Signature Project: MongoDB + Python Flask Web Framework + REST API + GKE

Name: MD ROKIBUL HASAN

ID: 19900

Step1 Create MongoDB using Persistent Volume on GKE, and insert records into it

1. Create a cluster as usual on GKE

gcloud container clusters create kubia1 --num-nodes=3 --zone=us-west1-b --machine-type=n1-standard-1

Wait for the creation to finish,

```
kubeconfig entry generated for kubia1.
NAME: kubia1
LOCATION: us-west1-b
MASTER_VERSION: 1.30.5-gke.1443001
MASTER_IP: 34.168.124.78
MACHINE_TYPE: n1-standard-1
NODE_VERSION: 1.30.5-gke.1443001
NUM_NODES: 3
STATUS: RUNNING
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$
```

2. Now create a Persistent Volume:

gcloud compute disks create --size=10GiB --zone=us-west1-b mongodb

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ gcloud compute disks create --size=10GiB --zone=us-west1-b mongodb
WARNING: You have selected a disk size of under [200GB]. This may result in poor I/O performance. For more information, see: https://d
sks#performance.
Created [https://www.googleapis.com/compute/v1/projects/summer-foundry-441517-q7/zones/us-west1-b/disks/mongodb].
NAME: mongodb
ZONE: us-west1-b
SIZE_GB: 10
TYPE: pd-standard
STATUS: READY

New disks are unformatted. You must format and mount a disk before it
can be used. You can find instructions on how to do this at:

https://cloud.google.com/compute/docs/disks/add-persistent-disk#formatting
```

- **Create a Persistent Volume (PV)**

First, create a YAML file for the Persistent Volume that references the disk you created in Google Cloud. The PV will be configured to use the mongodb disk.

Create a file named mongodb-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
```

```
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

- **Apply the PV Configuration**

Run the following command to apply the Persistent Volume configuration:

kubectl apply -f mongodb-pv.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f mongodb-pv.yaml
persistentvolume/mongodb-pv created
```

- **Create a Persistent Volume Claim (PVC)**

Next, you need to create a **Persistent Volume Claim** (PVC) that will request storage from the PV. Create a YAML file for the PVC, e.g., mongodb-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

- **Apply the PVC Configuration**

Run the following command to apply the Persistent Volume Claim:

kubectl apply -f mongodb-pvc.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f mongodb-pvc.yaml
persistentvolumeclaim/mongodb-pvc created
```

- **Verify the PV and PVC**

To check if your PV and PVC were created successfully and bound to each other, you can run:

kubectl get pv

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
mongodb-pv   10Gi       RWO            Retain           Available                          <unset>                          87s
```

kubectl get pvc

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl get pvc
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mongodb-pvc   Pending                                      standard-rwo   <unset>                 42s
```

3. Now create a mongodb deployment with this yaml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo:latest
          volumeMounts:
            - mountPath: /data/db
              name: mongodb-storage
      volumes:
        - name: mongodb-storage
          persistentVolumeClaim:
            claimName: mongodb-pvc
```

- Now apply mongodb-deployment.yaml file

  kubectl apply -f mongodb-deployment.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb created
```

4. Check if the deployment pod has been successfully created and started running

kubectl get pods

Please wait until you see the STATUS is running, then you can move forward

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
ml-app-deployment-5d8bc5876d-j4q2w  1/1     Running   0          89m
mongodb-f85ff6cdd-mpk6f             1/1     Running   0          38s
```

5. Create a service for the mongoDB, so it can be accessed from outside

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb  # This should match the label of your MongoDB deployment
  ports:
    - protocol: TCP
      port: 27017  # Default port for MongoDB
      targetPort: 27017  # Port where the MongoDB container is listening
  type: LoadBalancer  # This will expose the service externally
```

- Now apply mongodb-service.yaml

  kubectl apply -f mongodb-service.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f mongodb-service.yaml
service/mongodb-service created
```

6. Wait couple of minutes, and check if the service is up

kubectl get svc

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl get svc
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)           AGE
kubernetes        ClusterIP      34.118.224.1    <none>          443/TCP           15h
ml-app-service    LoadBalancer   34.118.239.59   35.185.243.85   80:32596/TCP      15h
mongodb-service   LoadBalancer   34.118.230.55   34.168.13.181   27017:30381/TCP   2m1s
```

Please wait until you see the external-ip is generated for mongodb-service, then you can

move forward

7. Now try and see if mongoDB is functioning for connections using the External-IP

kubectl exec -it mongodb-f85ff6cdd-mpk6f -- bash

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl exec -it mongodb-f85ff6cdd-mpk6f -- bash
root@mongodb-f85ff6cdd-mpk6f:/#
```

Now you are inside the mongodb deployment pod

Try

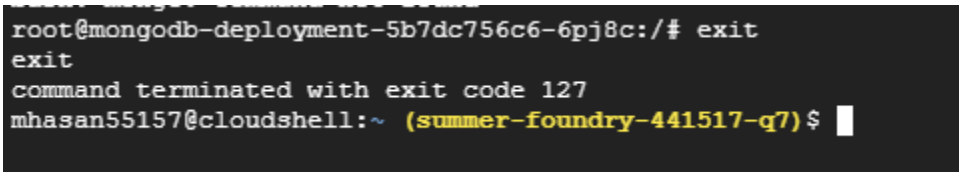
mongosh External-IP

You should see something like this, which means your mongoDB is up and can be

```
root@mongodb-deployment-5b7dc756c6-df9js:/# mongosh 34.102.74.65
Current Mongosh Log ID: 6736392bfb241f2d25fe6910
Connecting to:          mongodb://34.102.74.65:27017/?directConnection=true&appName=mongosh+2.3.2
Using MongoDB:          8.0.3
Using Mongosh:          2.3.2
mongosh 2.3.3 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
```

accessd using the 34.102.74.65

8. Type exit to exit mongodb and back to our google console



9. We need to insert some records into the mongoDB for later use. We will use node to do so.

• First Make sure node is installed.

• npm install mongodb

```
const { MongoClient } = require('mongodb');

async function insertStudents() {
  try {
    const url = "mongodb://34.102.74.65/studentdb";  // Updated the IP address
    const client = await MongoClient.connect(url, {
      useNewUrlParser: true,
      useUnifiedTopology: true  // Ensured the correct option is used
    });
    console.log("Connected successfully to MongoDB");

    const db = client.db("studentdb");

    // Data to insert
    const docs = [
      { student_id: 11111, student_name: "Bruce Lee", grade: 84 },
      { student_id: 22222, student_name: "Jackie Chen", grade: 93 },
      { student_id: 33333, student_name: "Jet Li", grade: 88 }
    ];

    // Insert documents into the collection
    const result = await db.collection("students").insertMany(docs);
    console.log(result.insertedCount + " documents inserted");

    // Fetch the student with student_id 11111
    const student = await db.collection("students").findOne({ student_id: 11111 });
    console.log("Found student:", student);

    await client.close();
    console.log("Connection closed");

  } catch (err) {
    console.error("Error:", err);
  }
}

insertStudents();
```

```
n the next major version
Connected successfully to MongoDB
3 documents inserted
Found student: {
  _id: new ObjectId('67365e19d2f3efab17bf6f4e'),
  student_id: 11111,
  student_name: 'Bruce Lee',
  grade: 84
}
Connection closed
```

Step 2: StudentServer Deployment

Having successfully configured the MongoDB service, the next phase involves the creation and deployment of the studentServer.js, which will retrieve records from the MongoDB instance. This server will be deployed on Google Kubernetes Engine for seamless scalability and management.

1.Create studentServer.js file

```
var http = require("http");
var url = require("url");
var mongodb = require("mongodb");
const { MONGO_URL, MONGO_DATABASE } = process.env;
var MongoClient = mongodb.MongoClient;
var uri = `mongodb://${MONGO_URL}/${MONGO_DATABASE}`;

var server = http.createServer(function (req, res) {
  var parsedUrl = url.parse(req.url, true);
  var student_id = parseInt(parsedUrl.query.student_id);

  if (/^\/api\/score/.test(req.url)) {
    MongoClient.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true }, function (err,
client) {
      if (err) throw err;
      var db = client.db("studentdb"); // Changed "studentdb" to MONGO_DATABASE for
consistency
      db.collection("students").findOne({ student_id: student_id }, (err, student) => {
        if (err) {
          throw new Error(err.message); // Fixed error handling
        }
        if (student) {
          res.writeHead(200, { "Content-Type": "application/json" });
          res.end(JSON.stringify(student) + "\n");
        } else {
          res.writeHead(404);
          res.end("Student Not Found \n");
        }
        client.close(); // Fixed typo: 'Client' -> 'client'
      });
    });
  } else {
    res.writeHead(404);
    res.end("Wrong url, please try again\n");
```

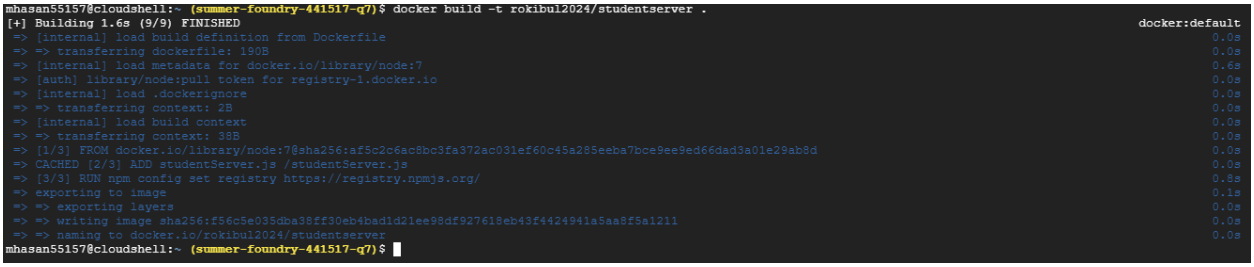
```
  }
});

server.listen(8080, () => {
  console.log("Server is listening on port 8080");
});
```

2. Create Dockerfile with the above node server.

```
FROM node:7

ADD studentServer.js /studentServer.js

ENTRYPOINT ["node", "studentServer.js"]

RUN npm config set registry https://registry.npmjs.org/
```

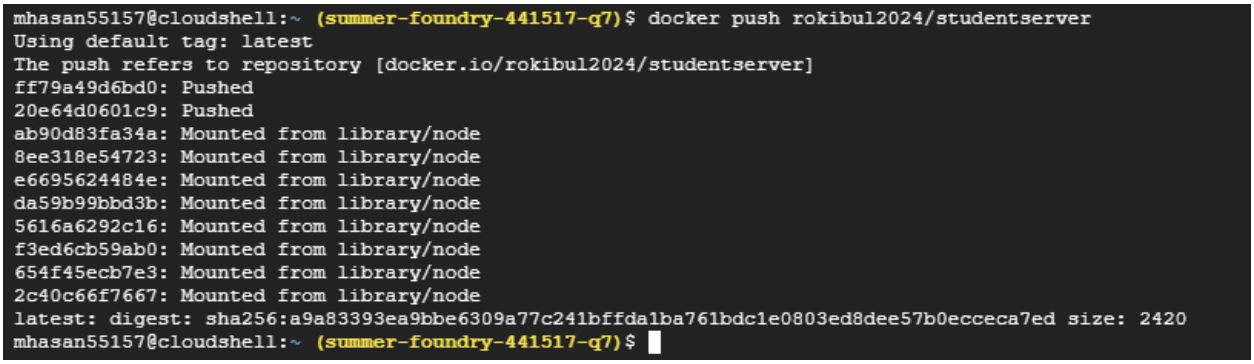
3. Build the studentserver docker image.

docker build -t rokibul2024/studentserver .



4. Push the docker image.

docker push rokibul2024/studentserver



Step 3: Bookshelf REST API Implementation and Deployment

The subsequent stage entails the creation of a Python Flask-based Bookshelf REST API.

This service will be orchestrated for deployment on Google Kubernetes Engine, aligning

with our commitment to robust, scalable cloud-native applications.

1. Begin by crafting the **bookshelf.py** file, which will serve as the foundation for your REST API, ensuring that it encapsulates the necessary endpoints and database interactions.

```
from flask import Flask, request, jsonify
from flask_pymongo import PyMongo
from flask import request
from bson.objectid import ObjectId
import socket
import os
app = Flask(__name__)
app.config["MONGO_URI"] =
"mongodb://" + os.getenv("MONGO_URL") + "/" + os.getenv("MONGO_DATABASE")
app.config['JSONIFY_PRETTYPRINT_REGULAR'] = True
mongo = PyMongo(app)
db = mongo.db
@app.route("/")
def index():
hostname = socket.gethostname()
return jsonify(
message="Welcome to bookshelf app! I am running inside {}
pod!".format(hostname)
)
@app.route("/books")
def get_all_tasks():
books = db.bookshelf.find()
data = []
for book in books:
data.append({
"id": str(book["_id"]),
"Book Name": book["book_name"],
"Book Author": book["book_author"],
"ISBN": book["ISBN"]
})
return jsonify(
data
)
@app.route("/book", methods=["POST"])
def add_book():
book = request.get_json(force=True)
db.bookshelf.insert_one({
"book_name": book["book_name"],
"book_author": book["book_author"],
"ISBN": book["isbn"]
})
```

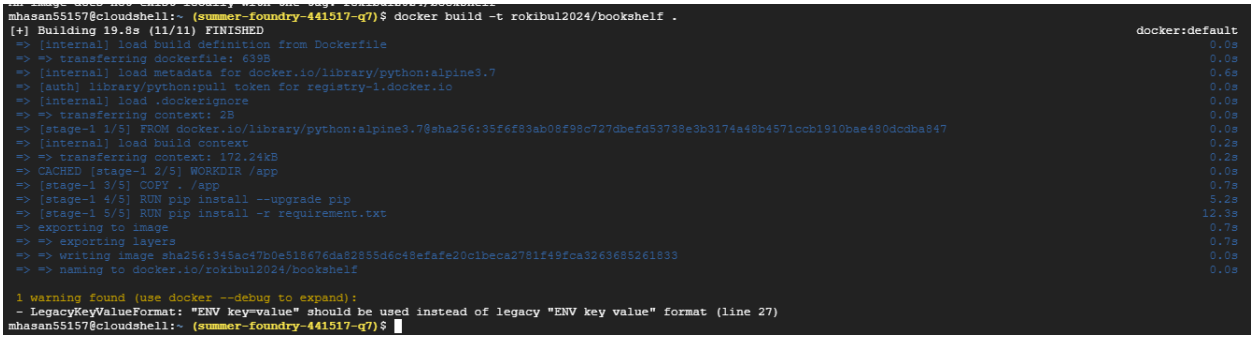
```
return jsonify(
message="Task saved successfully!"
)
@app.route("/book/<id>", methods=["PUT"])
def update_book(id):
data = request.get_json(force=True)
print(data)
response = db.bookshelf.update_many({"_id": ObjectId(id)}, {"$set":
{"book_name":
data['book_name'],
"book_author": data["book_author"],
"ISBN":
data["isbn"]
}})
if response.matched_count:
message = "Task updated successfully!"
else:
message = "No book found!"
return jsonify(
message=message
)
@app.route("/book/<id>", methods=["DELETE"])
def delete_task(id):
response = db.bookshelf.delete_one({"_id": ObjectId(id)})
if response.deleted_count:
message = "Task deleted successfully!"
else:
message = "No book found!"
return jsonify(
message=message
)
@app.route("/tasks/delete", methods=["POST"])
def delete_all_tasks():
db.bookshelf.remove()
return jsonify(
message="All Books deleted!"
)
if __name__ == "__main__":
app.run(host="0.0.0.0", port=5000)
```

2. Create a Docker file.

```
FROM python:alpine3.7
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
ENV PORT 5000
EXPOSE 5000
ENTRYPOINT [ "python3" ]
CMD [ "bookshelf.py" ]
```

3. Build the bookshelf app into a docker image.

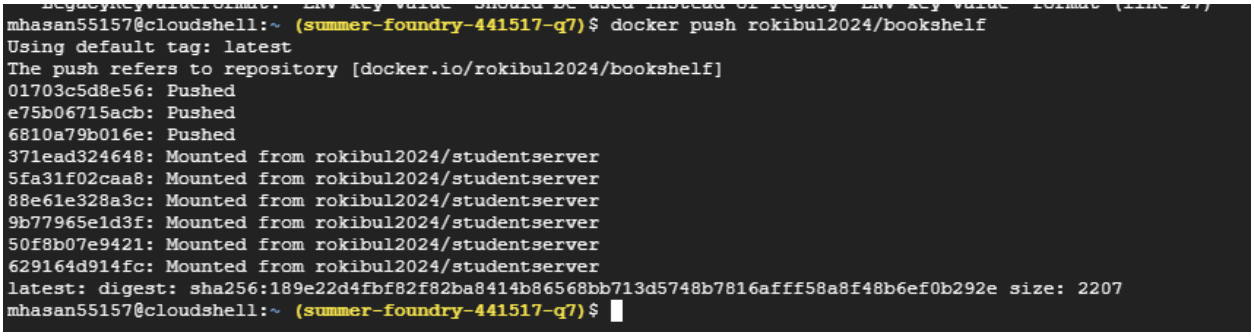
docker build -t rokibul2024/bookshelf .



Make sure this step builds successfully.

4. Push the docker image to your docker hub.

docker push rokibul2024/bookshelf



Step 4: Configuration Management

In this step, we'll establish a ConfigMap which serves as a key-value store for configuration data. This ConfigMap will store the MongoDB URL and database name, providing a streamlined way to configure both the studentServer and the Bookshelf REST API applications.

1. Create a new file entitled studentserver-configmap.yaml which will contain the necessary configuration parameters for the studentServer.

apiVersion: v1
kind: ConfigMap
metadata:
name: studentserver-config
data:
MONGO_URL: "34.102.74.65" // your mongo db external ip address
MONGO_DATABASE: mydb

2. Create a file named bookshelf-configmap.yaml

```
kind: ConfigMap
metadata:
name: bookshelf-config
data:
# SERVICE_NAME.NAMESPACE.svc.cluster.local:SERVICE_PORT
MONGO_URL: "34.168.232.150"
MONGO_DATABASE: mydb
```

Please note, the rationale behind the creation of the ConfigMap for both the studentServer and the Bookshelf REST API is to avoid the need for rebuilding the Docker images in the event of a MongoDB pod restart that results in the allocation of a new External IP. This approach ensures seamless continuity and minimizes downtime during service maintenance.

Step 5: Application Ingress Configuration

To facilitate streamlined access to both applications under a unified domain while distinguishing them by their respective paths, an ingress controller utilizing Nginx will be employed.

1. Start this process by crafting the studentserver-deployment.yaml file, which will

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  labels:
    app: studentserver-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - image: rokibul2024/studentserver
          imagePullPolicy: Always
          name: web
          ports:
            - containerPort: 8080
```

```
env:
  - name: MONGO_URL
    valueFrom:
      configMapKeyRef:
        name: studentserver-config  # Corrected ConfigMap name
        key: MONGO_URL
  - name: MONGO_DATABASE
    valueFrom:
      configMapKeyRef:
        name: studentserver-config  # Corrected ConfigMap name
        key: MONGO_DATABASE
```

2. Create bookshelf-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: bookshelf-deployment
labels:
app: bookshelf-deployment
spec:
replicas: 1
selector:
matchLabels:
app: bookshelf-deployment
template:
metadata:
labels:
app: bookshelf-deployment
spec:
containers:
- image: rokibul2024/bookshelf
imagePullPolicy: Always
name: bookshelf-deployment
ports:
- containerPort: 5000
env:
- name: MONGO_URL
valueFrom:
configMapKeyRef:
name: bookshelf-config
key: MONGO_URL
- name: MONGO_DATABASE
valueFrom:
configMapKeyRef:
name: bookshelf-config
key: MONGO_DATABASE
```

3. Create studentserver-service.yaml

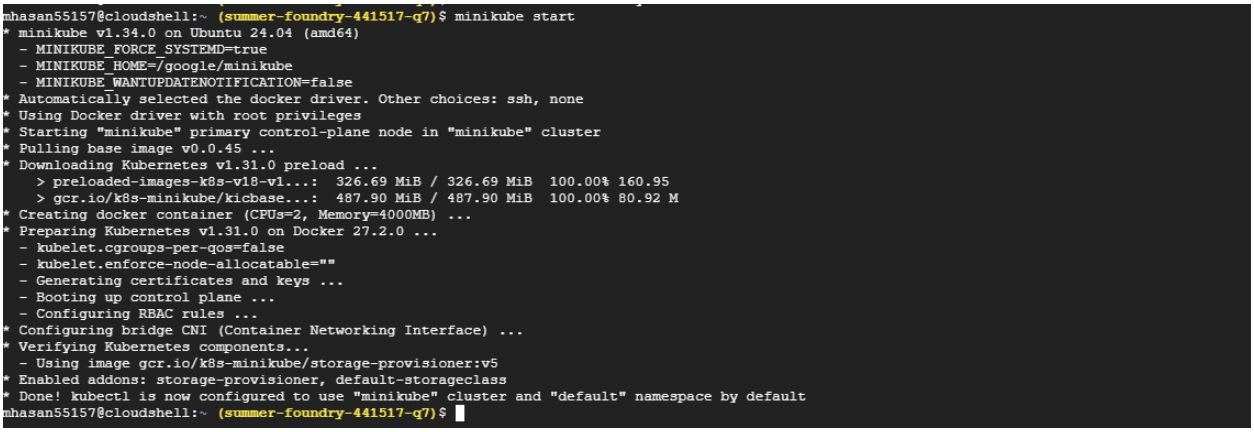
apiVersion: v1
kind: Service
metadata:
name: web
spec:
type: LoadBalancer
ports:
service port in cluster
- port: 8080
port to contact inside container
targetPort: 8080
selector:
app: web

4. Create bookshelf-service.yaml

apiVersion: v1
kind: Service
metadata:
name: bookshelf-service
spec:
type: LoadBalancer
ports:
service port in cluster
- port: 5000
port to contact inside container
targetPort: 5000
selector:
app: bookshelf-deployment

5. Start minikube

minikube start



6. Start Ingress

minikube addons enable ingress

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ minikube addons enable ingress
* ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
  - Using image registry.k8s.io/ingress-nginx/controller:v1.11.2
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
* Verifying ingress addon...
* The 'ingress' addon is enabled
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$
```

7. Create studentserver related pods and start service using the above yaml files

kubectl apply -f studentserver-deployment.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f studentserver-deployment.yaml
deployment.apps/web created
```

kubectl apply -f studentserver-configmap.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f studentserver-configmap.yaml
configmap/studentserver-config created
```

kubectl apply -f studentserver-service.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f studentserver-service.yaml
service/web created
```

8. Create bookshelf related pods and start service using the above yaml Tile

kubectl apply -f bookshelf-deployment.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f bookshelf-deployment.yaml
deployment.apps/bookshelf-deployment created
```

kubectl apply -f bookshelf-configmap.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f bookshelf-configmap.yaml
configmap/bookshelf-config created
```

kubectl apply -f bookshelf-service.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f bookshelf-service.yaml
service/bookshelf-service created
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$
```

9. Check if all the pods are running correctly

kubectl get pods

10. Create an ingress service yaml Tile called studentservermongoIngress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: server
  annotations:
```

```
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: cs571.project.com
      http:
        paths:
          - path: /studentserver(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
          - path: /bookshelf(/|$)(.*)
            pathType: Prefix
            backend:
              service:
                name: bookshelf-service
                port:
                  number: 5000
```

11. Create the ingress service using the above yaml Tile

kubectl apply -f studentservermongoIngress.yaml

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl apply -f studentservermongoIngress.yaml
Warning: path /studentserver(/|$)(.*) cannot be used with pathType Prefix
Warning: path /bookshelf(/|$)(.*) cannot be used with pathType Prefix
ingress.networking.k8s.io/server created
```

12. Check if ingress is running

kubectl get ingress

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ kubectl get ingress
NAME     CLASS   HOSTS              ADDRESS        PORTS   AGE
server   nginx   cs571.project.com  192.168.49.2   80      23s
```

13. Add Address to /etc/hosts

vi /etc/hosts

Add the address you got from above step to the end of the Tile

192.168.49.2 cs571.project.com

```
# IPv4 and IPv6 localhost aliases
127.0.0.1       localhost
::1             localhost

#
# Imaginary network.
#10.0.0.2               myname
#10.0.0.3               myfriend
#
# According to RFC 1918, you can use the following IP networks for private
# nets which will never be connected to the Internet:
#
#       10.0.0.0        -   10.255.255.255
#       172.16.0.0      -   172.31.255.255
#       192.168.0.0     -   192.168.255.255
#
# In case you want to be able to connect directly to the Internet (i.e. not
# behind a NAT, ADSL router, etc...), you need real official assigned
# numbers.  Do not try to invent your own network numbers but instead get one
# from your network provider (if any) or from your regional registry (ARIN,
# APNIC, LACNIC, RIPE NCC, or AfriNIC.)
#
169.254.169.254 metadata.google.internal metadata

10.88.0.4 cs-974534363109-default
192.168.49.2 cs571.project.com
"/etc/hosts" 36L, 1178B                                                    36,0-1        Bot
```

14. If everything goes smoothly, you should be able to access your applications

curl cs571.project.com/studentserver/api/score?student_id=11111

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl cs571.project.com/studentserver/api/score?student_id=11111
{"_id":"661729855450d30db3f0910c","student_id":11111,"student_name":"Bruce Lee","grade":84}
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$
```

15. On another path, you should be able to use the REST API with bookshelf application

curl cs571.project.com/bookshelf/books

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl http://cs571.project.com/bookshelf
{
  "message": "Welcome to bookshelf app! I am running inside bookshelf-deployment-6786768654-ss7g6 pod!"
}
```

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl cs571.project.com/bookshelf/books
[
  {
    "Book Author": "test",
    "Book Name": "123",
    "ISBN": "123",
    "id": "605d1ba7d40f50a395651765"
  }
]
```

Add a book

curl -X POST -d "{\"book_name\": \"cloud computing\",\"book_author\": \"unkown\", \"isbn\": \"123456\" }" http://cs571.project.com/bookshelf/book

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl -X POST -d "{\"book_name\": \"cloud computing\",\"book_author\": \"unkown\", \"isbn\": \"123456\" }" http://cs571.project.com/bookshelf/book
{
  "message": "Task saved successfully!"
}
```

Update a book

curl -X PUT -d "{\"book_name\": \"123\",\"book_author\": \"test\", \"isbn\": \"123updated\" }" http://cs571.project.com/bookshelf/book/id

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl -X PUT -d "{\"book_name\": \"123\",\"book_author\": \"test\", \"isbn\": \"123updated\" }" http://cs571.project.com/bookshelf/book/605d1ba7d40f50a395651765
{
  "message": "Task updated successfully!"
}
```

Delete a book

curl -X DELETE cs571.project.com/bookshelf/book/id

```
mhasan55157@cloudshell:~ (summer-foundry-441517-q7)$ curl -X DELETE cs571.project.com/bookshelf/book/605d1ba7d40f50a395651765
{
  "message": "Task deleted successfully!"
}
```