Arpit Singh

19BCG10069

Kaiburr – Task 3

Technical Task

(Placement)

GitHub Link: https://github.com/TSM-ArpitSG/Kaiburr/tree/main/Kaibur_Tasks/Task3

Task 3:

Kubernetes:

Use the application that you created in task #1 or task #2. Create dockerfiles and build docker images. Create kubernetes yaml manifests for the application (at least a deployment and a service). It's ok to expose the application with a LoadBalancer or NodePort service or with an ingress. Spin up a single-node local Kubernetes cluster (Docker Desktop, Kind or Minikube) or use a managed cluster like EKS, AKS, GKE etc. Deploy MongoDB to the cluster (it's ok to use a community helm chart for this, any other approach is fine as well). Then deploy the application to the cluster by applying your manifests. The following requirements should be fulfilled:

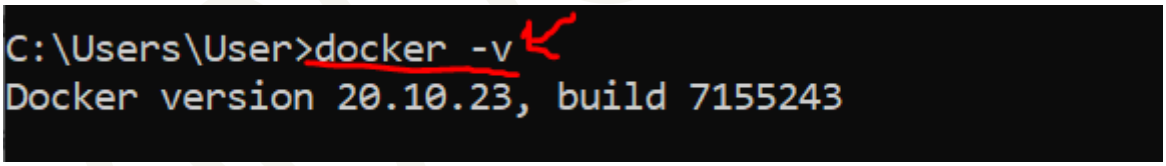
- you can bring your application up by applying your yaml manifests
- mongodb is running in a separate pod
- the application should take mongo connection details from the environment variables
- the app endpoints should be available from your host machine
- a persistent volume should be used to store the MongoDB data. I.e., when you delete the MongoDB pod the records in the db should not disappear.

Steps-(https://hub.docker.com/repository/docker/arpitsh/myapp/general):

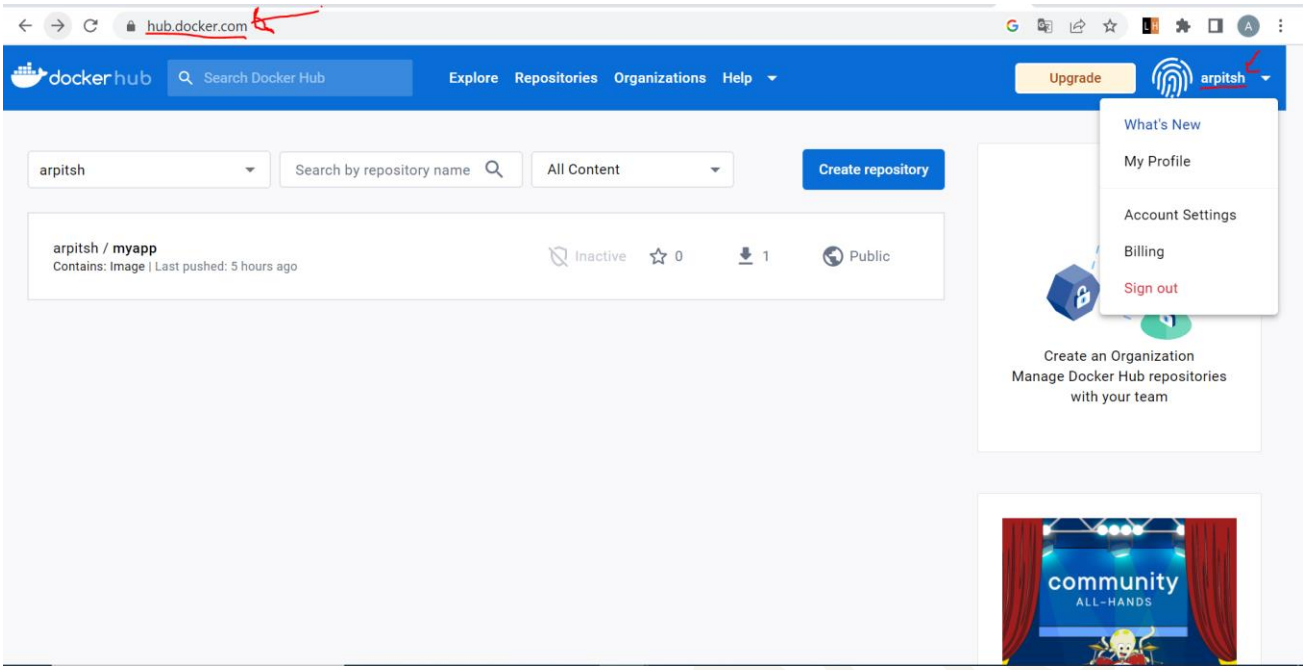
1. **Create Dockerfiles**:
   To ***containerize your application***, you need to create ***Dockerfiles that specify the image configuration***. You need to create "***two Dockerfiles***": one for your '***application***' and one for the '***MongoDB***' instance that you will run in a separate pod. The Dockerfile for your application should include the necessary ***dependencies, environment variables, and any other configuration*** files that your application requires. The Dockerfile for MongoDB should include the necessary ***configuration files and the MongoDB server***.

   a. ***Download Docker***:
      i. Go to the official Docker website at "***https://www.docker.com/get-started***" and click on the "***Download***" button.
      ii. Choose your operating system ('***Windows***', macOS, or Linux) and follow the instructions to download the '***Docker installer***' for your OS.
      iii. Once the installer is downloaded, run it and follow the installation prompts to complete the installation process.
      iv. Configure Env Variable to add : "***C:\Program Files\Docker***"

   b. '***Successful Installation***' of Docker can be checked as follows:



   c. ***Create a 'Docker Hub' account:***
      i. Go to the Docker Hub website at '***https://hub.docker.com/***' and click on the "***Sign up***" button.
      ii. Enter your email address and choose a password, then click on the "***Sign up for Docker Hub***" button.
      iii. Check your email for a verification link and click on it to verify your account.
      iv. Once your account is verified, you can ***log in to Docker Hub*** and start using it to share and ***store your Docker images***.



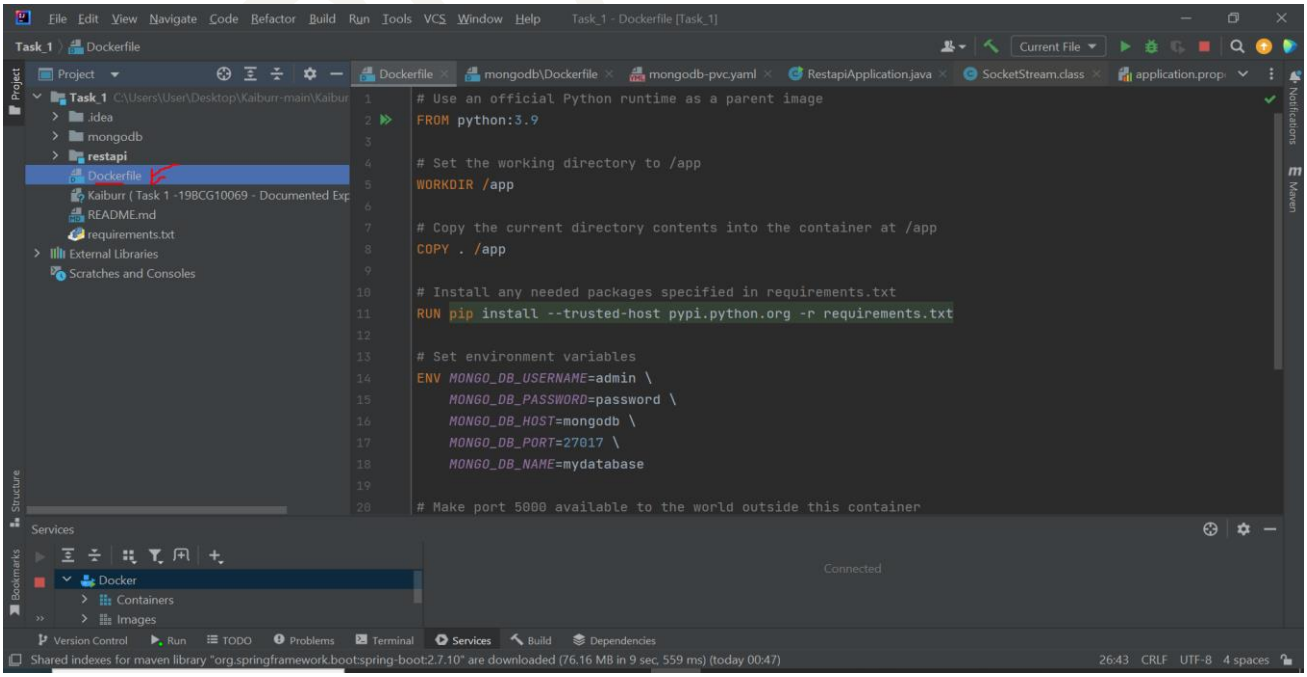
d. ***Creating the Dockerfiles:***

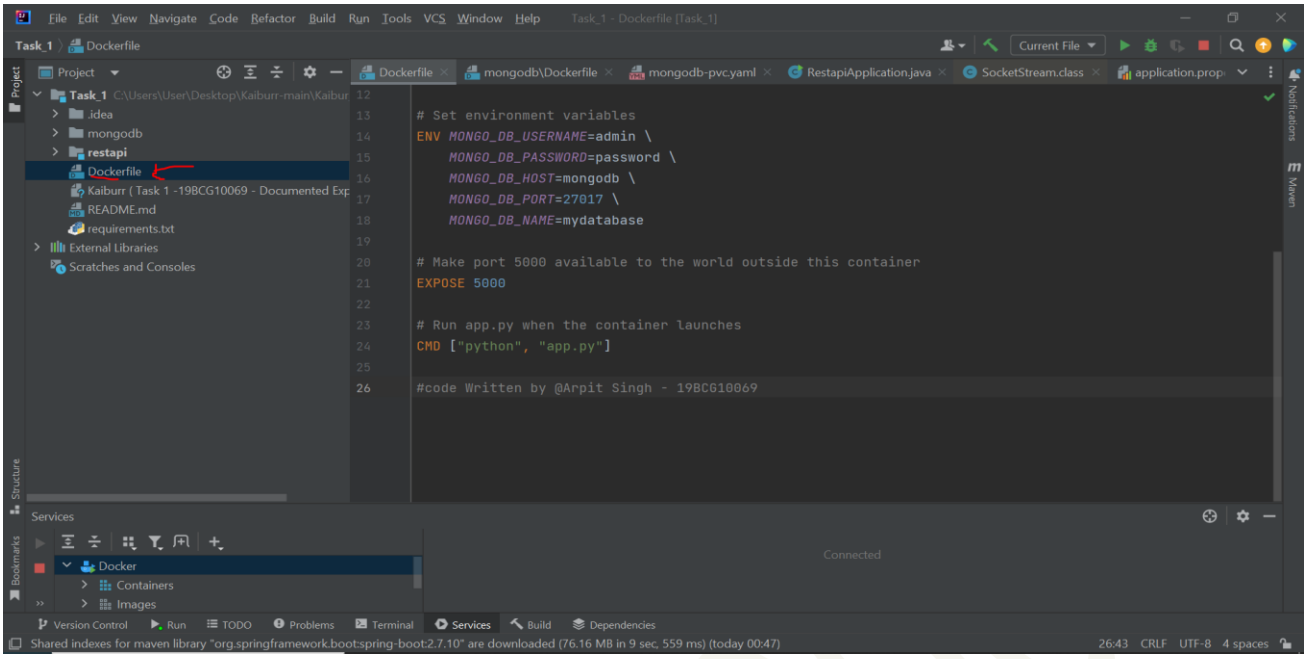
i. Open up the "***root directory***" of your application.

- In this case we are deploying the ***Rest Api*** that we created in ***Task 1***. Therefore, the root directory is like: "***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***".

ii. In this directory "***add a file named Dockerfile***":

- In this file add the '***Following code***':



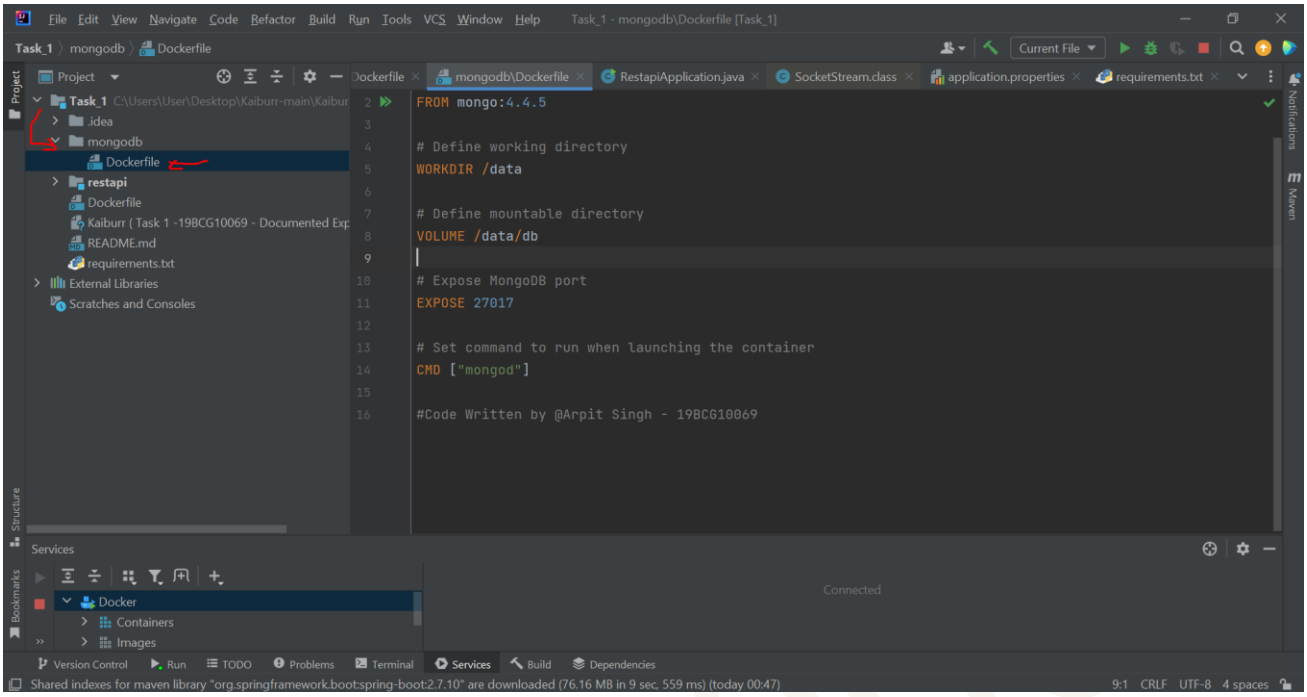


- In this file, you will define the steps required to build a ***Docker image for your application***.This Dockerfile starts with a ***base image that has Java 11*** installed (FROM ***adoptopenjdk/openjdk11:jre-11.0.12_7-alpine***). It then ***sets the working directory to /app***, copies the application jar file to the container, and sets the ***necessary environment variables for connecting to the MongoDB*** instance. It exposes ***port 8080*** and starts the application using the command ***java -jar my-application.jar.***

iii. Next we create a ***Dockerfile for MongoDB***:

- From the ***Root Directory*** "***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***" ***create another directory named "mongodb"***.
- Inside this "***mongodb***" Create a "***Dockerfile***" with the '***Following code***'

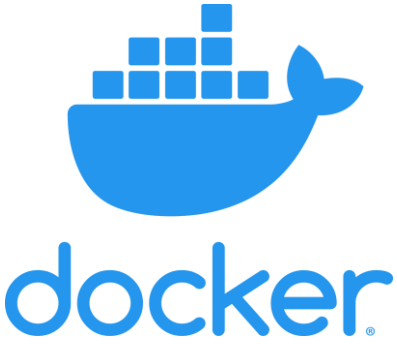
:

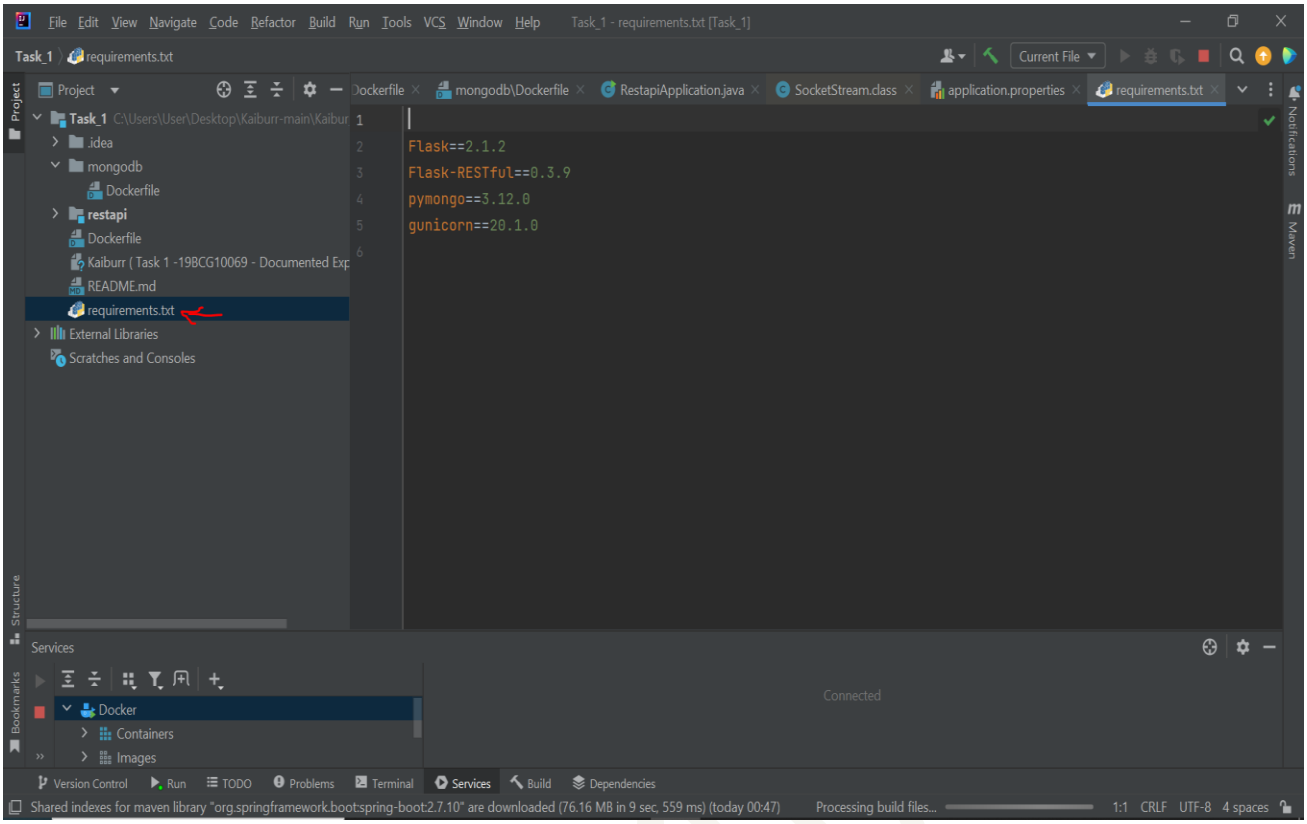


- In this file, you will define the steps required to build a ***Docker image for MongoDB***.This Dockerfile specifies that you want to use the official ***MongoDB image*** as the base image, sets the working ***directory to "/data"***, defines a mountable directory named ***"/data/db"*** (where the MongoDB data will be stored), exposes ***port 27017***, and sets "***mongod***" as the command to run when the container launches.

iv. Make sure to ***save both of these Dockerfiles*** in their respective directories.

v. Make sure to add "***Requirements.txt***" in the root folder as well. Specifying all the ***packages and dependencies*** on which our project works.





e. "***Build the Docker images***":

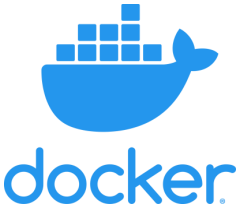
i. Open the terminal or ***command prompt*** in the root directory of your application.

ii. Navigate and run the following ***command to build the Docker image*** for the application:

- cd "***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***"

- ```
  docker build -t myapp .
  ```

- Note: ***Make sure to include the period (.)*** at the end of the command to specify the ***current directory*** as the build context.


  ```

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1

C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1>docker build -t myapp .
[+] Building 9.0s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 748B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 10.64kB
 => [1/4] FROM docker.io/library/python:3.9@sha256:af38b5d60e73a971088774fd9de87621f5425a55813970ea74357bb2de1ed246
 => CACHED [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN pip install --trusted-host pypi.python.org -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:f224c532e9e6076d6b8b49ab528e0f1018d0b2ea9b359294374e2b4d61124349
 => => naming to docker.io/library/myapp

C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1>
```

iii. Navigate to the ***/mongodb directory*** within the root directory.

- cd ***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1\mongodb***.

- `docker build -t myappservice .`

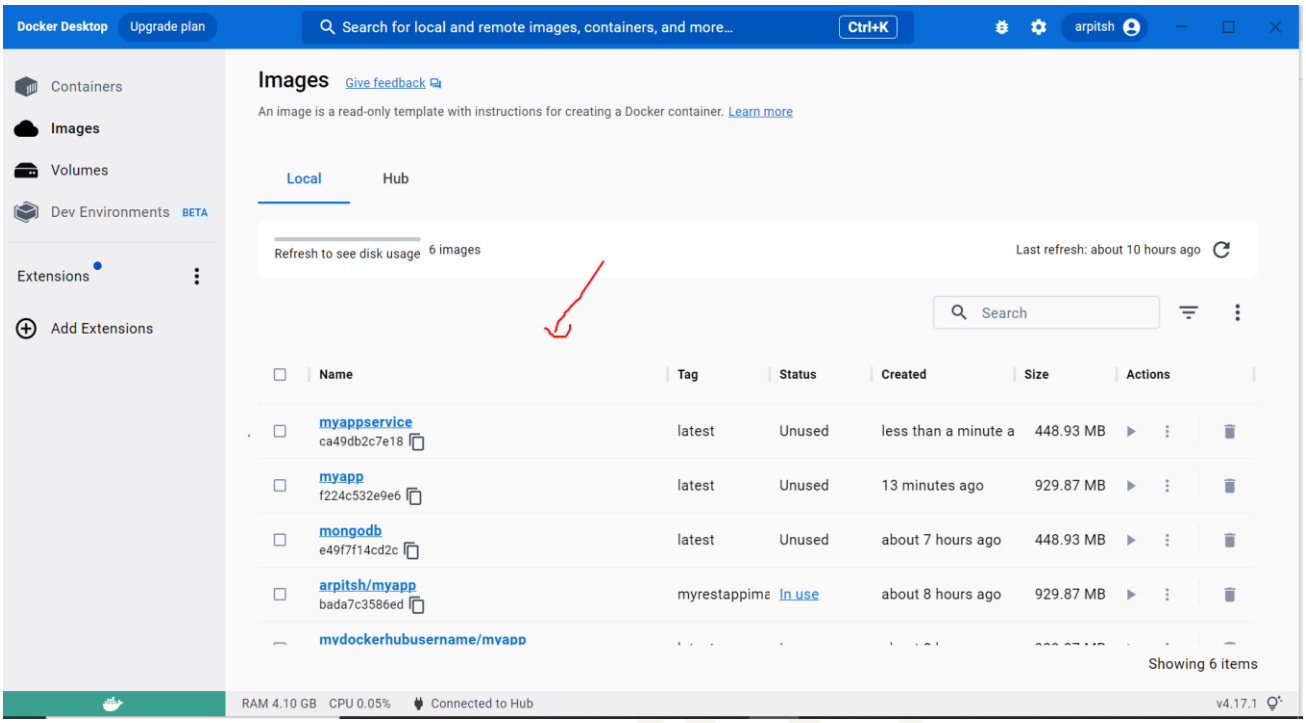
- Note: ***Make sure to include the period (.)*** at the end of the command to specify the ***current directory*** as the build context.

```
C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1>cd mongodb

C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1\mongodb>docker build -t myappservice .
[+] Building 2.5s (7/7) FINISHED
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 354B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/mongo:4.4.5                                      2.3s
 => [auth] library/mongo:pull token for registry-1.docker.io                                        0.0s
 => CACHED [1/2] FROM docker.io/library/mongo:4.4.5@sha256:0145cc5df5e657154b83fa04b90e48f6f89b608850d0906addfc0d1681a66a1e  0.0s
 => [2/2] WORKDIR /data                                                                             0.0s
 => exporting to image                                                                              0.0s
 => => exporting layers                                                                             0.0s
 => => writing image sha256:ca49db2c7e18d13729275518d44be29f0cea9b234dd4dafbcf709910c4a98a8d        0.0s
 => => naming to docker.io/library/myappservice                                                     0.0s

C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1\mongodb>
```

iv. Collection of all the "***images created***" can be viewed at '***Docker Hub***':



f. "***Publishing Docker Images in Hub***":

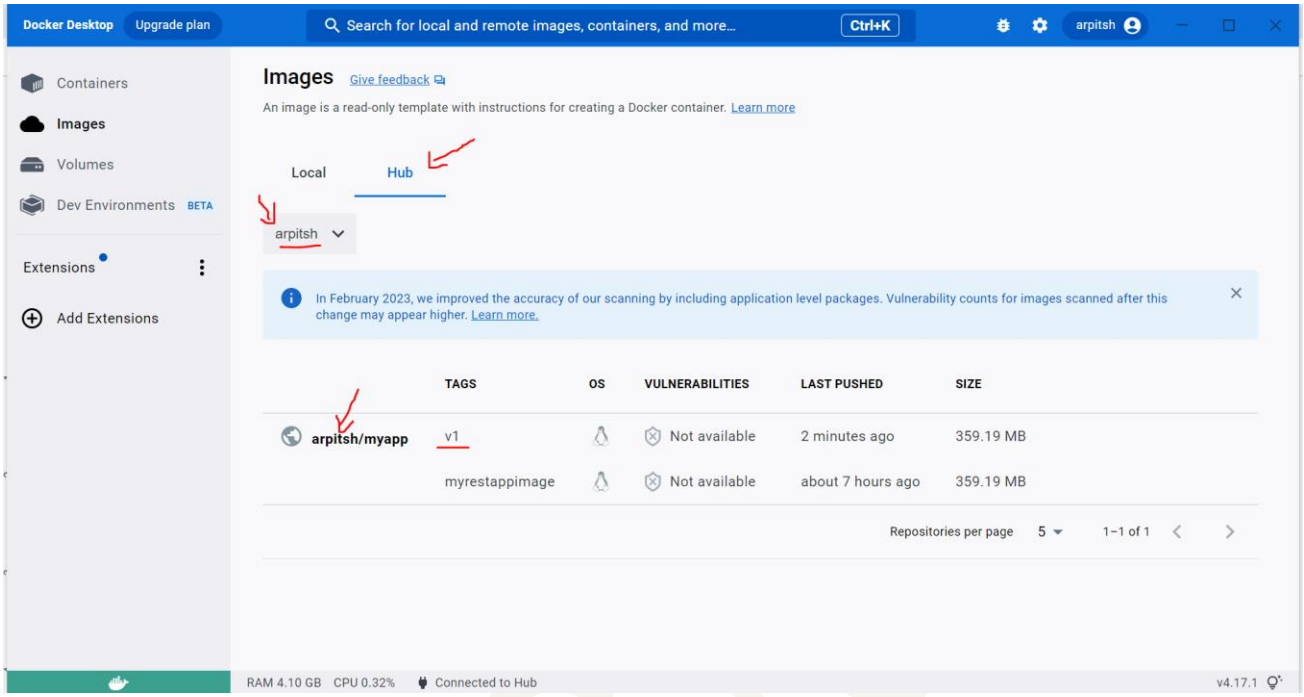
i. Once you have a Docker Hub account, log in to Docker Hub using the '***docker login command***' in your terminal. Enter your Docker Hub ***username and password*** when prompted.

ii. Next, ***tag your local Docker image*** with the name of your ***Docker Hub repository*** using the docker tag command. The format of the command is:

- `docker tag <local-image-name> <dockerhub-username>/<image-name>:<image-tag>`
- For example, if your Docker Hub username is ***arpitsh***, your image name is ***myapp***, and your tag is ***latest*** (default tag), the command would be: `docker tag myapp arpitsh/myapp:latest`
- Finally, ***push*** your Docker image to Docker Hub using the docker push command: `docker push <dockerhub-username>/<image-name>:<image-tag>`
- Using the same example as above, the command would be: `docker push arpitsh/myapp:latest`

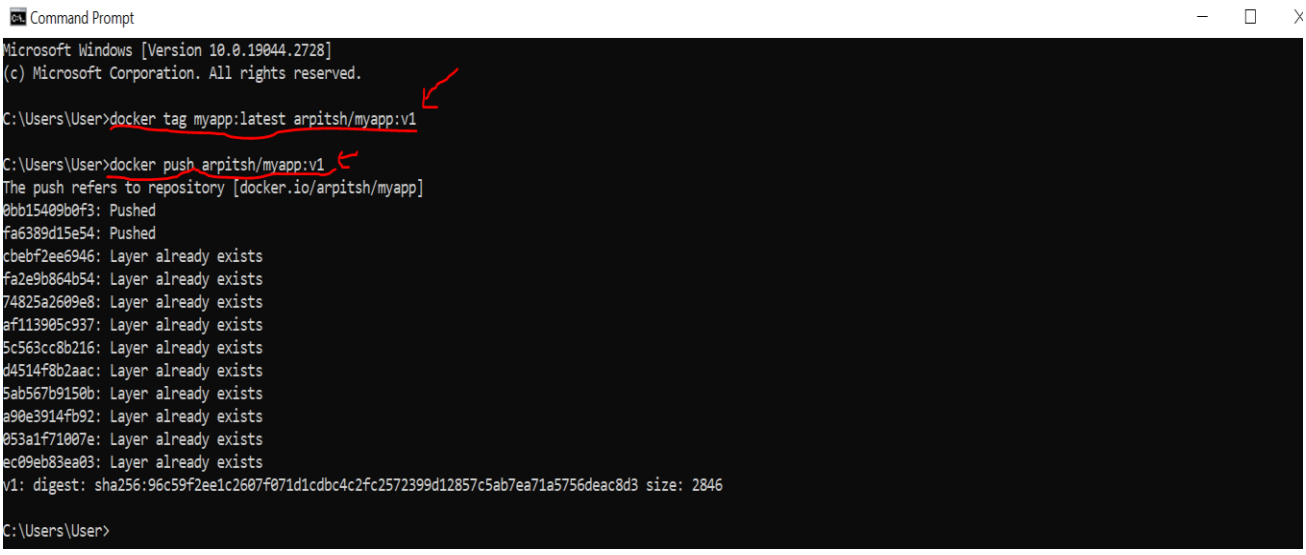
Example:

- My image name here is "***myapp***" and by default it has tag '***latest***'
  and my username is "***arpitsh***" as registered with docker cloud, and I created a '***public repository named myapp.***'



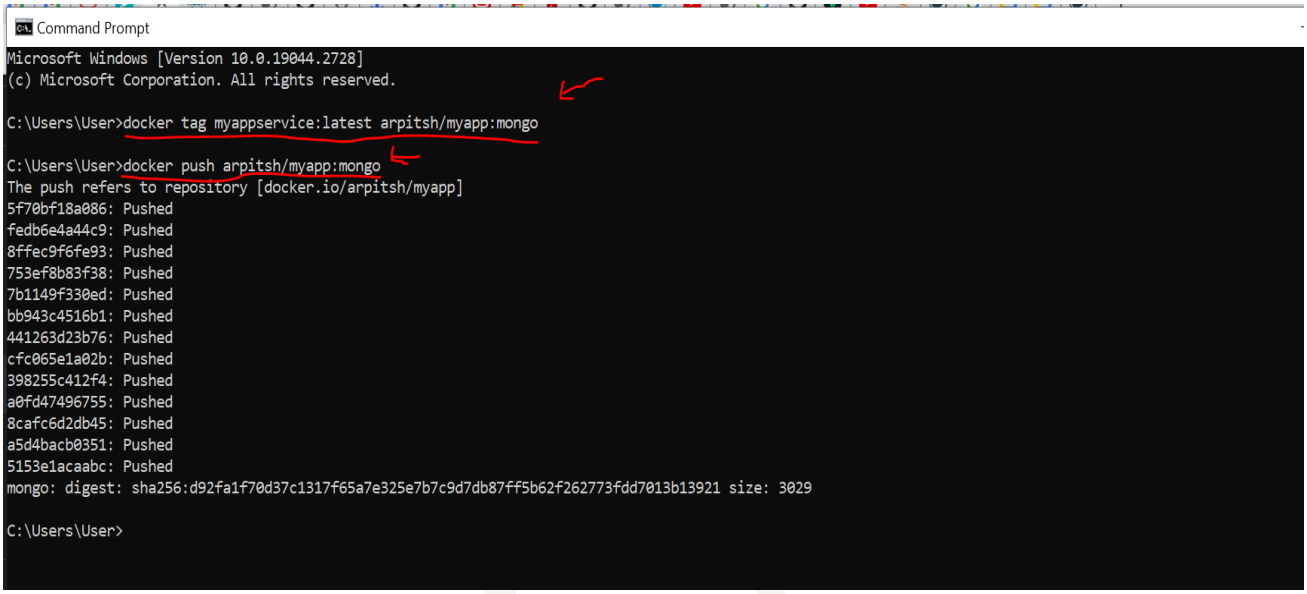
- So, my personal repository becomes now "***arpitsh/myapp***" and I want to push my image with tag "***v1***" .
- I "***tagged***"as below :
  - **docker tag myapp:latest arpitsh/myapp:v1**

- ***Pushed the image*** to my personal docker repository as below:
  - **docker push arpitsh/myapp:v1**



iii. Similarly we ***push the docker image "myappservice"*** that represents the "***mongodb Dockerfile***":

- *Tag* : **docker tag myappservice:latest arpitsh/myapp:mongo**
- *Push* :**docker push arpitsh/myapp:mongo**



2. **Create Kubernetes manifests**:

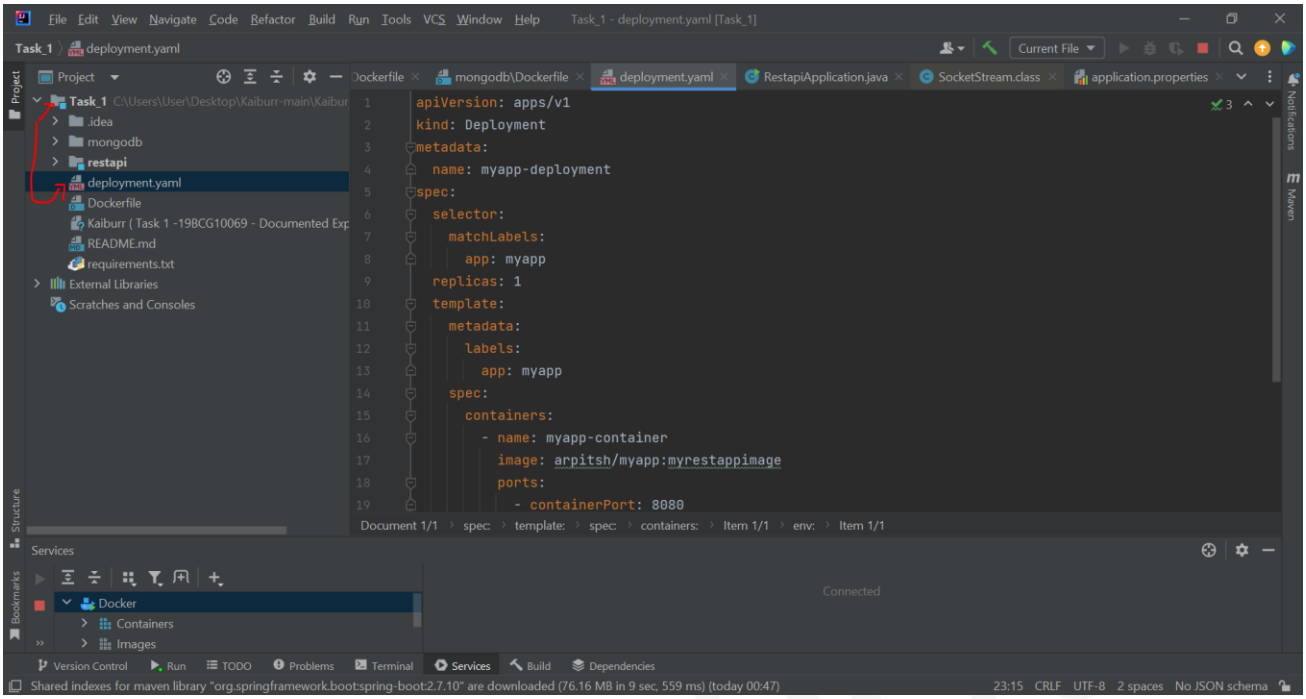
Next, you need to create "***Kubernetes manifests for your application***". You need to create at least ***a deployment and a service***. The deployment manifest specifies the ***container image, the number of replicas, and any environment variables*** that your application requires. The service manifest specifies how your application should be ***exposed to the network***. You can use either a ***LoadBalancer*** (***I used this***) or NodePort service or an Ingress.

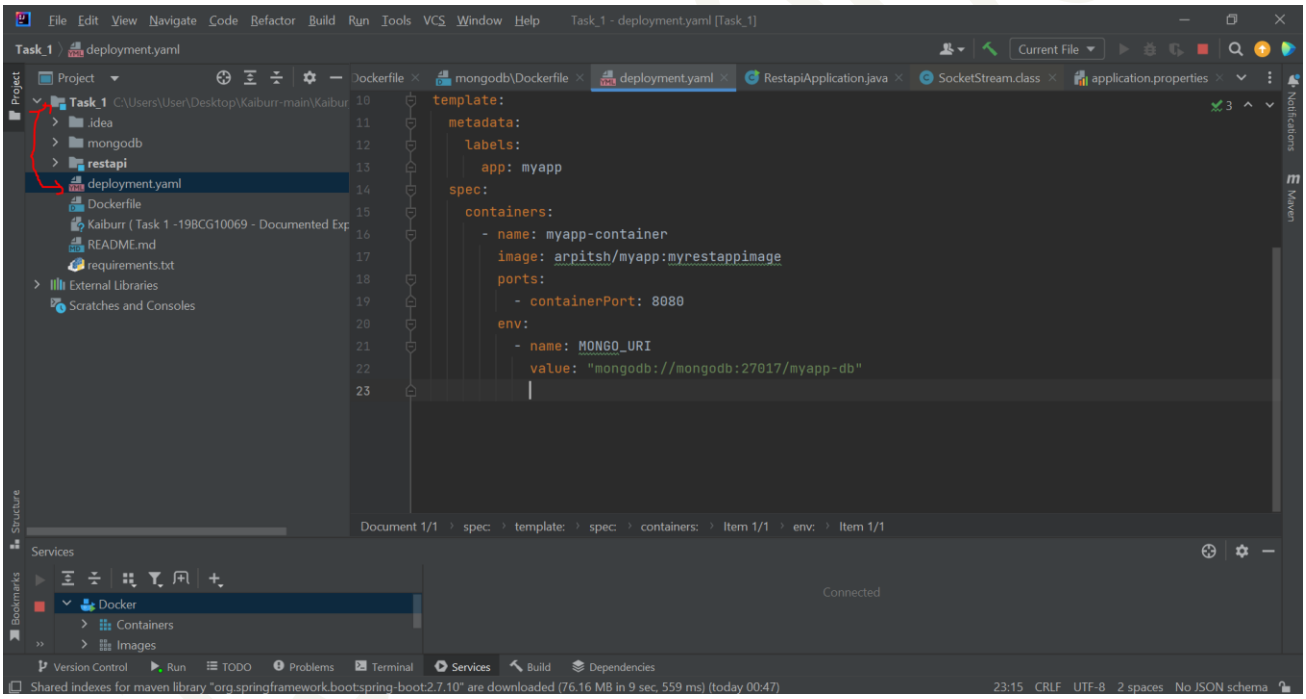
a. ***Creating 'deployment.yaml':***

i. Create a ***new file in the root directory*** of your project called '***deployment.yaml***' and open it in a ***text editor (like VsCode)***.
"***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***"

ii. Inside this "***Task1***" Create a "***deployment.yaml***" with the '***Following code***':

kubernetes





iii. ***"Save and close the deployment.yaml"*** file.

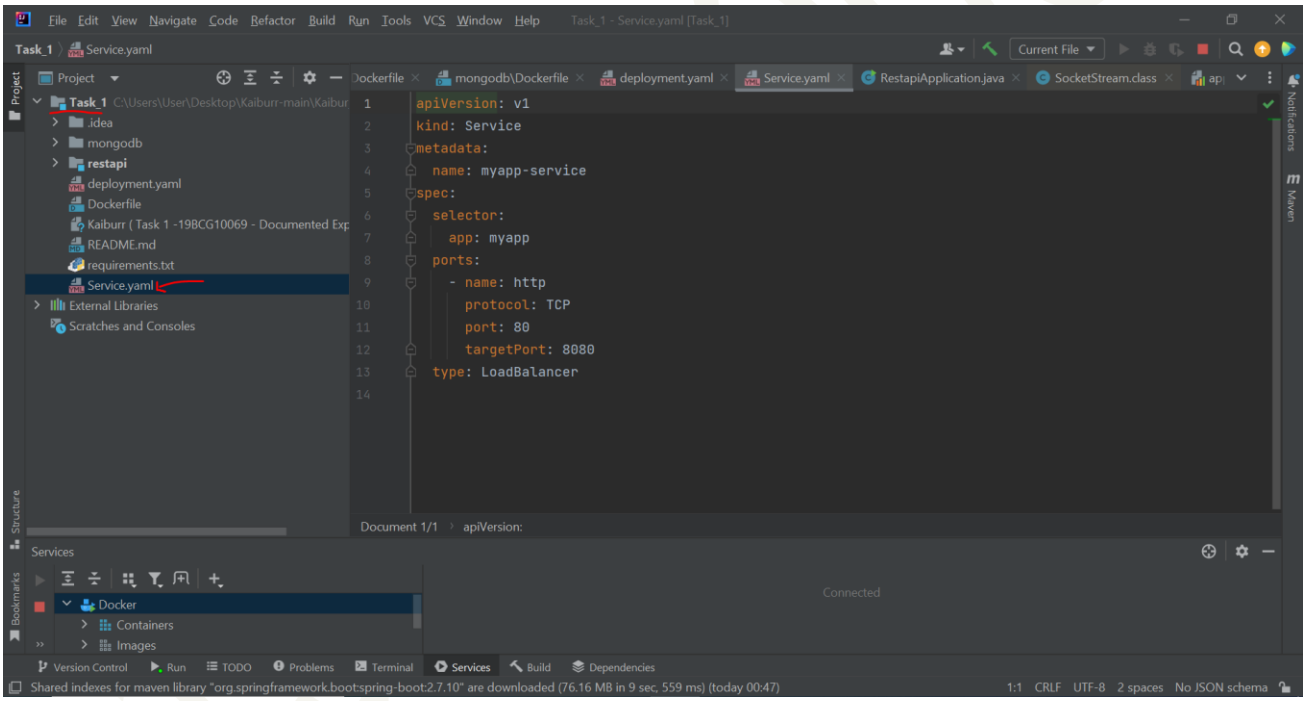
- In this deployment manifest, we have specified the ***name of the deployment, the number of replicas, and the container image*** to be used for the deployment. We have also specified the name of the ***MongoDB service*** that we will create in the next step.

- We have also defined an ***environment variable MONGO_URI*** which contains the URL of the MongoDB database. Here, we have specified the name of the MongoDB service as mongodb and the name of the database as ***myapp-db***.

b. ***Creating 'Service.yaml':***

i. Create a ***new file in the root directory*** of your project called '***Service.yaml***' and open it in a ***text editor (like VsCode)***.
"***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***"

ii. Inside this "***Task1***" Create a "***Service.yaml"*** with the '***Following code***':

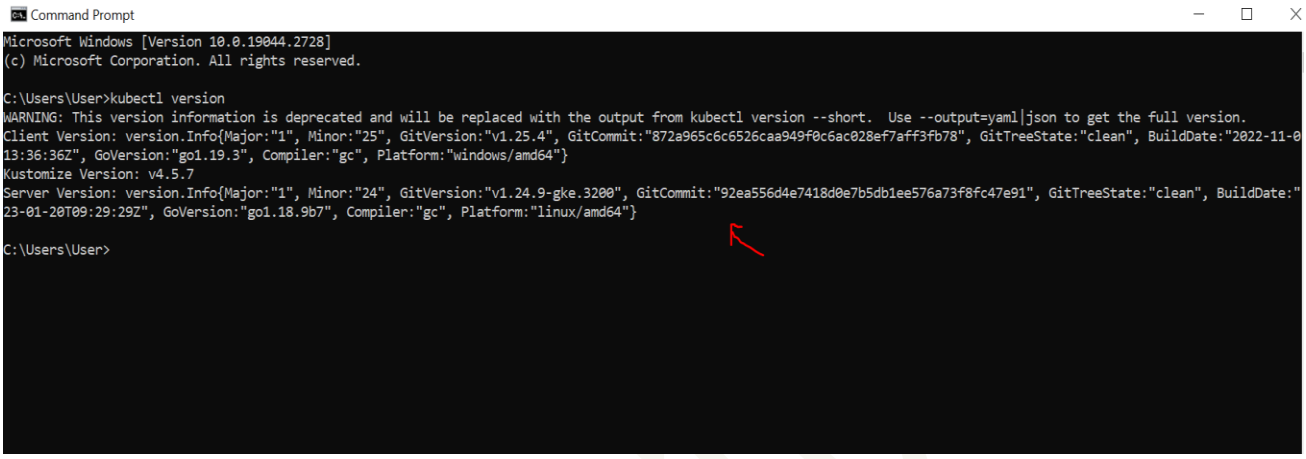


iii. ***"Save and close the Service.yaml"*** file.

- This manifest ***creates a Kub"ernetes service*** that ***exposes your application on port "80"***, and forwards ***traffic to port 8080*** on the pods that match the "***app: myapp"*** label selector (which should match the pods created by your ***deployment***).

3. **Installing kubectl and Create Kubernetes cluster**:
    a. To "***install kubectl***", you can follow the instructions provided by the Kubernetes documentation, which can be found here: ***https://kubernetes.io/docs/tasks/tools/install-kubectl/***. The installation process depends on your operating system, so make sure to select the appropriate guide for your system.

❖ ***"Successful Installation"*** can be checked as follows('***kubectl –version'***) :



    b. ***Creating "Kubernetes Cluster " (kaiburrcluster - V. IMP.):***
        i. To create a ***Kubernetes cluster***, you have several options depending on your needs and resources. Here are a few options to consider:

- ***Create a cluster using a cloud provider:*** Many cloud providers offer managed Kubernetes services, such as Amazon EKS, Google Kubernetes Engine, and Microsoft Azure Kubernetes Service. These services can help you quickly create and manage a Kubernetes cluster in the cloud.

- ***Create a cluster using Kubernetes distributions***: Kubernetes distributions, such as Rancher and OpenShift, provide preconfigured Kubernetes clusters that you can deploy on your own infrastructure.

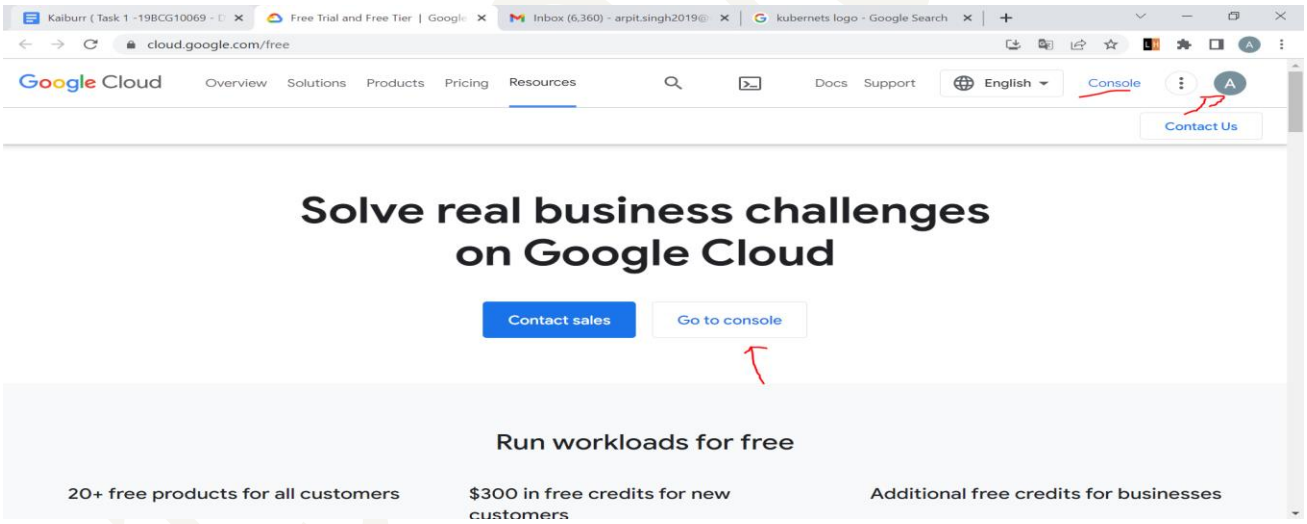
- ***Create a cluster using Kubernetes installers***: Kubernetes installers, such as kubeadm and kops, allow you to create and configure a Kubernetes cluster on your own

infrastructure. These tools require more setup and configuration than cloud providers or Kubernetes distributions.

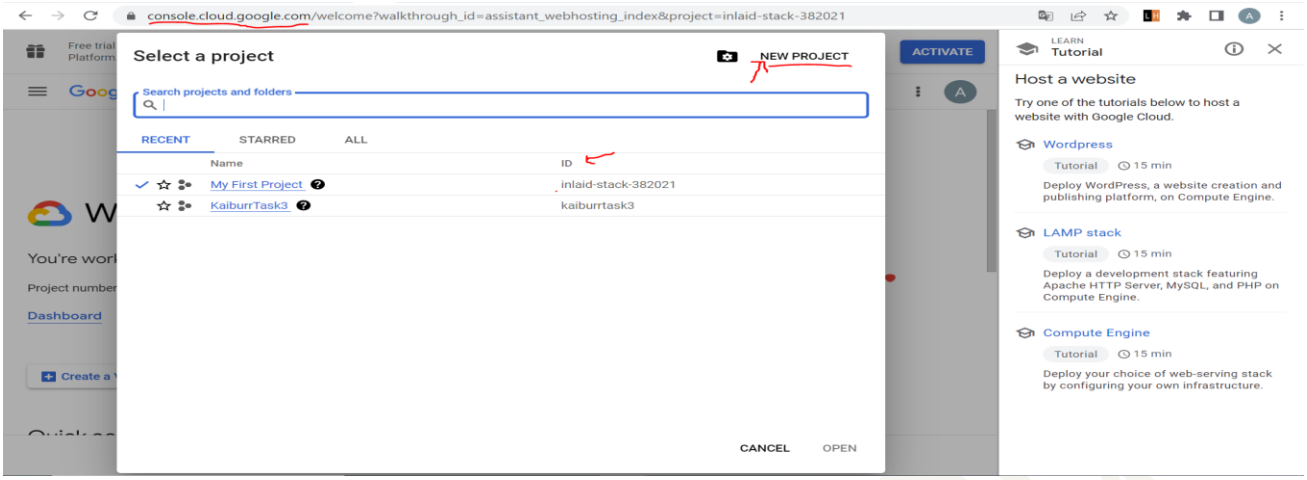
Google Kubernetes Engine (GKE):

➔ One of the ***easiest ways*** to create a Kubernetes cluster is by using a managed Kubernetes service provided by a cloud provider. For the ***purpose of this example***, I will show you how to create a ***cluster using Google Kubernetes Engine*** (GKE).

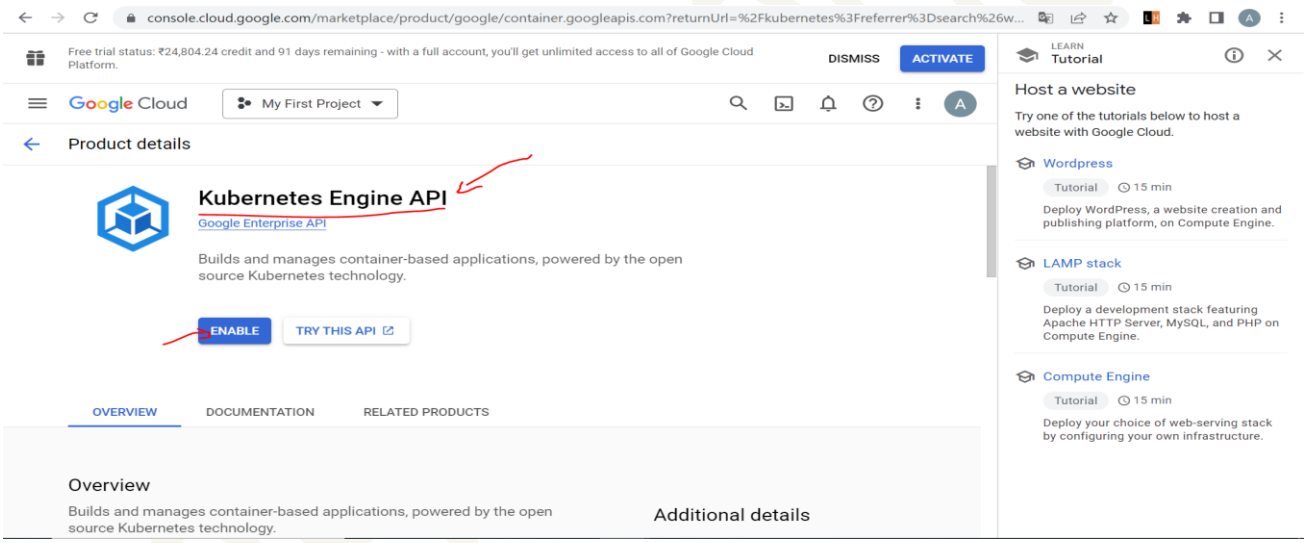
➔ Step 1: Create a Google Cloud Platform ***(GCP) account*** if you don't already have one. You can sign up for a free trial at ***https://cloud.google.com/free***.



➔ Step 2 : Once you have a GCP account, go to the '***GCP Console***' at ***https://console.cloud.google.com/*** and create a ***new project***.



➔ Step 3 : Enable the ***Kubernetes Engine API*** by going to the ***API Library*** and searching for "***Kubernetes Engine API***". Then, click on "***Enable***".

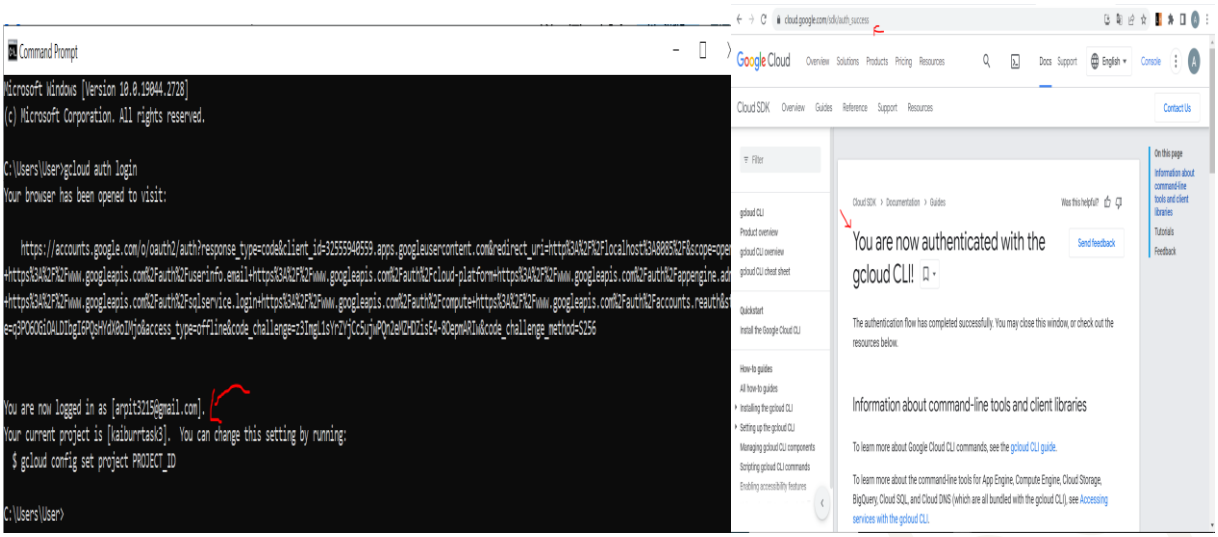


➔ Step 4 : Install the "***Google Cloud SDK***" on your local machine. You can download it from ***https://cloud.google.com/sdk/docs/install***.

➔ Step 5 : Once the SDK is installed, open a terminal and ***authenticate with your GCP account*** by running the command:

- `gcloud auth login`

Google Cloud



➔ Step 6 : '***Set the default project***' for the SDK by running the command:

- `gcloud config set project <PROJECT_ID>`
- Replace '***<PROJECT_ID>***' with the '**ID**' of the project you created in '***Step 2***'. Ex : "***gcloud config set project kaiburrtask3***"

```
C:\Users\User>gcloud config set project kaiburrtask3
Updated property [core/project].

C:\Users\User>
```

➔ Step 7 : ***Create*** a "***Kubernetes cluster***" using the following command:

- `gcloud container clusters create <CLUSTER_NAME> --num-nodes=<NUM_NODES> --zone=<ZONE>`

- Replace ***<CLUSTER_NAME>*** with the name you want to give your ***cluster***, ***<NUM_NODES>*** with the ***number of nodes*** you want to create, and ***<ZONE>*** with the ***Google Cloud zone*** where you want to create the cluster. Ex : "***gcloud container clusters create kaiburrtask3 –num-nodes=1 --zone=asia-south1-a***"

- This creates a cluster named kaiburtask3 with 1 node and zone as asia-south1-a.

Google Cloud

➔ Step 8 : After a few minutes, your cluster should be ready. You can ***check the status*** of your cluster by running the ***command***:

- `gcloud container clusters list`
- This will ***display a list of all the clusters in your project***, along with their status.

```
C:\Users\User>gcloud container clusters list
NAME           LOCATION       MASTER_VERSION   MASTER_IP       MACHINE_TYPE  NODE_VERSION     NUM_NODES  STATUS
kaiburrcluster  asia-south1-a  1.24.9-gke.3200  35.200.198.145  e2-medium     1.24.9-gke.3200  1          RUNNING

C:\Users\User>
```

4. **Deploying our Application to 'Pods'**:
   a. ***Retrieving Cluster Credentials:***
      i. Retrieve the cluster credentials: Depending on how you created your Kubernetes cluster, you may need to ***retrieve the cluster credentials to authenticate with the cluster.*** You can retrieve the credentials by using the ***cloud provider's CLI tool*** or by downloading the credentials file from the Kubernetes dashboard.

      ii. "***gcloud container clusters get-credentials [CLUSTER_NAME] --zone [ZONE]***", replacing ***[CLUSTER_NAME]*** with the name of your ***GKE*** cluster and ***[ZONE]*** with the zone where your cluster is ***located***.

      iii. `gcloud container clusters get-credentials kaiburrcluster --zone asia-south1-a`

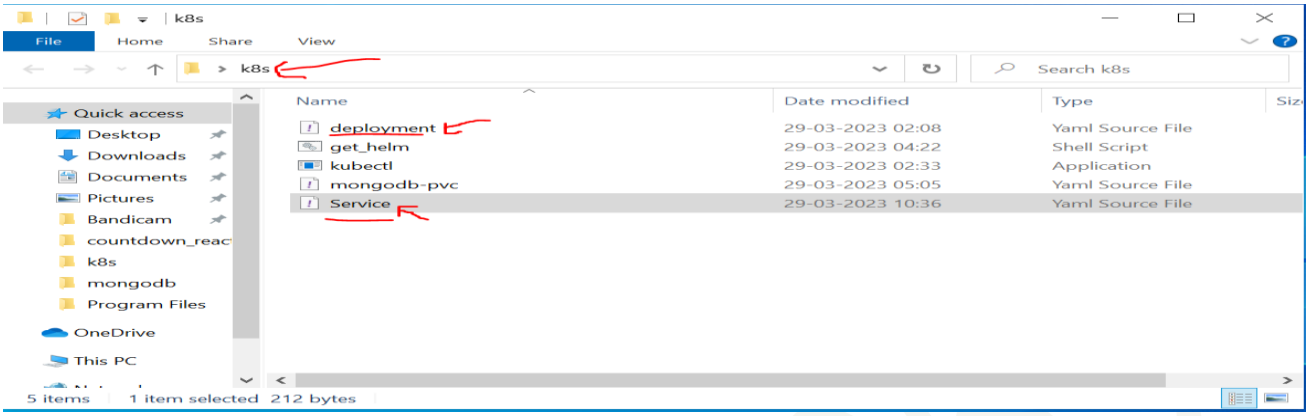
      iv. This command downloads the cluster credentials and a***dds them to your local kubectl configuration file***, which enables you to ***authenticate*** with the cluster and execute Kubernetes commands .

```
C:\Users\User>
C:\Users\User>gcloud container clusters get-credentials kaiburrcluster --zone asia-south1-a
Fetching cluster endpoint and auth data.
kubeconfig entry generated for kaiburrcluster.

C:\Users\User>
```

❖ ***Folder Creation*** (Optional - for Convenience) :

➢ I created a ***Folder named "ks8"*** having all the ***yaml files*** from which the ***pods and deployment*** had to be created. (Service.yaml/deployment.yaml)



➢ ***get_helm and mongodb-pvc discussed later***.

b. ***Deploying the App (deployment.yaml):***
   i. Go to the "***k8s***" folder or wherever you have stored "***deployment.yaml***".
      - cd "***C:\Users\User\Desktop\k8s***"
      - cd '***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***'
   ii. Run the following command to ***create the deployment***:
      - "***kubectl apply -f deployment.yaml***"

```
C:\Users\User\Desktop\k8s>kubectl apply -f deployment.yaml
deployment.apps/myapp-deployment unchanged
```

   iii. Like this our ***Application is deployed*** on the remote cloud server.

c. ***Deploying the Service (Service.yaml):***
   i. Go to the "***k8s***" folder or wherever you have stored "***Service.yaml***".
      - cd "***C:\Users\User\Desktop\k8s***"
      - cd '***C:\Users\User\Desktop\Kaiburr-main\Kaibur_Tasks\Task_1***'
   ii. Run the following command to ***create the deployment***:
      - "***kubectl apply -f Service.yaml***"

```
C:\Users\User\Desktop\k8s>kubectl apply -f Service.yaml
service/myapp-service configured
```

iii. Like this our ***Service is deployed*** on the remote cloud server.

- ***Verify that the deployment and service are created*** and running by running the following command:
  - ***"kubectl get all"***

```
C:\Users\User>kubectl get all
NAME                                    READY   STATUS             RESTARTS          AGE
pod/mongodb-849d55bbb6-nn4cm            1/1     Running            0                 8h
pod/myapp-deployment-7fcf74c859-29vzk   0/1     CrashLoopBackOff   120 (3m36s ago)   9h

NAME                     TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
service/kubernetes       ClusterIP      10.20.0.1       <none>         443/TCP        10h
service/mongodb          ClusterIP      10.20.2.21      <none>         27017/TCP      9h
service/myapp-service    LoadBalancer   10.20.13.191    35.244.15.67   80:30370/TCP   9h

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb             1/1     1            1           9h
deployment.apps/myapp-deployment    0/1     1            0           9h

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-68b4fd77c6            0         0         0       9h
replicaset.apps/mongodb-849d55bbb6            1         1         1       8h
replicaset.apps/myapp-deployment-7fcf74c859   1         1         0       9h

C:\Users\User>
```

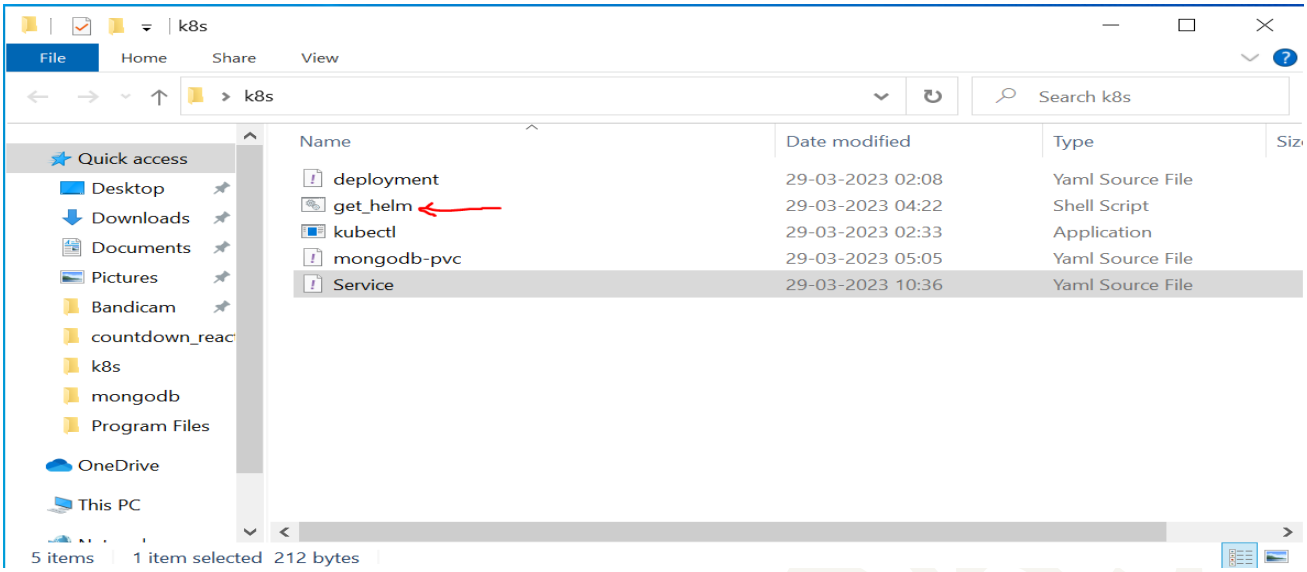
  - This command will show you a ***list of all the resources*** in your Kubernetes cluster. You should see ***your deployment and service listed***.

5. **Deploy MongoDB to the cluster:**

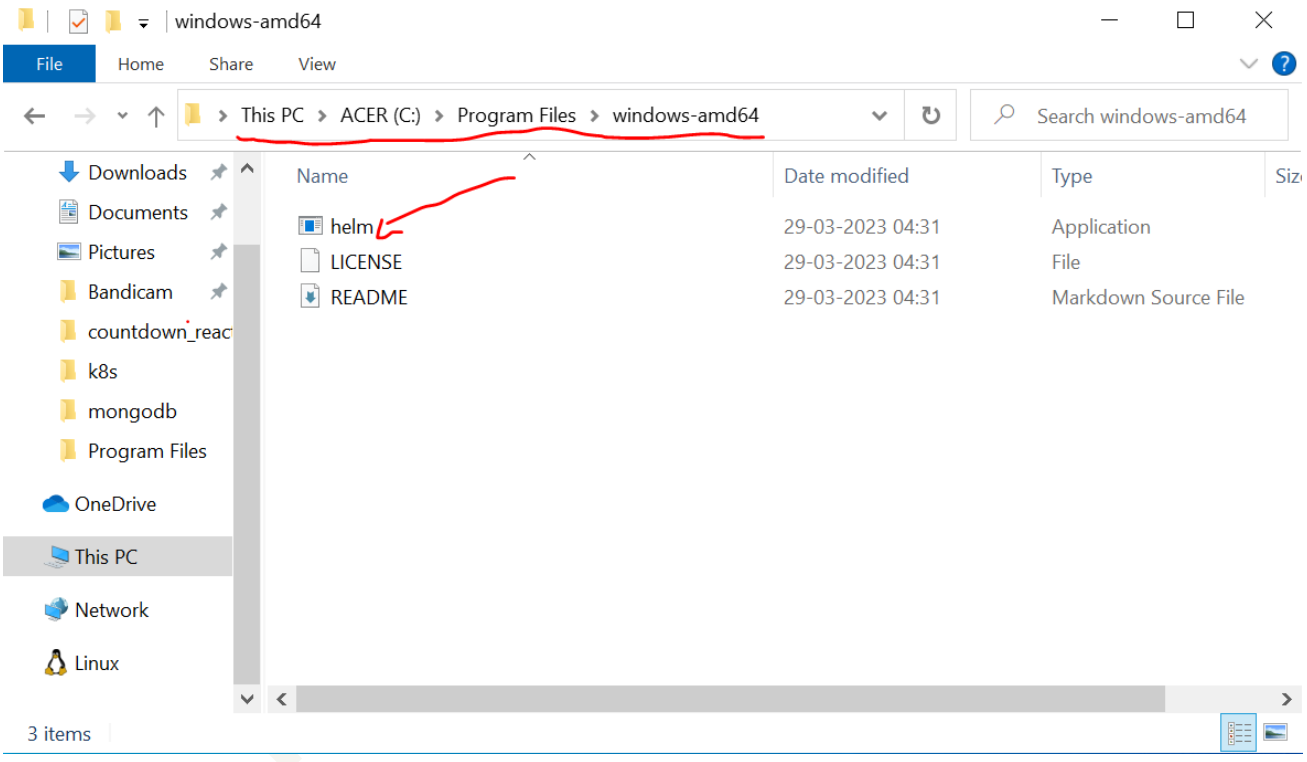
   To deploy ***MongoDB to the cluster***, you can use a community ***Helm chart***. Helm is a package manager for Kubernetes that allows you to ***easily install and manage applications***. You can install Helm by following the instructions on the Helm website.

   a. ***Installing Helm & Adding MongoDB:***

      i. ***'Install Helm'*** by following the instructions on the Helm website: ***https://helm.sh/docs/intro/install/***. I downloaded the p***ackage and unzipped it in the "k8s"*** folder that we created earlier. I did so that all these files can be put together for convenience.



ii. Go to the Directory where '***helm.exe***' is installed: (***C:\Program Files\windows-amd64***)



iii. Once Helm is installed, add the '***Bitnami repository***' by running the following command:

- '***helm repo add bitnami https://charts.bitnami.com/bitnami***'

```
C:\Program Files\windows-amd64>helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
```

iv. ***Update the Helm*** repository by running the following command:

- '***helm repo update***'

```
C:\Program Files\windows-amd64>helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
```

v. ***Update the Helm*** repository by running the following command:

- '***helm repo update***'

```
C:\Program Files\windows-amd64>helm install mongodb bitnami/mongodb
NAME: mongodb
LAST DEPLOYED: Wed Mar 29 04:41:32 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: mongodb
CHART VERSION: 13.9.3
APP VERSION: 6.0.5

** Please be patient while the chart is being deployed **

MongoDB&reg; can be accessed on the following DNS name(s) and ports from within your cluster:

    mongodb.default.svc.cluster.local

To get the root password run:

    export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace default mongodb -o jsonpath="{.data.mongodb-root-password}" | base64 -d)

To connect to your database, create a MongoDB&reg; client container:

    kubectl run --namespace default mongodb-client --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" --image docker.io/bitnami/mongodb:6
0.5-debian-11-r1 --command -- bash

Then, run the following command:
    mongosh admin --host "mongodb" --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/mongodb 27017:27017 &
    mongosh --host 127.0.0.1 --authenticationDatabase admin -p $MONGODB_ROOT_PASSWORD
```

- ***This command installs the latest version of the MongoDB chart from the Bitnami repository.***

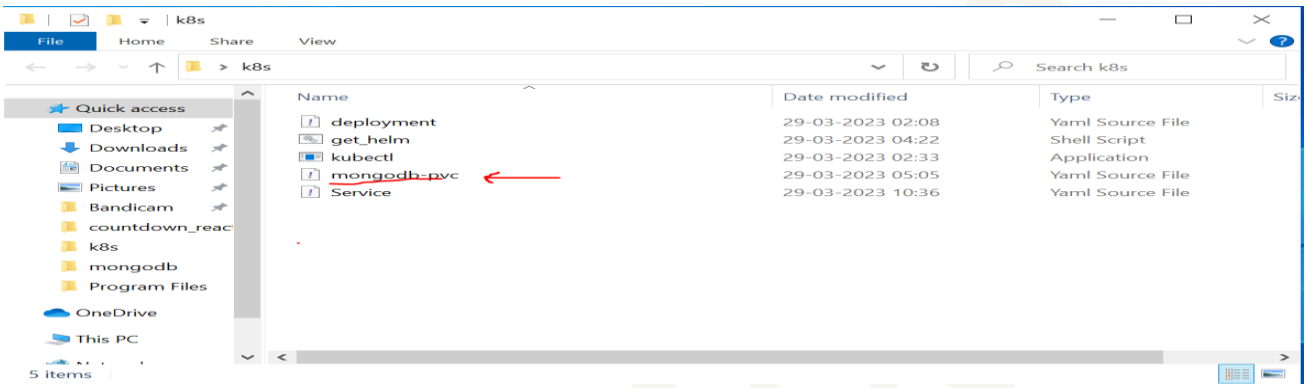
b. ***Configure persistent storage:***

i. Finally, you need to c***onfigure persistent storage for MongoDB***. This ensures that the ***data stored in the database is not lost*** when the pod is deleted. To configure persistent storage, you can create a

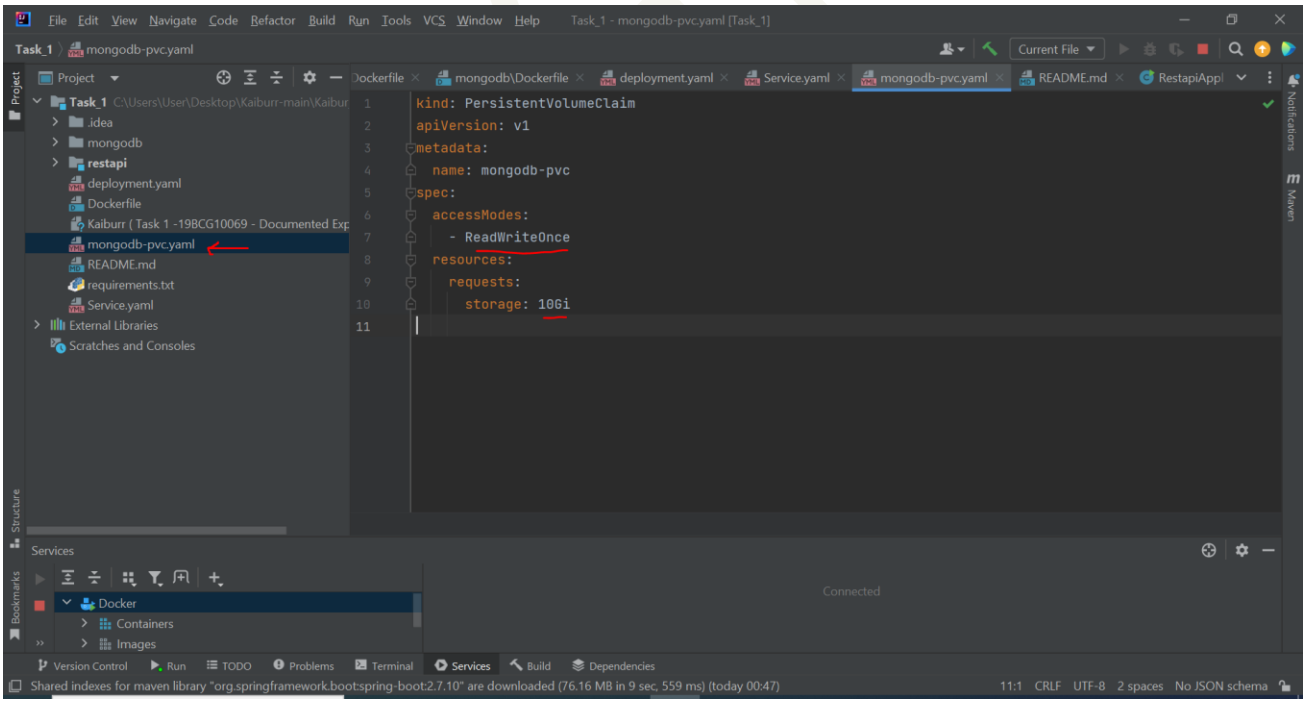
PersistentVolumeClaim and attach it to the MongoDB pod. The PersistentVolumeClaim ***specifies the storage requirements***, and the PersistentVolume provides the actual storage.

ii. ***Create*** a PersistentVolumeClaim ***(PVC) YAML file***:

- ***Navigate to "k8s" Folder*** or wherever all the ***YAML files*** are stored.



- Create a file named "***mongodb-pvc.yaml***" and add the **following YAML code** to it:



- This ***YAML code creates a PersistentVolumeClaim named mongodb-pvc with ReadWriteOnce access mode and requests 10Gi of storage***.

iii. ***Apply*** PersistentVolumeClaim ***(PVC) - mongodb-pvc.yaml***:

- If you used ***Helm to deploy MongoDB***, you can modify the deployment using the ***following command***:
- "***helm upgrade --install mongodb bitnami/mongodb --set persistence.existingClaim=mongodb-pvc***"

```
C:\Program Files\windows-amd64>helm upgrade --install mongodb bitnami/mongodb --set persistence.existingClaim=mongodb-pvc
Release "mongodb" has been upgraded. Happy Helming!
NAME: mongodb
LAST DEPLOYED: Wed Mar 29 05:17:38 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
CHART NAME: mongodb
CHART VERSION: 13.9.3
APP VERSION: 6.0.5

** Please be patient while the chart is being deployed **

MongoDB&reg; can be accessed on the following DNS name(s) and ports from within your cluster:

    mongodb.default.svc.cluster.local

To get the root password run:

    export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace default mongodb -o jsonpath="{.data.mongodb-root-password}" | base64 -d)

To connect to your database, create a MongoDB&reg; client container:

    kubectl run --namespace default mongodb-client --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" --image docker.io/bitnami/mongodb:6.
0.5-debian-11-r1 --command -- bash

Then, run the following command:
    mongosh admin --host "mongodb" --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/mongodb 27017:27017 &
    mongosh --host 127.0.0.1 --authenticationDatabase admin -p $MONGODB_ROOT_PASSWORD
```

- This command ***upgrades the existing MongoDB deployment with the bitnami/mongodb chart and sets the persistence.existingClaim value to mongodb-pvc***, which tells Kubernetes to use the mongodb-pvc PersistentVolumeClaim for MongoDB's persistent storage.

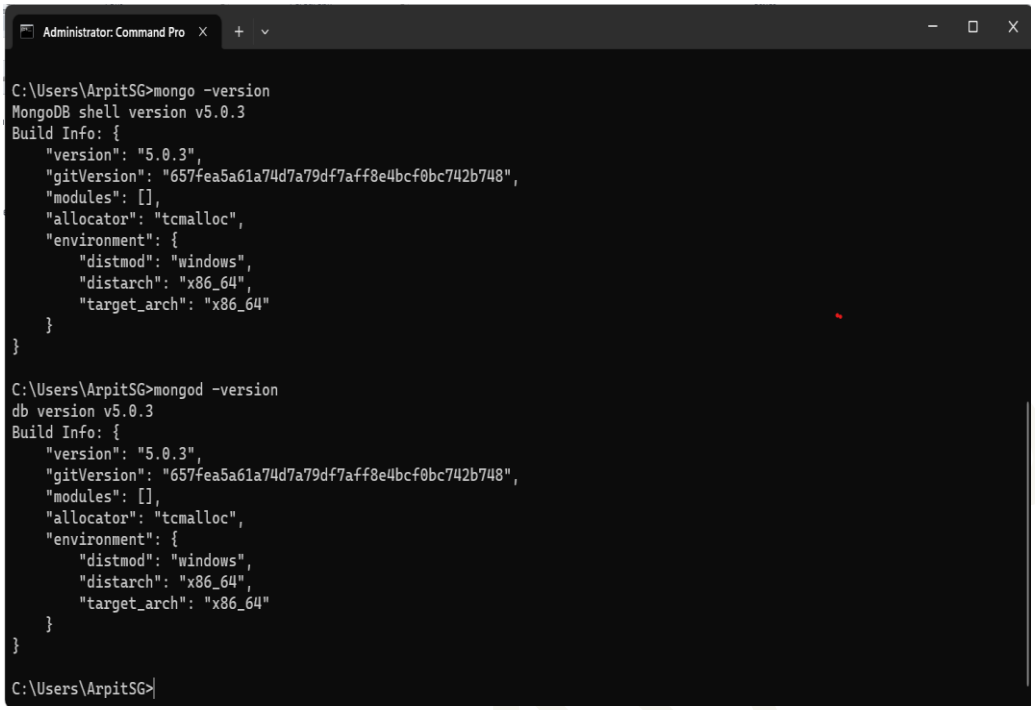
That's it! With these steps, you have configured ***persistent storage for MongoDB*** on the cluster.

6. **Verification of cluster:**
    a. Run this command : `kubectl get services`

```
C:\Users\User>kubectl get service
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP      10.20.0.1      <none>         443/TCP        11h
mongodb         ClusterIP      10.20.2.21     <none>         27017/TCP      10h
myapp-service   LoadBalancer   10.20.13.191   35.244.15.67   80:30370/TCP   11h

C:\Users\User>
```

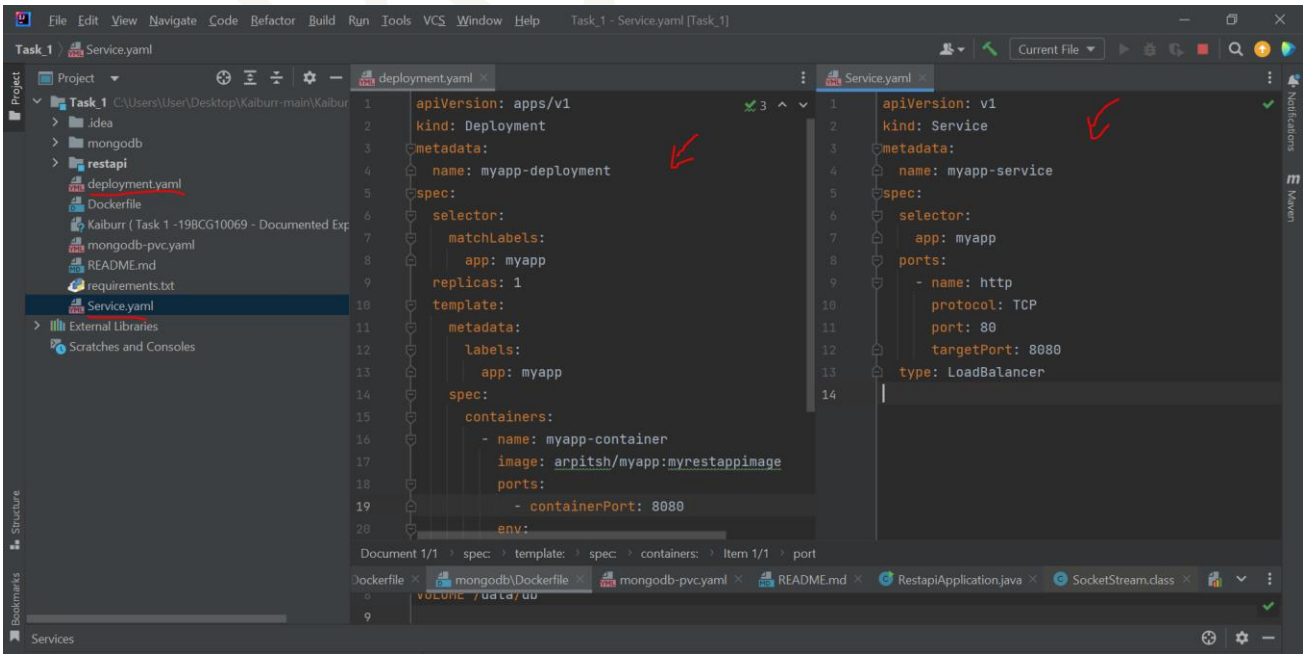
That's it! You have now created Kubernetes manifests for your application and deployed them to your Kubernetes cluster.



7. API Testing/Output:

- **you can bring your application up by applying your yaml manifests**:

  → Yes.



- **mongodb is running in a separate pod**

  → Yes.

```
C:\Users\User>kubectl get service
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP      10.20.0.1      <none>         443/TCP        11h
mongodb         ClusterIP      10.20.2.21     <none>         27017/TCP      10h
myapp-service   LoadBalancer   10.20.13.191   35.244.15.67   80:30370/TCP   11h
```

- **the application should take mongo connection details from the environment variables**

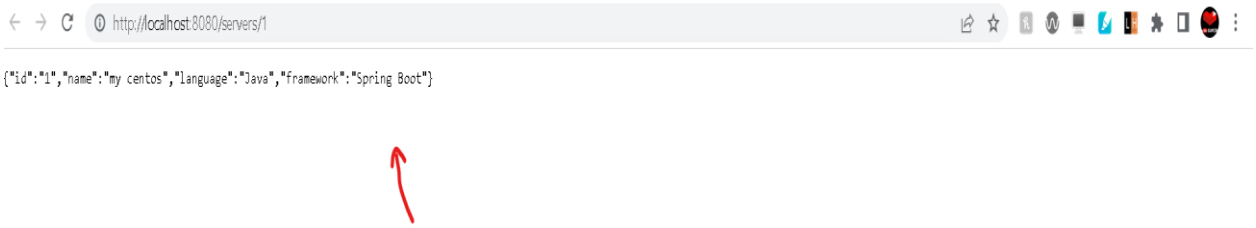
  → Yes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: arpitsh/myapp:myrestappimage
          ports:
            - containerPort: 8080
          env:
            - name: MONGO_URI
              value: "mongodb://mongodb:27017/myapp-db"
```

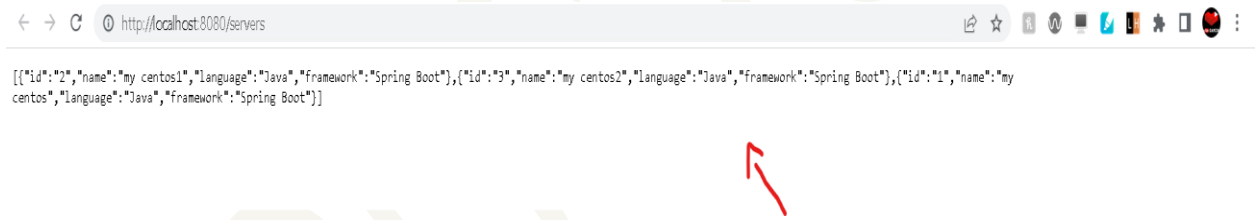
- **the app endpoints should be available from your host machine**

  → Yes.

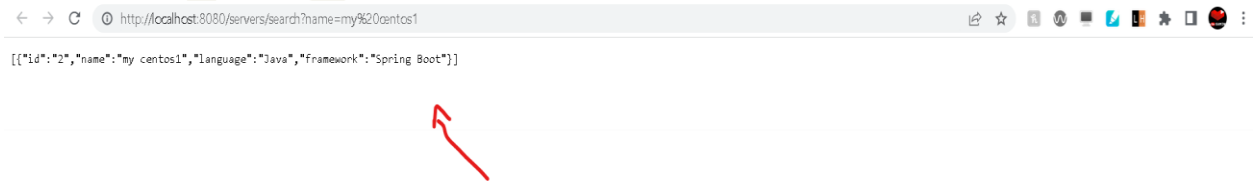
  a. To get a ***single server by ID***, you can send a ***GET*** request to ***http://localhost:8080/servers/{id}***, where {id} is the ID of the server you want to retrieve. Ex: ***http://localhost:8080/servers/1***.



b. To ***get all servers***, you can send a ***GET*** request to ***http://localhost:8080/servers.***

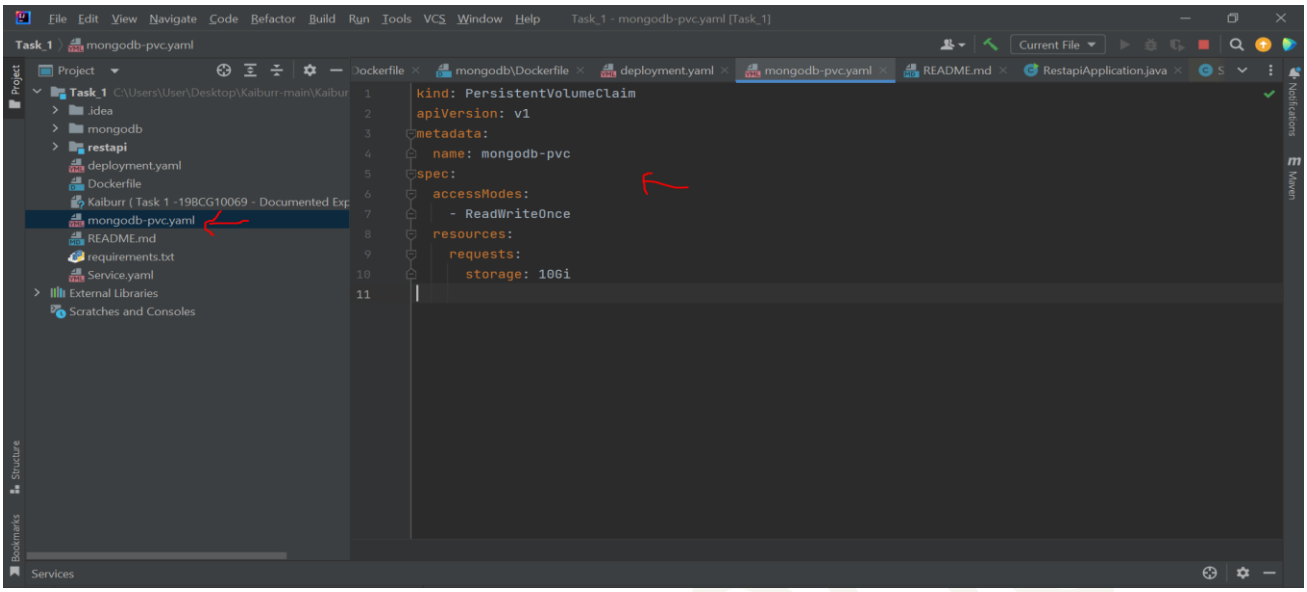


c. To ***search servers by name***, you can send a ***GET*** request to ***http://localhost:8080/servers/search?name={name}***, where {name} is the name of the server you want to search for. Ex: ***http://localhost:8080/servers/search?name=my centos1.***



- **a persistent volume should be used to store the MongoDB data**

→ Yes.



Summary of the steps to create a Deploy your Application using kubernetes (Task 3):

1. Create Dockerfiles for your application and MongoDB instance.
2. Build Docker images using the command "docker build -t <image-name>:<version> <path-to-dockerfile>".
3. Create Kubernetes manifests, including at least a deployment and a service.
4. Install Helm and deploy MongoDB to the cluster using the command "helm install mongodb bitnami/mongodb".
5. Deploy your application to the cluster using the command "kubectl apply -f <path-to-manifests>".
6. Verify the deployment by accessing the endpoints from your host machine using the command "kubectl get services".
7. Configure persistent storage for MongoDB by creating a PersistentVolumeClaim and attaching it to the MongoDB pod.

Explanation on Tools/Technology(s) Used:

- **Docker**:
  a. The purpose of using Docker in ***this project*** is to containerize the application and MongoDB instance. Docker allows developers to create lightweight, portable containers that include all the necessary dependencies and configuration to run an application. This makes it easy to deploy the application to different environments, such as a local development environment or a production server, without worrying about the differences in operating systems or system configurations. Additionally, Docker provides isolation between the application and the host system, which increases security and stability.

- **Kubernetes**:
  a. The purpose of using Kubernetes in ***this project*** is to orchestrate and manage the deployment, scaling, and management of containerized applications. Kubernetes provides a platform for automating the deployment, scaling, and management of containerized applications, including features such as self-healing, auto-scaling, and rolling updates.

  b. ***In this project***, Kubernetes is used to deploy and manage the Docker containers for the application and MongoDB instance, including scaling the number of replicas based on demand and managing the application's network connectivity and storage. Kubernetes also provides the ability to roll out updates to the application without downtime and roll back to previous versions if necessary, making it a powerful tool for managing and maintaining containerized applications in production.

❖ **Helm**:

a. ***In this project***, Helm is used as a package manager for Kubernetes. Helm allows you to easily install, manage, and upgrade applications in your Kubernetes cluster. Instead of manually configuring all the components required for the application, you can use a pre-configured chart that defines the application's deployment, service, and other necessary Kubernetes resources. The chart also allows you to customize the application configuration, such as specifying environment variables, resource limits, and storage requirements, which simplifies the process of deploying and managing the application in a Kubernetes cluster.

❖ **Google Cloud Provider**:

a. The purpose of using ***Google Cloud provider in this project*** is to host and manage the Kubernetes cluster on the Google Cloud Platform. Google Cloud provides a managed Kubernetes service called Google Kubernetes Engine (GKE) that allows developers to deploy and manage containerized applications on Google Cloud. By using GKE, developers can easily create and manage a Kubernetes cluster without having to worry about the underlying infrastructure. GKE also provides features such as automatic scaling, load balancing, and monitoring, which make it easier to manage and scale the applications running on the cluster. Additionally, GKE integrates with other Google Cloud services such as Cloud Storage, BigQuery, and Pub/Sub, which can be used to build more complex and scalable applications.

Thank You!