Docker + MongoDB + Flask Setup Guide

This guide explains how to prepare, install, and test a Flask application connected to MongoDB using Docker.

Preparation

1. Download the Source Code

Download the project source code from the provided Google Drive folder.

---

2. Install Docker (macOS)

Follow Docker Desktop installation for Mac.

Alternatively, using command line:

```
sudo hdiutil attach Docker.dmg
sudo /Volumes/Docker/Docker.app/Contents/MacOS/install
sudo hdiutil detach /Volumes/Docker
```

Check Docker version:

```
docker --version
```

---

3. Install Kubernetes Tools (Optional)

Install **kubectl** and **minikube** using Homebrew:

```
brew install kubectl
brew install minikube
```

Verify installations:

```
kubectl --help
minikube --help
```

Test on Local Machine

1. Set Up the Environment

Unzip the downloaded file, navigate into the project folder, and install dependencies:

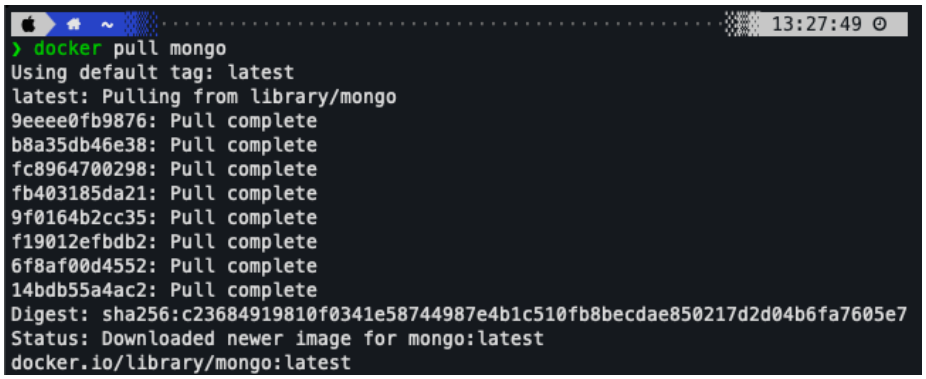
```
pip install -r requirements.txt
```

---

2. Run MongoDB Using Docker

Pull the official MongoDB image:

```
docker pull mongo
```



Run the MongoDB container:

```
docker run -d   --name mongodb   -p 27017:27017   -v mongo_data:/data/db   mongo
```

Explanation: - `-d` → Run in detached mode (background) - `--name mongodb` → Container name - `-p 27017:27017` → Expose MongoDB default port - `-v mongo_data:/data/db` → Persist data using Docker volume - `mongo` → Use the official MongoDB image

Check if the container is running:

```
docker ps
```

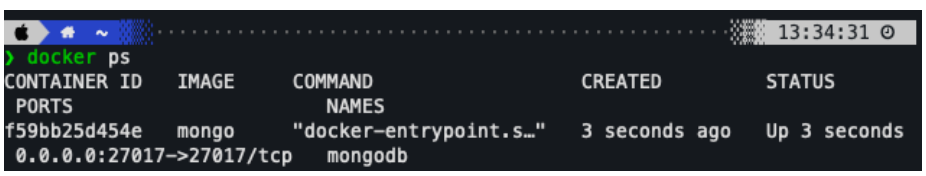


Figure 1: Docker PS Mongo

---

3. MongoDB Configuration in `app.py`

The Flask app connects to MongoDB as follows:

```
mongodb_host = os.environ.get('MONGO_HOST', 'localhost')
mongodb_port = int(os.environ.get('MONGO_PORT', '27017'))
client = MongoClient(mongodb_host, mongodb_port)
db = client.camp2016
todos = db.todo
```

---

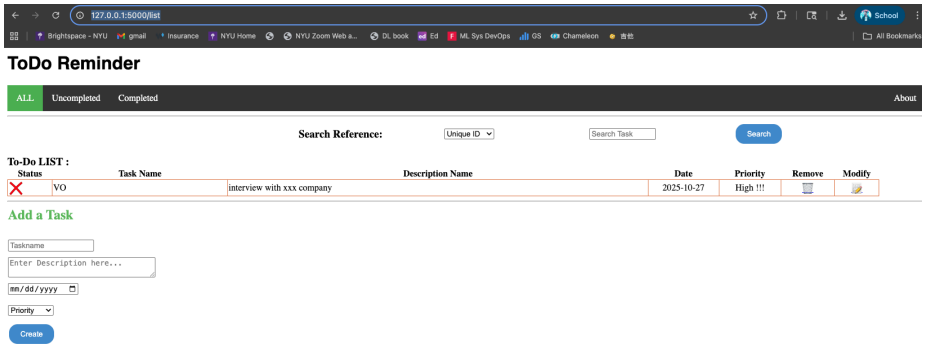
4. Start the Flask Application

Run the Flask app:

```
python3 app.py
```

Then open your browser and go to:

http://127.0.0.1:5000



to verify the application is running locally.

Inspect Docker Volumes

To check your MongoDB volume:

```
docker volume inspect mongo_data
docker volume ls
```

```
13:50:26
❯ docker volume inspect mongo_data
[
    {
        "CreatedAt": "2025-10-14T17:34:31Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mongo_data/_data",
        "Name": "mongo_data",
        "Options": null,
        "Scope": "local"
    }
]
```

```
14:05:52
❯ docker volume ls
DRIVER    VOLUME NAME
local     ad5e9c4af71df7749ded8a0adb8ac6d819e951c9bccda34615133fcb9fb3c748
local     mongo_data
```

Inspect MongoDB Data

Access MongoDB shell inside the container:

```
docker exec -it mongodb mongosh
```

Then run:

```
use camp2016                # switch to database
show collections            # list collections
db.todo.find().pretty()     # view data inside 'todo' collection
```

You've successfully set up Docker, MongoDB, and Flask locally!

Containize

1.Create a Dockfile for Flask App

```
# Use the official Python 3.8 slim image as the base image
FROM python:3.8-slim

# Set the working directory within the container
WORKDIR /app

COPY requirements.txt .

# Upgrade pip and install Python dependencies
RUN pip3 install --upgrade pip && pip install --no-cache-dir -r requirements.txt

# Copy the rest of app
COPY . .
```

```
# Expose Flask port default
EXPOSE 5000

CMD ["python", "app.py"]
```

2.build the image

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

3. buil the conatiner

```
docker run -d -p 5000:5000 flask-app flask-app
```

bug

- MacOS using port 5000 for airplay, need to shut down it to use 5000 port
- pymongo.errors.ServerSelectionTimeoutError: localhost:27017: [Errno 111] Connection refused
  - inside the flask container, localhost means the Flask containner itself, not the host or Mongo cantainer.

  `client = MongoClient('localhost', 27017)`

fix it

```
- using docker-compose.yml
- for localtest manual docker run with --network
```

```
#create a bridge network name flasknet, so conatiners on the same network can communicate by
docker network create flasknet
docker run -d --name mongodb --network flasknet -p 27017:27017 mongo
docker run -d --name flask-app --network flasknet -p 5000:5000 flasknet -e MONGO_HOST=mongod
```

4. push to Docker Hub

we can create a new tag and push the image to the Docker Hub

```
docker push majiny/flask-app:latest
```

Deploy to the minikube

Since we have minikube downloaded before, so that we can create the deploy to the minikube.

1. Start minikube

using the doc to start, minikube service

```
minikube start
```

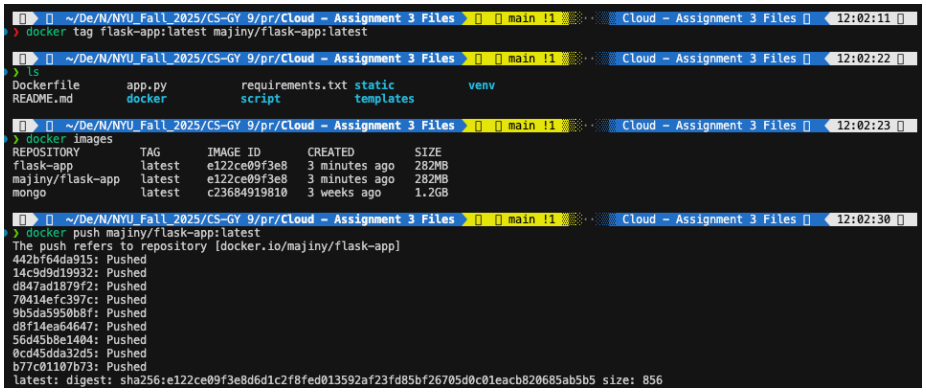


Figure 2: Push to DockerHub

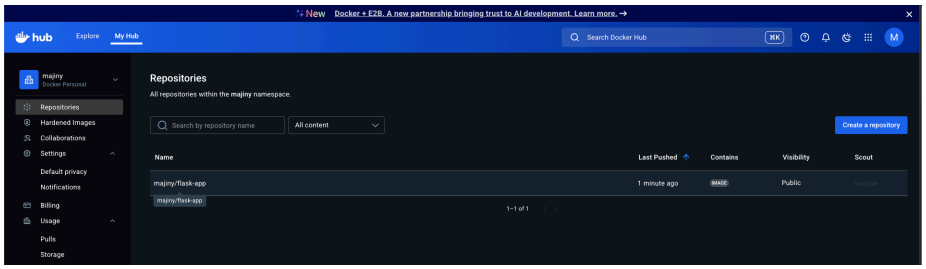


Figure 3: Image in DockerHub

2. create the yaml file

```
1. mongodb-pvc.yaml - Persistent Storage for MongoDB:
    - persistentVolumeClaim(PVC)-> A request for storage space in Kubernetes.
    - accessModes: ReadWriteOnce-> The volumen can be mounted as read-write by only one node
    - resources.requests.storage-> Requests 256 MB of space.,
MongoDB stores its database files under /data/db. Without a PVC, that data disappears when t
2. mongodb-deployment.yaml - Running MongoDB Pod:
    - kind: Deployment-> Ensures a specific # of identical pods run continuously.
    - metadata.name-> Deployment name (mongodb-deployment).
    - selector/labels-> Used to connect pods, services, and deployments.
    - containers-> Defines which image to run (we used mongo:latest).
    - containerPort-> MongoDB listens on port 27017 (defauly port for MongoDB).
    - aresources.requests-> Reserves 256 MB memory and 0.5 CPU for Mongo.
    - volumeMounts-> Mounts the PVC inside the container at /data/db.
    - volumes-> Attaches the PVC created earlier (mongodb-pvc)
MongoDB Deployment spins up MongoDB and mounts a persistent volume for storage. Even if the
3. mongodb-service.yaml - Internal Service for MongoDB:
    - kind: Service-> exposes a stable endpoint (DNS name) for Mongo pods.
    - selector.app: mongo-> Connects this Service to pods with label app: mongo.
    - port/ targetPort-> Routes traffic from Service port 27017-> pod port 27017.
Flask can not directly know a pod's IP (pods are dynamic). Instead, it connects to mongo-ser
4. flask-deployment.yaml - Flask App Deployment:
    - replicas: 1 -> Run on Flask pod (for Minikube demon).
    - images: majiny/flask-app-> The flask app we pushed to Docker Hub before.
    - containerPort: 5000 -> Flask runs internally on port 5000.
    - en variables -> Tell Flask where to find MongoDB.
        - MONGO_HOST=mongo-service
        - MONGO_PORT=27017
    - imagePullyPolicy: IfNotPresent->Uses a cached local image if available
This defines the applicaiton container. It runs Flask, connects to MongoDB vai service name,
5. flask-service.yaml - Exposing Flask to the Outside World:
    - type: LoadBalancer -> Create an external access point/ outside cluster.
    - port: 5000 -> Port exposed to the world.
    - targetPort: 5000 -> Port inside the pod where Flaks runs.
    - selector.app: to-do _> Link to Flaks pod labeld app: to-do
This lets users access ahte Flask app from outside Kubernetes.
```

3. deploy service

using the code to deploy the service.

```
kubectl apply -f mongodb-pvc.yaml
kubectl apply -f mongodb-service.yaml
kubectl apply -f mongodb-deployment.yaml
kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
```

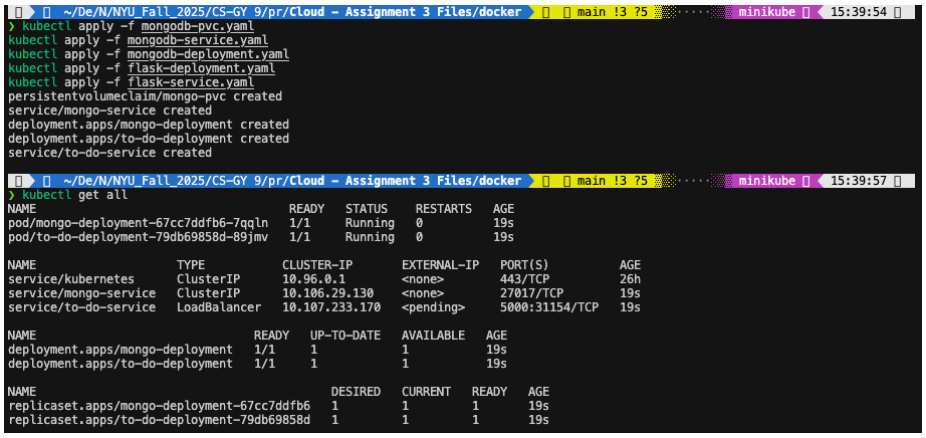


Figure 4: Deploy On Minikube

4. Test the app in Minikube

To ensure it is running on Minikube by accessing it using the service URL

`minikube service to-do-service`

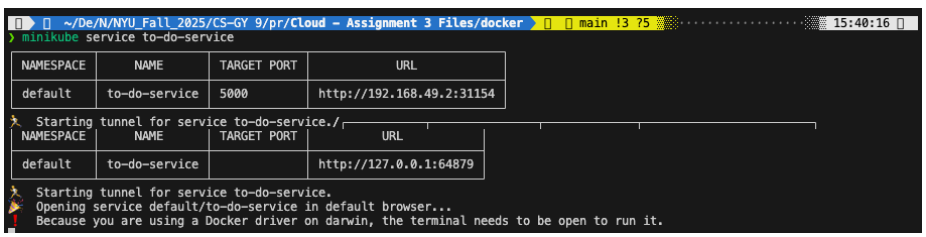
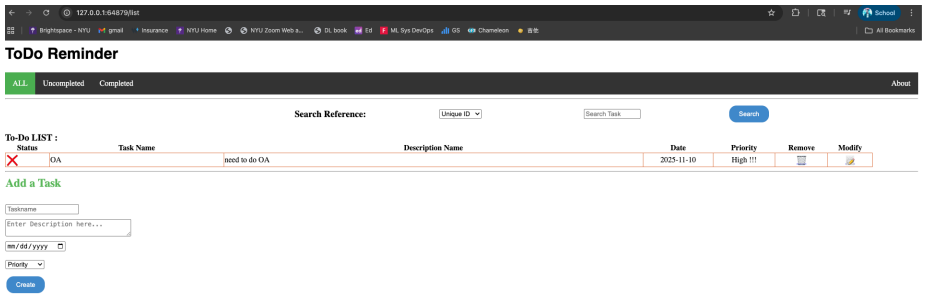


Figure 5: Minikube service

and we can see the URL address is http://127.0.0.1:64879 and we can visited the



site by using this URL.

Deploy to AWS EKS

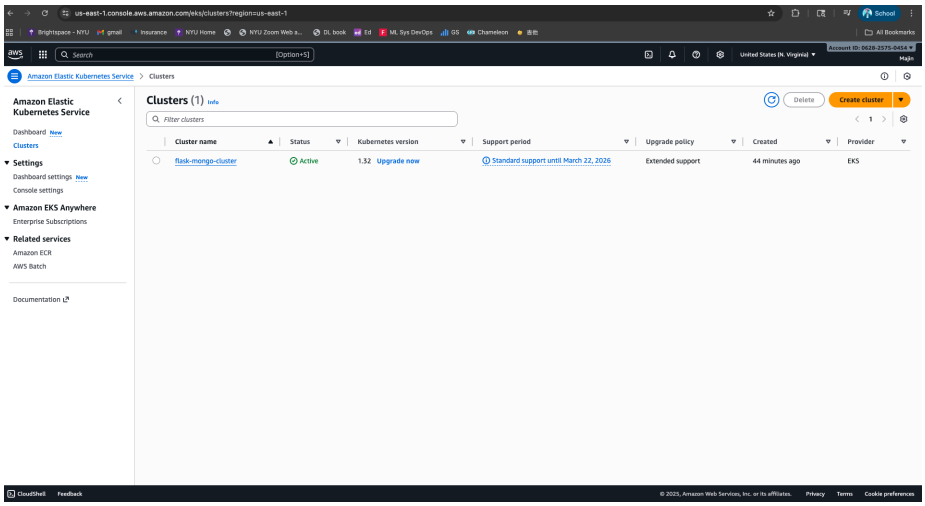
1. download AWS CLI

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /

$ which aws
/usr/local/bin/aws
$ aws --version
aws-cli/2.31.9 Python/3.13.7 Darwin/25.1.0 source/arm64
```

2. Create cluster

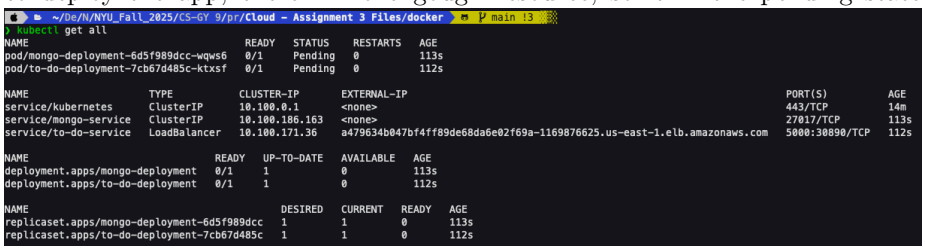
the easy we to create the cluster, we can use [eksctl][https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html]

```
eksctl create cluster \
  --name flask-mongo-cluster \
  --region us-east-1 \
  --nodes 1 \
  --node-type t3.micro
```



After launch, we can see

since there we limited cpu as 500m and memory: 256Mi. and when we trying to deploy the app, there in no engough resource, so it in the pending state



```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/mongo-deployment-6d5f989dcc-wqws6   0/1     Pending   0          113s
pod/to-do-deployment-7cb67d485c-ktxsf   0/1     Pending   0          112s

NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)          AGE
service/kubernetes    ClusterIP      10.100.0.1       <none>                                                                   443/TCP          14m
service/mongo-service ClusterIP      10.100.186.163   <none>                                                                   27017/TCP        113s
service/to-do-service LoadBalancer   10.100.171.36    a479634b047bf4ff89de68da6e02f69a-1169876625.us-east-1.elb.amazonaws.com  5000:30890/TCP   112s

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongo-deployment      0/1     1            0           113s
deployment.apps/to-do-deployment      0/1     1            0           112s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/mongo-deployment-6d5f989dcc     1         1         0       113s
replicaset.apps/to-do-deployment-7cb67d485c     1         1         0       112s
```

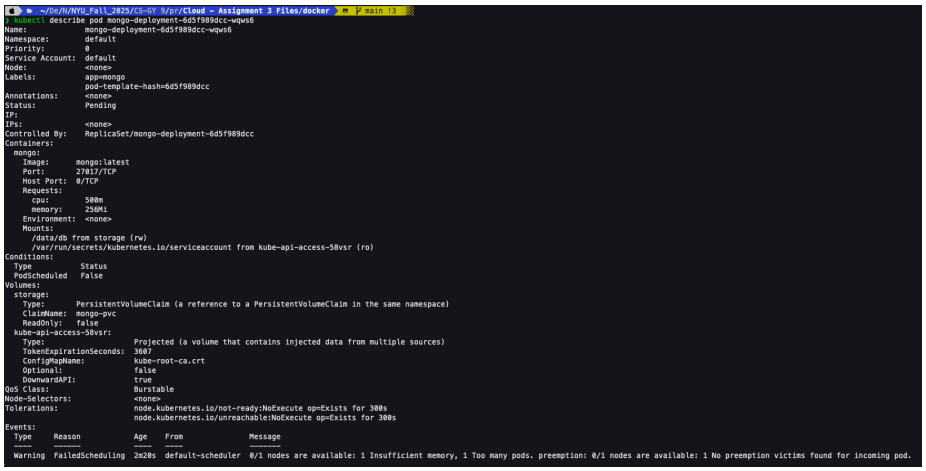
using the kubectl to check why it is pending

```
kubectl describe pod <POD_ID>
```

delete the resource we had before and rebuild

```
eksctl delete cluster --name flask-mongo-cluster --region us-east-1
```

```
eksctl create cluster \
  --name flask-mongo-cluster \
  --region us-east-1 \
  --nodegroup-name flask-nodegroup \
  --nodes 2 \
  --nodes-min 1 \
  --nodes-max 3 \
  --managed \
  --node-type t3.small
```



```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
kubectl describe pod mongo-deployment-6d5f989dcc-wqws6
Name:             mongo-deployment-6d5f989dcc-wqws6
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           app=mongo
                  pod-template-hash=6d5f989dcc
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Controlled By:    ReplicaSet/mongo-deployment-6d5f989dcc
Containers:
  mongo:
    Image:      mongo:latest
    Port:       27017/TCP
    Host Port:  0/TCP
    Requests:
      cpu:        500m
      memory:     256Mi
    Environment:  <none>
    Mounts:
      /data/db from storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-58vsr (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  storage:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  mongo-pvc
    ReadOnly:   false
  kube-api-access-58vsr:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    Optional:                false
    DownwardAPI:             true
QoS Class:                   Burstable
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  FailedScheduling  2m20s  default-scheduler  0/1 nodes are available: 1 Insufficient memory, 1 Too many pods. preemption: 0/1 nodes are available: 1 No preemption victims found for incoming pod.
```

Figure 6: kubectl_describe

3. set up addon

Since we want to use pvc storage we need to Enable OIDC provbider (required for IAM-> K8s mapping)

```
eksctl utils associate-iam-oidc-provider \
  --region us-east-1 \
  --cluster flask-mongo-cluster \
  --approve
```

then create IAM role for the EBS CSI driver

```
eksctl create iamserviceaccount \
  --region us-east-1 \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster flask-mongo-cluster \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --approve \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

Install the AWS EBS CSI driver add-on

```
aws eks create-addon \
  --region us-east-1 \
  --cluster-name flask-mongo-cluster \
  --addon-name aws-ebs-csi-driver \
  --service-account-role-arn arn:aws:iam::<id>:role/AmazonEKS_EBS_CSI_DriverRole  #use  the
```

To verify installation

```
kubectl get pods -n kube-system | grep ebs
```

```
❯ kubectl get pods -n kube-system | grep ebs
ebs-csi-controller-5d97d64f67-gqkfh   6/6     Running   0          3m43s
ebs-csi-controller-5d97d64f67-qvltw   6/6     Running   0          3m43s
ebs-csi-node-2k2dt                    3/3     Running   0          3m43s
ebs-csi-node-bhpm4                    3/3     Running   0          3m43s
```

4. deploy to eks To get pvc running we run same code we had as we deploy to the Minikube

```
kubectl apply -f mongodb-pvc.yaml
kubectl get pvc
```

```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl apply -f mongodb-pvc.yaml
persistentvolumeclaim/mongo-pvc created

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl get pbc
error: the server doesn't have a resource type "pbc"

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl get pvc
NAME        STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mongo-pvc   Pending                                      gp2            <unset>                 9s
```

we will see there is still no VOLUME, CAPACITY, ACCESS, and MODES by checking pvc mongo-pvc

```
kubectl describe pvc mongo-pvc
```

we see the message waiting for first Consumer to be created before binding

```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl describe pvc mongo-pvc
Name:          mongo-pvc
Namespace:     default
StorageClass:  gp2
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       <none>
Events:
  Type    Reason                Age                From                         Message
  ----    ------                ----               ----                         -------
  Normal  WaitForFirstConsumer  14s (x6 over 80s)  persistentvolume-controller  waiting for first consumer to be created before binding
```

After we apply mongo-deployment

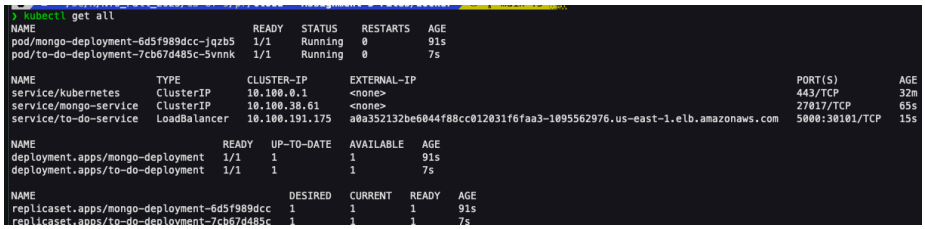
```
kubectl apply -f mongodbv-deployment.yaml
kubectl get pods
kubectl get pvc
```

```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongo-deployment created

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3
❯ kubectl get pods
kubectl get pvc
NAME                                  READY   STATUS              RESTARTS   AGE
mongo-deployment-6d5f989dcc-jqzb5     0/1     ContainerCreating   0          7s
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mongo-pvc   Bound    pvc-c3bf6222-f140-4683-a793-511a7aacd207   1Gi        RWO            gp2            <unset>                 118s
```

Figure 7: complete PVC

then we apply reset of yaml file

```
kubectl apply -f mongodb-service.yaml
kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
kubectl get all
```

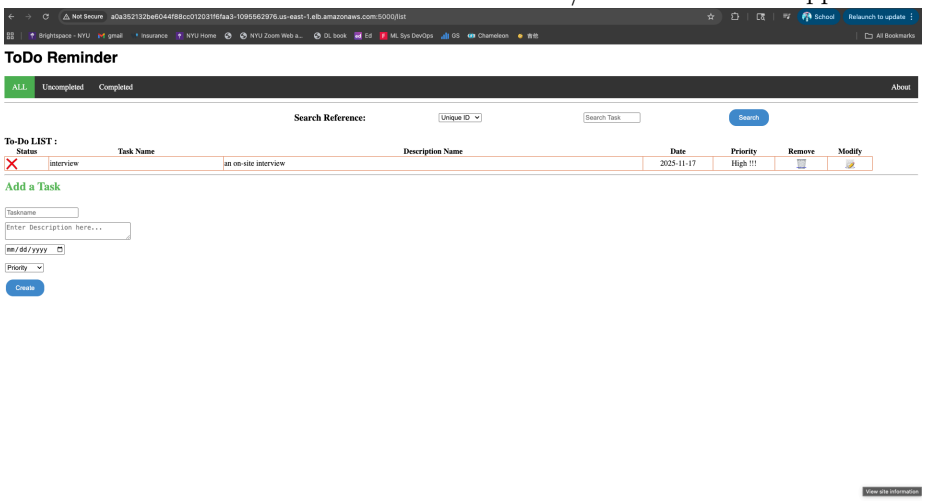


```
❯ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/mongo-deployment-6d5f989dcc-jqzb5   1/1     Running   0          91s
pod/to-do-deployment-7cb67d485c-5vnnk   1/1     Running   0          7s

NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)          AGE
service/kubernetes    ClusterIP      10.100.0.1       <none>                                                                     443/TCP          32m
service/mongo-service ClusterIP      10.100.38.61     <none>                                                                     27017/TCP        65s
service/to-do-service LoadBalancer   10.100.191.175   a0a352132be6044f88cc012031f6faa3-1095562976.us-east-1.elb.amazonaws.com   5000:30101/TCP   15s

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongo-deployment      1/1     1            1           91s
deployment.apps/to-do-deployment      1/1     1            1           7s

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/mongo-deployment-6d5f989dcc      1         1         1       91s
replicaset.apps/to-do-deployment-7cb67d485c      1         1         1       7s
```

Figure 8: k8s_aws

The all nodes are running, and we have a external-ip for our to-do-service but we can not visited yet, it takes time to set up waiting_external_ip

now we can use http://a0a352132be6044f88cc012031f6faa3-1095562976.us-east-1.elb.amazonaws.com:5000/ to visited our application



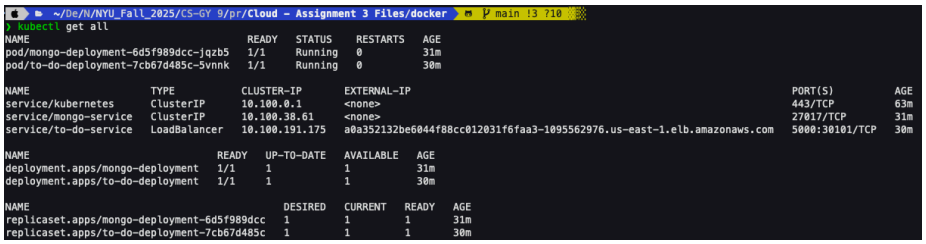
Deployments and ReplicaSets

Before we change the replicas1 we have defined replicas as 1

```
spec:
  replicas: 1
```

after we change it to 5

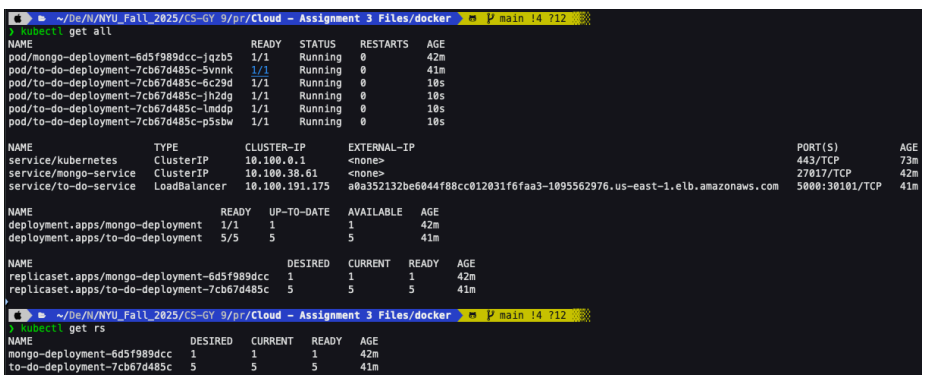
```
spec:
```



```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !3 ?10
❯ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/mongo-deployment-6d5f989dcc-jqzb5   1/1     Running   0          31m
pod/to-do-deployment-7cb67d485c-5vnnk   1/1     Running   0          30m

NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)          AGE
service/kubernetes      ClusterIP      10.100.0.1       <none>                                                                     443/TCP          63m
service/mongo-service   ClusterIP      10.100.38.61     <none>                                                                     27017/TCP        31m
service/to-do-service   LoadBalancer   10.100.191.175   a0a352132be6044f88cc012031f6faa3-1095562976.us-east-1.elb.amazonaws.com   5000:30101/TCP   30m

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongo-deployment       1/1     1            1           31m
deployment.apps/to-do-deployment       1/1     1            1           30m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/mongo-deployment-6d5f989dcc       1         1         1       31m
replicaset.apps/to-do-deployment-7cb67d485c       1         1         1       30m
```

Figure 9: get_all_re1

```
  replicas: 5

kubectl apply -f flask-deployment.yaml
# Verify the ReplicasSet
kubectl get all
kubectl get rs
```



```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?12
❯ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/mongo-deployment-6d5f989dcc-jqzb5   1/1     Running   0          42m
pod/to-do-deployment-7cb67d485c-5vnnk   1/1     Running   0          41m
pod/to-do-deployment-7cb67d485c-6c29d   1/1     Running   0          10s
pod/to-do-deployment-7cb67d485c-jh2dg   1/1     Running   0          10s
pod/to-do-deployment-7cb67d485c-lmddp   1/1     Running   0          10s
pod/to-do-deployment-7cb67d485c-p5sbw   1/1     Running   0          10s

NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)          AGE
service/kubernetes      ClusterIP      10.100.0.1       <none>                                                                     443/TCP          73m
service/mongo-service   ClusterIP      10.100.38.61     <none>                                                                     27017/TCP        42m
service/to-do-service   LoadBalancer   10.100.191.175   a0a352132be6044f88cc012031f6faa3-1095562976.us-east-1.elb.amazonaws.com   5000:30101/TCP   41m

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongo-deployment       1/1     1            1           42m
deployment.apps/to-do-deployment       5/5     5            5           41m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/mongo-deployment-6d5f989dcc       1         1         1       42m
replicaset.apps/to-do-deployment-7cb67d485c       5         5         5       41m
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?12
❯ kubectl get rs
NAME                                  DESIRED   CURRENT   READY   AGE
mongo-deployment-6d5f989dcc           1         1         1       42m
to-do-deployment-7cb67d485c           5         5         5       41m
```

Figure 10: Verify ReplicasSet

When we delete the pod we need to know the pod name

```
kubectl get pods
kubectl delete pod to-do-deployment-7cb67d485c-5vnnk to-do-deployment-7cb67d485c-6c29d
kubectl get pods
```

we will see the two pods were deleted, and by checking pods we will see two new

pods created

```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?12
❯ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mongo-deployment-6d5f989dcc-jqzb5     1/1     Running   0          47m
to-do-deployment-7cb67d485c-5vnnk     1/1     Running   0          46m
to-do-deployment-7cb67d485c-6c29d     1/1     Running   0          5m21s
to-do-deployment-7cb67d485c-jh2dg     1/1     Running   0          5m21s
to-do-deployment-7cb67d485c-lmddp     1/1     Running   0          5m21s
to-do-deployment-7cb67d485c-p5sbw     1/1     Running   0          5m21s

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?13
❯ kubectl delete pod to-do-deployment-7cb67d485c-5vnnk to-do-deployment-7cb67d485c-6c29d
pod "to-do-deployment-7cb67d485c-5vnnk" deleted from default namespace
pod "to-do-deployment-7cb67d485c-6c29d" deleted from default namespace

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?13
❯ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mongo-deployment-6d5f989dcc-jqzb5     1/1     Running   0          48m
to-do-deployment-7cb67d485c-9gtjd     1/1     Running   0          9s
to-do-deployment-7cb67d485c-fddc5     1/1     Running   0          9s
to-do-deployment-7cb67d485c-jh2dg     1/1     Running   0          6m31s
to-do-deployment-7cb67d485c-lmddp     1/1     Running   0          6m31s
to-do-deployment-7cb67d485c-p5sbw     1/1     Running   0          6m31s
```

we can scale down the replicas without changing yaml file

`kubectl scale deployment to-do-deployment --replicas=3`

```
~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?14
❯ kubectl scale deployment --replicas=3
error: resource(s) were provided, but no name was specified

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?14
❯ kubectl scale deployment to-do-deployment --replicas=3
deployment.apps/to-do-deployment scaled

~/De/N/NYU_Fall_2025/CS-GY 9/pr/Cloud - Assignment 3 Files/docker  main !4 ?14
❯ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mongo-deployment-6d5f989dcc-jqzb5     1/1     Running   0          53m
to-do-deployment-7cb67d485c-jh2dg     1/1     Running   0          11m
to-do-deployment-7cb67d485c-lmddp     1/1     Running   0          11m
to-do-deployment-7cb67d485c-p5sbw     1/1     Running   0          11m
```

Figure 11: CLI scaled down