask-app-deployment-7dc476bc58 to 1 from 2
  Normal  ScalingReplicaSet  35s   deployment-controller  Scaled up replica set flask-app-deployment-6765d8cb75 to 2 from 1
  Normal  ScalingReplicaSet  34s   deployment-controller  Scaled down replica set flask-app-deployment-7dc476bc58 to 0 from 1
```

Liveness and Readiness probes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app-deployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: karmanya1804/todoapp:v2
        ports:
        - containerPort: 3000
        env:
        - name: MONGO_HOST
          value: mongodb
        - name: MONGO_PORT
          value: "27017"
        livenessProbe:
            tcpSocket:
              port: 3000
            initialDelaySeconds: 5
            periodSeconds: 5
        readinessProbe:
            httpGet:
              path: /
              port: 3000
              httpHeaders:
            initialDelaySeconds: 10
            periodSeconds: 10
```

To test the readiness probe, I'll delete the mongo service and pod. The Flask app won't be able to reach the MongoDB instance, causing a 500 error code when the readiness probe attempts to reach it. This will result in no traffic being routed to that pod. Since all the other pods that are part of the replica set won't be able to connect to the MongoDB instance, no pod will receive any incoming traffic. Consequently, the client will receive an error indicating that the site is unreachable.



{{< pagebreak >}}

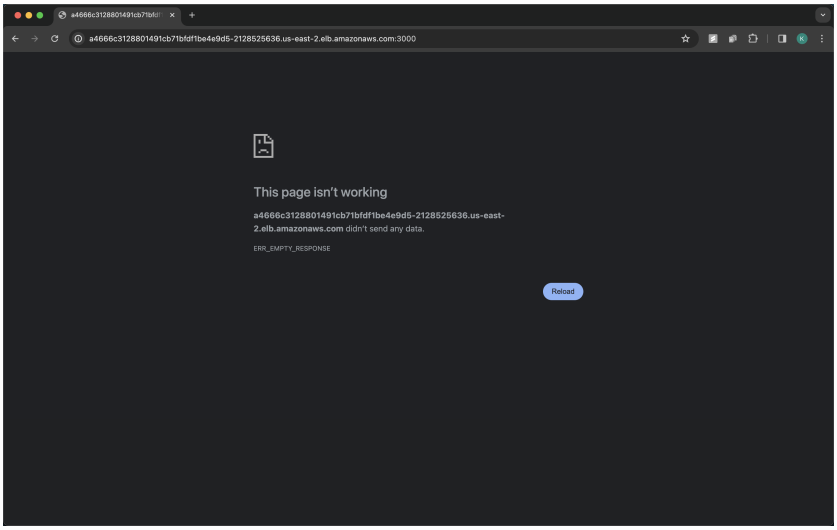
```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
flask-app-deployment-84c5db4b69-mlrh8  0/1     Running   0          7m28s
flask-app-deployment-84c5db4b69-n78h7  0/1     Running   0          7m28s
flaskapp-rc-2pjch                      1/1     Running   0          6h9m
flaskapp-rc-6cnp9                      1/1     Running   0          5h14m
flaskapp-rc-fvkq7                      1/1     Running   0          6h9m
flaskapp-rc-h86wv                      1/1     Running   0          6h11m
flaskapp-rc-lnsgt                      1/1     Running   0          5h14m
flaskapp-rc-ntzcf                      1/1     Running   0          5h14m
```

{{< pagebreak >}}

We can see that the flask app is now in not ready state and the readiness probe must also show that there is an error.

```
Containers:
  flask-app:
    Container ID:   containerd://d9b38d88cb3a741c62efe14ce2068314aa1dbd33c0754534656b587c8a4cd748
    Image:          karmanya1804/todoapp:v2
    Image ID:       docker.io/karmanya1804/todoapp@sha256:8b83053ec874262ec7cab11be289e86605fc00cacba7572fbaa5a6855194b2e8
    Port:           3000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 18 Mar 2024 03:06:05 -0400
    Ready:          False
    Restart Count:  0
    Liveness:       tcp-socket :3000 delay=5s timeout=1s period=5s #success=1 #failure=3
    Readiness:      http-get http://:3000/ delay=10s timeout=1s period=10s #success=1 #failure=3
    Environment:
      MONGO_HOST:  mongodb
      MONGO_PORT:  27017
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8k6fd (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       False
  ContainersReady             False
  PodScheduled                True
Volumes:
  kube-api-access-8k6fd:
    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
  ----     ------     ----               ----     -------
  Warning  Unhealthy  33s (x92 over 23h)  kubelet  Readiness probe failed: Get "http://192.168.47.213:3000/": context deadline exceeded (Client.Time
out exceeded while awaiting headers)
```

The application website is also down



On turning the mongoDB service back on, we can see that all the pods are back up

{{< pagebreak >}}

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb-deployment created
persistentvolumeclaim/mongo-pvc unchanged
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl apply -f mongodb-service.yaml
service/mongodb unchanged
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                             READY   STATUS             RESTARTS         AGE
faulty-pod-7bd7b48bb8-w5hb9                      0/1     CrashLoopBackOff   26 (2m39s ago)   110m
flask-app-deployment-84c5db4b69-mlrh8            1/1     Running            0                23h
flask-app-deployment-84c5db4b69-n78h7            1/1     Running            0                23h
flaskapp-rc-6cnp9                                1/1     Running            0                28h
flaskapp-rc-fvkq7                                1/1     Running            0                29h
flaskapp-rc-h86wv                                1/1     Running            0                29h
flaskapp-rc-j9x2l                                1/1     Running            0                12h
flaskapp-rc-lnsgt                                1/1     Running            0                28h
flaskapp-rc-ntzcf                                1/1     Running            0                28h
mongodb-deployment-5dc765c98f-rgjw2              1/1     Running            0                12s
prometheus-alertmanager-0                        1/1     Running            0                100m
prometheus-kube-state-metrics-75cd95986-gb7n7    1/1     Running            0                20h
prometheus-prometheus-node-exporter-6mxdc        1/1     Running            0                20h
prometheus-prometheus-node-exporter-kqld6        1/1     Running            0                20h
prometheus-prometheus-node-exporter-m6ctf        1/1     Running            0                20h
prometheus-prometheus-node-exporter-x7rx2        1/1     Running            0                20h
prometheus-prometheus-pushgateway-76976dc66-z2h7j   1/1     Running            0                20h
prometheus-server-549bfd7d89-hm9bq               2/2     Running            0                159m
```

In a similar fashion we can test for the liveness probe as well. One way could be to not start the application and the liveness probe will fail to setup tcp connection on port 5000.

Alerting with Prometheus

The procedure involves transferring metrics data from the Metrics Server to the AMP (Application Monitoring Platform) using a Prometheus server installed within Kubernetes cluster through Helm. Here is the guide on how to do this.

```
prometheus-alertmanager-0                        1/1     Running            0                106m
prometheus-kube-state-metrics-75cd95986-gb7n7    1/1     Running            0                20h
prometheus-prometheus-node-exporter-6mxdc        1/1     Running            0                20h
prometheus-prometheus-node-exporter-kqld6        1/1     Running            0                20h
prometheus-prometheus-node-exporter-m6ctf        1/1     Running            0                20h
prometheus-prometheus-node-exporter-x7rx2        1/1     Running            0                20h
prometheus-prometheus-pushgateway-76976dc66-z2h7j   1/1     Running            0                20h
prometheus-server-549bfd7d89-hm9bq               2/2     Running            0                165m
```

Now that our prometheus server is setup, we need to add alerts and setup the alert manager configs so that whenever the alert is triggered, the prometheus server hits our slack api and sends an alert message on the slack channel. To do this, first we will create a alertmanager-config.yaml, which consists of the web-hook url of slack-api, through which we will connect out Prometheus server to the slack channel.

Below is the alertmanager-config.yaml file:

Note: Make sure the following access policy is attached to the topic

```
{
    apiVersion: v1
data:
  alertmanager.yml: |
    global:
      resolve_timeout: 1m
      slack_api_url: 'https://hooks.slack.com/services/T06GC3BP0DD/B06PK
    route:
      receiver: 'slack-notifications'
      group_interval: 5m
      group_wait: 10s
      repeat_interval: 3h
    receivers:
      - name: 'slack-notifications'
        slack_configs:
          - channel: '#prometheus-alerts'
            send_resolved: true
            icon_url: https://avatars3.githubusercontent.com/u/3380462
            title: |-
              [{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .A
              {{- if gt (len .CommonLabels) (len .GroupLabels) -}}
                {{" "}}(
                {{- with .CommonLabels.Remove .GroupLabels.Names }}
                  {{- range $index, $label := .SortedPairs -}}
                    {{ if $index }}, {{ end }}
                    {{- $label.Name }}="{{ $label.Value -}}"
                  {{- end }}
                {{- end -}}
                )
              {{- end }}
            text: >-
              {{ range .Alerts -}}
              Alert: {{ .Annotations.title }}{{ if .Labels.severity }} -
              Description: {{ .Annotations.description }}
              Details:
                {{ range .Labels.SortedPairs }} • {{ .Name }}: {{ .Value
                {{ end }}
              {{ end }}
    templates:
      - /etc/alertmanager/*.tmpl
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"alertmanager.yml":"global:\n  resolve_
    meta.helm.sh/release-name: prometheus
    meta.helm.sh/release-namespace: default
  creationTimestamp: "2024-03-18T09:58:57Z"
  labels:
    app.kubernetes.io/instance: prometheus
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: alertmanager
    app.kubernetes.io/version: v0.27.0
    helm.sh/chart: alertmanager-1.9.0
  name: prometheus-alertmanager
  namespace: default
  resourceVersion: "351293"
  uid: 8fd7a888-54be-4266-a0ed-8cf9ee91428a

}
```

In this file, we have fetched the alert manager configs already present on the prometheus service and added our own configs to it. We have added the configs in 'alertmanager.yml:' value in the data field. The config specifies the slack-api url, the channel name, the receiver name and other essential configs required to send a notification to slack.

Once prometheus server is setup, next we need to create the alert definition to send the alerts to SNS topic.

```
alertmanager_config:
  route:
    receiver: 'sns-receiver'
  receivers:
    - name: 'sns-receiver'
      sns_configs:
      - topic_arn: 'arn:aws:sns:us-east-2:197499403368:amp-alerts'
        sigv4:
          region: us-east-2
        subject: 'amp alert'
```

Next, We will look into prometheus-config.yaml file, specifically the rules section which defines our alerting rules, which if broken, will lead to the alert getting triggered and the alertmanager will then send a slack notification.

```
rules: |
    groups:
      - name: alerting-rules
        rules:
        - alert: HighFailureRate
          annotations:
            description: '{{ $labels.pod }} has restarted {{ $value }} t
              10 minutes.'
            summary: High failure rate detected in pod {{ $labels.pod }}
          expr: increase(kube_pod_container_status_restarts_total[10m])
          for: 1m
          labels:
            severity: critical
```

This configuration sets up a critical alert in Prometheus, designed to trigger when a pod within the observed scope experiences an increase in restarts exceeding a certain threshold over a 10-minute window. Specifically, the alert is named "HighFailureRate" and will fire if there is an increase of more than one restart during the last 10 minutes. For the alert to be activated, the specified condition must be met continuously for at least 1 minute.

Now, to test that out alert is working as expected, we will create a new yaml file called faulty-pod.yaml. This file defines the configurations to create a pod that will consistently fail and restart, thus allowing us to check if the alert we have created to check the number of restarts of a pod is working.

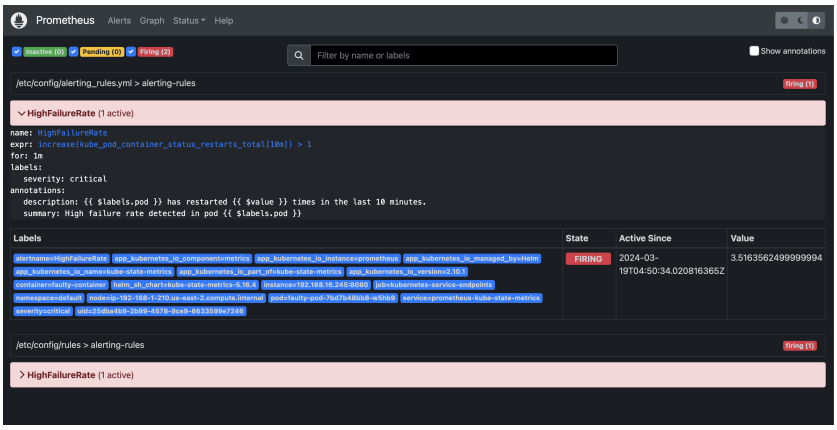
faulty-pod.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: faulty-pod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: faulty-pod
  template:
    metadata:
      labels:
        app: faulty-pod
    spec:
      containers:
      - name: faulty-container
        image: busybox
        command: ["sh", "-c", "exit 1"]
```

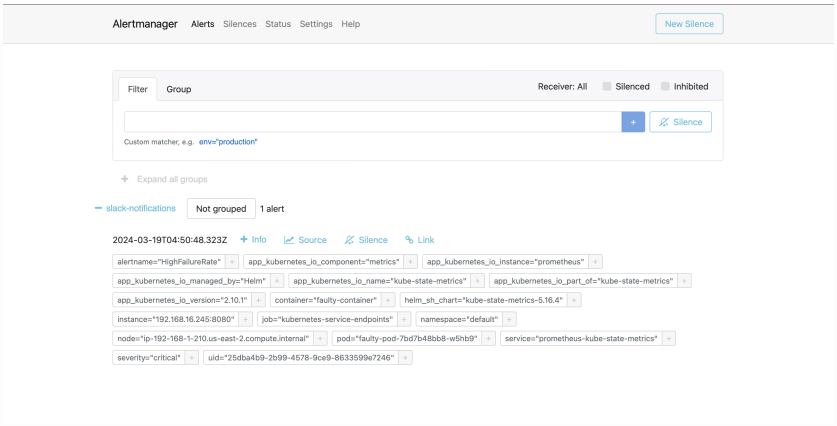
Upon applying this configuration, we can see that the pod is created and is continuously failing and retrying.

```
Karmanyas-MBP:todo_app karmanyamendiratta$ kubectl get pods
NAME                                    READY   STATUS             RESTARTS        AGE
faulty-pod-7bd7b48bb8-w5hb9             0/1     CrashLoopBackOff   7 (106s ago)    12m
```

We can see that our pod is in CrashLoopBackoff status and we can observe on our prometheus dashboard now that the alert we created earlier has started firing

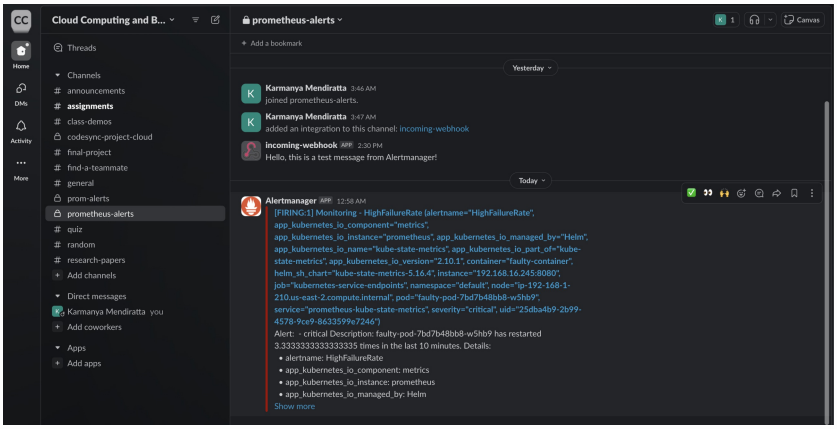


In the alertmanager dashboard, we can see that our prometheus alert has called the alertmanager successfully and now we should be getting alerts on slack.



Here is the slack message received when the faulty pod is down and the alertmanager has sent the notification:

{{< pagebreak >}}



This is the complete alert that was received in the notification:

```
[FIRING:2] Monitoring - HighFailureRate (alertname="HighFailureRate", ap
Alert:  - critical Description: faulty-pod-7bd7b48bb8-rn8r2 has restarte
  • alertname: HighFailureRate
  • app_kubernetes_io_component: metrics
  • app_kubernetes_io_instance: prometheus
  • app_kubernetes_io_managed_by: Helm
  • app_kubernetes_io_name: kube-state-metrics
  • app_kubernetes_io_part_of: kube-state-metrics
  • app_kubernetes_io_version: 2.10.1
  • container: faulty-container
  • helm_sh_chart: kube-state-metrics-5.16.4
  • instance: 192.168.16.245:8080
  • job: kubernetes-service-endpoints
  • namespace: default
  • node: ip-192-168-1-210.us-east-2.compute.internal
  • pod: faulty-pod-7bd7b48bb8-rn8r2
  • service: prometheus-kube-state-metrics
  • severity: critical
  • uid: 7b48dc81-6b97-4b7c-8cb6-206426b088ab
```